Mobile Geräte in verteilten Anwendungsumgebungen:

Ein Integrationsansatz zwischen
Abstraktion und Migration

Dissertation
zur Erlangung des Doktorgrades
am Fachbereich Informatik der Universität Hamburg

vorgelegt von

Stefan Müller–Wilken

aus Eckernförde

Hamburg 2002

Stefan Müller–Wilken
AB „Verteilte Systeme und Informationssysteme“
Fachbereich für Informatik der Universität Hamburg
Vogt–Kölln–Straße 30
D–22527 Hamburg

smueller@informatik.uni-hamburg.de

Genehmigt vom Fachbereich für Informatik der Universität Hamburg
auf Antrag von:

Betreuer und 1. Gutachter:	Prof. Dr. Winfried Lamersdorf
2. Gutachter:	Prof. Dr. Stefan Fischer
3. Gutachter:	Prof. Dr. Dietmar P.F. Möller

Hamburg, den 11. April 2002

gez. Prof. Dr.–Ing. H. Siegfried Stiehl
Dekan

Zusammenfassung

Mobile Computer dienten bislang überwiegend als isolierter Informationsspeicher und werden erst allmählich als wichtiger Baustein eines *verteilten* Anwendungssystems wahrgenommen. Gleichzeitig hat die verfügbare Information in Intranets und dem Internet rasant zugenommen. Die Erschließung mobiler Computer als Zugangspfade zu den in verteilten Systemen verfügbaren Informationen bietet somit ein großes Potential auf dem Weg zu einem allgegenwärtigen Wissensspeicher. Als hemmender Faktor ist dabei nicht nur das Fehlen effizienter Funknetze anzusehen. Insbesondere der Anwendungsentwurf gestaltet sich im Kontext mobiler Computer als sehr komplex: Die *Heterogenität* existierender Systemplattformen, deren *Geschlossenheit* gegenüber Modifikationen, stark *eingeschränkte Ressourcen*, aber auch die *Dynamik* innerhalb eines von mobilen Computern gebildeten Netzwerkes haben bislang die Nutzung derartiger Geräte zum Zugriff auf verteilte Anwendungen erschwert.

In diesem Zusammenhang ist es Ziel der vorliegenden Arbeit, Lösungswege aufzuzeigen, wie die Integration mobiler Geräte als Nutzer und Anbieter von Informationen in verteilten Anwendungssystemen erleichtert werden kann. Es wird die These aufgestellt, dass sich das hohe Abstraktionsniveau der Anwendungsebene in besonderem Maße als Basis für flexible Integrationsansätze eignet und anhand einer im Rahmen dieser Arbeit konzeptionell hergeleiteten und sodann in ihrer konkreten Umsetzung beschriebenen Integrationsumgebung der Nachweis hierzu geführt.

Die Arbeit gliedert sich dabei in zwei Teile. Im ersten Teil werden notwendige Grundlagen aufgearbeitet, die das Fundament der Arbeit bilden. Es werden unter anderem Definitionen von Mobilität vorgestellt, Ansätze zur Klassifikation von Integrationsumgebungen beschrieben und existierende Integrationsansätze einander gegenübergestellt. Vor dem Hintergrund dieser Betrachtungen wird sodann ein neuartiges Integrationskonzept hergeleitet, das basierend auf den Prinzipien von *Abstraktion* und *Migration* die Einbettung mobiler Dienstnehmer und Diensterbringer in verteilte Anwendungsysteme ermöglicht. Im zweiten Teil wird dann am Beispiel einer konkreten Umsetzung die Tragfähigkeit der entworfenen Konzepte verdeutlicht. Es werden die funktionalen Schwerpunkte der entwickelten Integrationsumgebung dargestellt, Anhaltspunkte für deren Einordnung gegeben und schließlich wesentliche Realisierungsdetails vorgestellt.

Danksagungen

An erster Stelle gilt der Dank meinem Betreuer Herrn Prof. Dr. Winfried Lamersdorf, der mich auf den Etappen meiner akademischen Laufbahn seit meinem Eintritt in das Hauptstudium über viele Jahre betreut und gefördert hat. Seinem Engagement verdanke ich jene Freiräume, die mir einen tiefen Einblick in den Forschungsbetrieb im akademischen sowie kommerziellen Umfeld dies– und jenseits des Äquators ermöglichten. Seine Arbeitsgruppe „Verteilte Systeme“ der Universität Hamburg ist über acht Jahre meine akademische Heimat gewesen.

Danken möchte ich auch meinen ehemaligen Kollegen in der Arbeitsgruppe VSYS, insbesondere Herrn Dr. Frank Griffel, der seit dem gemeinsamen Studium auf ähnlichen Wegen unterwegs war wie ich und der doch immer wieder mit überraschenden Reiseberichten zu faszinieren wusste. Mehr als einmal half er mir mit seinen Ideen und nahm sich die Zeit, diese mit mir zu entwickeln. Ebenso Herrn Dr. Mark Tuan Tu, dessen Hilfsbereitschaft mich ein ums andere Mal beeindruckte und den ich nie anders als konstruktiv erlebt habe, Herrn Dr. Frank Wienberg, dem ich ein anspruchsvolles Anwendungsszenario verdanke sowie Herrn Dr. Michael Merz, der mit der Fülle seiner Publikationen immer die Referenzklasse für mich darstellte und der so schnell eilte, dass es nicht immer leicht war zu folgen. Erwähnt seien auch die Mitglieder meiner „Hydeparc“ Projektgruppe, die mit ihren Studien– und Diplomarbeiten zum Gelingen meiner Arbeit beigetragen haben und deren Begeisterung und tatkräftige Unterstützung mich oft genug anspornten. Ihren Teil zum Ganzen beigesteuert haben Marko Dützmann (JSML / Application Shadow), Marko Kaiser (JBSA Gateway / Session Management), Hanno Müller (JiSDA Prototyp), Christian P. Kunze (Segmentierung), Matthias Ferdinand (Push ECA), Jens Fransson (JSML Layout), Robert Baric (Mobile JINI), Christian Hansen (DCC) und die übrigen „Hydeparcer“.

Ein besonders herzliches Dankeschön gebührt meinen Schwiegereltern Marlene und Klaus Wilken, die mir während der Niederschrift über Monate eine überaus produktive Arbeitsumgebung boten und mich ebenso mit moralischem Beistand wie mit dem nötigen Proviant versorgten. Beides hat unmittelbar zum Resultat beigetragen. Auch hat Herr Wilken den überwiegenden Teil des Manuskripts korrekturgelesen, eine Arbeit, um die ich ihn nicht beneidet habe. Vielen Dank auch dafür! Sollten dennoch Fehler zu finden sein, so sind diese ausschließlich mir oder dem Textsatzsystem LaTeX zuzuschreiben. Meinen Eltern Konrad und Renate Müller, die mich auch durch meine gesamte Studien– und Promotionszeit hindurch begleitet haben, danke ich dafür, dass sie mich lehrten, immer das Höchste anzustreben. Ihrer Erziehung verdanke ich, dass ich das Ziel niemals aus den Augen verlor.

Schließlich danke ich meiner Frau Susanne, die Stunden grenzenloser Euphorie und abgrundtiefer Verzweiflung gleichermaßen geduldig ertrug, die auch dann noch an mich glaubte, wenn ich es kaum mehr tat, die mein Promotionsvorhaben insbesondere in seiner Endphase zu hassen gelernt hat und die dennoch einen Teil der Korrekturen übernahm. Zu guter Letzt soll auch meine Tochter Svea nicht unerwähnt bleiben, die es immer wieder verstand, meine Prioritäten in ihrem Sinne zu ordnen. Für die letzten Monate möchte ich mich entschuldigen. Ich werde in Zukunft ausgeglichener sein und mir mehr Zeit für euch nehmen – versprochen!

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Defizite mobiler Geräte 3

1.2 Integrationsebenen 5

1.3 Fragestellung und Gang der Untersuchung 6

I Integration mobiler Geräte in verteilte Umgebungen **9**

2 Grundlagen mobiler Systeme **11**

2.1 Soziale vs. physische Mobilität 12

2.2 Mobilität in der Datenverarbeitung 12

2.2.1 Portabilität vs. Mobilität 13

2.2.2 Hardware– vs. Softwaremobilität 13

2.3 Merkmale mobiler Datenverarbeitung 15

2.3.1 Ressourcenknappheit 17

2.3.2 Eingeschränkte Sicherheit 17

2.3.3 Variierende Kommunikationsbedingungen 17

2.3.4 Eingeschränkte Benutzungsschnittstelle 18

2.3.5 Heterogenität 18

2.3.6 Hohe Marktdynamizität 18

2.3.7 Eingeschränkter Nutzungskontext 18

2.4 Klassifikationskriterien für mobile Geräte 19

2.4.1 Anbindung als Klassifikationskriterium 20

2.4.2 Geräteklassen am Beispiel 21

2.4.2.1 Portable Computer 21

2.4.2.2 Miniaturisierte PCs 22

2.4.2.3 Laptops, Notebooks und Subnotebooks 22

2.4.2.4 Persönliche Digitale Assistenten (PDAs) 23

2.4.2.5 Mobiltelefone 24
2.4.2.6 Wearables 25
2.4.2.7 Ubiquitäre Systeme 27
2.5 Zusammenfassung 28

3 Integrationsstrategien für mobile Geräte 31
3.1 Ziele der Geräteintegration 32
3.2 Bewertungskriterien für Integrationsansätze 33
3.2.1 Awareness und Adaptation 34
3.2.2 Agilität und originalgetreue Wiedergabe 39
3.3 Integrationsansätze im Überblick: eine OSI–Sicht 42
3.3.1 Bitübertragungsschicht 43
3.3.2 Sicherungsschicht 43
3.3.2.1 AIRMAIL 44
3.3.2.2 Multi Service Link 45
3.3.3 Netzwerkschicht 46
3.3.3.1 IPv4 und Mobilität 48
3.3.3.2 Triangle Routing und andere Anomalien 52
3.3.3.3 Mobile IPv6 54
3.3.4 Transportschicht 56
3.3.4.1 Beispiele 61

4 Ein generischer Integrationsansatz auf Anwendungsebene 63
4.1 Die Anwendungsebene als Integrationsfundament 63
4.2 Existierende Ansätze im Überblick 66
4.2.1 Betriebssystembasierte Integration 68
4.2.1.1 Window System X11 70
4.2.1.2 Remote Desktop Protocol 73
4.2.1.3 Virtual Network Computing 77
4.2.1.4 Puppeteer 80
4.2.2 Applikationsserverbasierte Integration 83
4.2.2.1 DataX 85
4.2.2.2 Mobile Data Synchronization System 88
4.2.2.3 Portal To Go 90
4.2.3 Entwurfszeit–Transformation 92
4.2.3.1 User Interface Markup Language 94

4.2.3.2 Universal Lightweight Client 98
4.2.3.3 eXtensible User interface Language 100
4.2.3.4 XML Formularbeschreibungssprachen 101
4.2.4 Bewertung existierender Ansätze 103
4.3 Einbindung von Dienstnehmern — Integration durch Abstraktion 113
4.3.1 Integration durch Abstraktion 116
4.3.1.1 Introspektion als Schlüsselmechanismus 117
4.3.2 Entfernte Steuerung von Anwendungen 125
4.3.3 Realisierung der Abstraktionsschicht 132
4.3.4 Universelle Geräteintegration mittels generischen Adaptern 142
4.3.5 Gerätespezifische Adaption durch Geräteprofile 150
4.3.5.1 Merkmale und Profile 151
4.3.5.2 Geräteprofile als Abbild konkreter Eigenschaften 154
4.3.5.3 Geräteklassen als Basis optimierter Diensterbringung 156
4.3.5.4 Strategien zur optimalen Diensterbringung . . . 167
4.3.6 Konkrete Realisierungstechniken 169
4.3.6.1 Resource Description Framework 169
4.3.6.2 Composite Capability/Preference Profile 171
4.3.6.3 UAProf 176
4.3.7 Geräteprofil– und Geräteklassenkonzepte im Vergleich . . 181
4.3.8 Sitzungssteuerung auf Anwendungsebene 184
4.4 Einbindung von Diensterbringern – Integration durch Migration . 194
4.4.1 Middleware im Überblick 195
4.4.2 Nutzung von Middleware zur Geräteintegration 196
4.4.2.1 Mobile CORBA 196
4.4.2.2 OnTheMove 201
4.4.2.3 JINI 203
4.4.3 Integration durch Migration 207
4.4.4 Unterstützung mobiler Diensterbringer durch JINI 213
4.4.5 Ein generischer Ansatz zur Migrations–basierten Integration 215
4.4.5.1 Integration mittels Surrogaten 216
4.4.5.2 Nutzung opaker Treiber als Integrationsmechanismus 219
4.4.5.3 Realisierung opaker Treiber 221
4.4.5.4 Vorteile des Konzepts 223
4.5 Diskussion: Surrogate vs. opake Treiber 224
4.6 Zusammenfassung 227

II Eine Integrationsumgebung auf Anwendungsebene 233

5 Eine Integrationsumgebung für mobile Geräte 235

5.1 Entwurfskriterien 235
5.2 Die funktionalen Schwerpunkte der Integrationsumgebung 238
5.3 Klassifikation der Integrationsumgebung 248
5.3.1 Klassifikation als Infrastruktur oder Umgebung 248
5.3.2 Klassifikation als Middleware 248
5.3.3 Klassifikation als Framework 251
5.3.4 Klassifikation als Meta–Middleware 252
5.3.5 Klassifikation als Architektur in ‚0+n' Schichten 261
5.4 Zusammenfassung 265

6 Die „Java Border Service Architecture" (JBSA) 267

6.1 Die Auszeichnungssprache JSML 269
6.2 Realisierung der Abstraktionsschicht 272
6.2.1 Initialisierung des Application Shadow 274
6.2.2 Introspektion mittels Scanner 276
6.2.3 Behandlung von Layout Aspekten 278
6.3 Transformation von JSML Dokumenten 280
6.4 Der HTTP–ECA 281
6.4.1 Geräte– und Anwendungssitzung im HTTP–ECA 282
6.4.2 Ermittlung von Gerätetypen im HTTP–ECA 282
6.4.3 Kaskaden zur Behandlung spezifischer Merkmale 284
6.5 Sitzungssteuerung 285
6.5.1 Phasensteuerung durch das JBSA Gateway 285
6.5.2 Abarbeitung einer Anfrage 286
6.5.3 Signalisierung in der Sitzungssteuerung 288
6.6 Umkehrung der Rollenverteilung mittels Push 290
6.7 Zusammenfassung 293

7 Die „Jini Small Device Adapters" (JiSDA) 295

7.1 Dienstregistrierung und Dienstlokation in JINI 295
7.1.1 Erschließung der Infrastruktur – Discovery 296
7.1.2 Registrierung eines Dienstes – Join 298
7.1.3 Abfrage von Dienstangeboten – Lookup 299
7.1.4 Erneuerung von Angeboten — Leasing 300

7.2 JiSDA: Ein Treiberkonzept für ressourcenschwache JINI Dienste 301
7.2.1 Ein modifiziertes Lokationsprotokoll für JINI 302
7.2.2 Aufbau des opaken Treibers 304
7.2.3 Entwicklungsprozess 305
7.3 JiSDA Beispiel–Szenario: Nutzung einer Miniatur–Kamera 307
7.3.1 Aufgabe und Funktion des opaken Treibers 308
7.3.2 Dienstattribute im Kamera–Szenario 309
7.3.3 Interne Registrierung des Kamera Treibers 311
7.3.4 Nutzung des Kamera Treibers durch den Klienten 311
7.3.5 Integration in die Systemsoftware der Kamera 312
7.4 Zusammenfassung 313

8 Zusammenfassung und Ausblick 315
8.1 Zusammenfassung 315
8.2 Ausblick 318

Literaturverzeichnis 321

„... As researchers and users, there are a number of enchanting reasons why nomadicity should interest you.“

Leonard Kleinrock: *Nomadic Computing – An Opportunity. [Kle95]*

Kapitel 1

Einleitung

Mit der Verbreitung von Mobiltelefonen und Persönlichen Digitalen Assistenten (PDAs) ist die moderne Informationstechnologie in Bereiche vorgedrungen, die der EDV zuvor weitestgehend verwehrt geblieben waren: Die persönlichen Sphären ihrer Anwender. Der Computer stellt nicht länger ein technisches Hilfsmittel dar, das nur auf den Schreibtischen der Büros verfügbar ist: Mit PDA und Mobiltelefon ist das Werkzeug zu unserem ständigen Begleiter geworden.

Die ständige Verfügbarkeit führt zu einer ungezwungenen Beschäftigung mit den Möglichkeiten. Neugierde als wichtiger Faktor sorgt dafür, dass Machbares ausgetestet wird und sich immer neue Anwendungsfelder quasi aus der Verwendung heraus ergeben. Wie im Fall des Short Message System (SMS) können überraschend Märkte entstehen, welche durch die Dynamik ihres Wachstums der technischen Entwicklung positive Impulse zu geben vermögen. Durch die sich folglich ergebende massenhafte Verwendung treten Anforderungen an die Technik deutlich zu Tage und müssen erfüllt werden, um dem entstandenen Markt gerecht zu werden.

Im mobilen Einsatz können Computer genauso wie Daten und Benutzer nach Belieben ihren Standort wechseln und dabei doch ständig mit anderen Benutzern und ihren Daten in Kontakt bleiben. Die Ortsveränderung von Ressourcen wird im mobilen Anwendungsfeld zum Merkmal des Systems, und der gewollte Verbindungsabbau tritt als wichtiger Realisierungsaspekt neben bisherige Probleme wie den Verbindungsabbruch im Fehlerfall. Im mobilen System wird der statische Aufbau des Rechnerverbundes zunehmend durch dessen ständige dynamische Neugestaltung abgelöst. Bewährte Technologien sind hierdurch nurmehr bedingt einsetzbar.

Während die zunehmende Mobilität der Menschen in unserer modernen Gesellschaft als soziale Komponente die Motivation für den Bedarf nach mobilen Systemen schafft, ist die Erfüllung des entstandenen Bedarfs nur auf der Basis der technischen Weiterentwicklung möglich. Elementare Grundvoraussetzung ist die zunehmende Miniaturisierung verfügbarer Systeme: Nur das problemlos mitzuführende Gerät wird zum ständigen Begleiter werden. Aber auch andere technische Innovationen haben als Wegbereiter dazu beigetragen, eine weitgehende Akzeptanz für mobile Computersysteme zu schaffen.

Allen voran kann die steigende Leistungsfähigkeit von Energiespeichern genannt werden. Nur die Verwendung von Batterien mit hoher Lebensdauer ermöglicht eine wirklich autonome Nutzung mobiler Geräte, die deren Verwendung fernab von Steckdosen und Stromkabeln sinnvoll werden lässt. Die notwendige Aufladung der Batterien muss immer weniger als hinderlich wahrgenommen werden, da sie einfach am Ende eines Arbeitstages geschehen kann. Nebenbei haben die Energiespeicher auch immer mehr an Gewicht verloren, so dass sie neben den eigentlichen Komponenten des mobilen Gerätes nur noch marginal auftragen.

Auch Entwicklungen im Bereich der Bildschirmtechnik haben zur Verbreitung mobiler Systeme beigetragen. Durch die Kombination mit Drucksensoren konnte der Bildschirm des mobilen Computers zu einer interaktiven Benutzungsschnittstelle erweitert werden, die der intuitiven Handhabung eines Informationsmediums durch den Menschen sehr nahe kommt: Durch direktes Tippen auf Auswahlknöpfe und sogar durch Eingabe handschriftlicher Befehle kann der technische Charakter des zugrunde liegenden Computers verborgen werden und die Verwendung durch technisch nicht weiter vorgebildete Benutzer einfach werden.

Die immer noch stark steigende Rechenleistung mobiler Geräte lässt darüber hinaus eine weitere Interaktionsschnittstelle für diese Geräte zunehmend realistisch erscheinen, die zuvor selbst für stationäre Computer als utopisch galt: Die gesprochene Sprache. Bereits heute ist es in begrenztem Maße möglich, den PDA durch Kommandos zu steuern, die über ein Mikrofon in das Gerät gesprochen werden. Es ist sicherlich nicht zuletzt die Einführung solch intuitiver Schnittstellen, die der mobilen EDV zu immer weiterer Verbreitung verhilft.

Schließlich ist die zunehmende Kommunikationsgeschwindigkeit sowie die flächendeckende Verfügbarkeit heutiger bzw. zukünftiger Mobilkommunikationssysteme als Faktor zu nennen. Satellitensysteme wie IRIDIUM ermöglichen die mobile Datenkommunikation von jedem Punkt der Erde aus. Mit GSM, GPRS und zunehmend auch UMTS existieren Technologien, welche den kostengünstigen Zugriff auf entfernte Informationssysteme aus dem mobilen Umfeld heraus nahezu jederzeit möglich machen; im drahtlosen LAN erreichen Übertragungskanäle eine Geschwindigkeit, die auch zur Vermittlung multimedialer Daten geeignet ist, und schließlich erhält mit Bluetooth eine Funknetztechnologie für den Nahbereich weite Verbreitung, die auch den direkten Austausch von Mobilgeräten untereinander effizient werden lässt. In der Breite ihres Spektrums ermöglichen mobile Kommunikationsnetzwerke demnach zunehmend die Illusion einer nahezu permanenten Verbindung des mobilen Geräts zum entfernten Kommunikationspartner.

Drahtlose Kommunikationsnetze können mit den durch sie verbundenen Mobilgeräten als spezialisierte Form eines verteilten Systems verstanden werden, als dessen Anpassung an einen allgemeinen Trend hin zu zunehmender Mobilität des menschlichen Benutzers. Während drahtlose Netze den verteilten Systemen grundsätzlich in ihren Merkmalen, den auftretenden Problemen sowie deren Lösungen gleichen, stellen sie den Anwendungsentwickler aufgrund der Optimierung hinsichtlich des speziellen Einsatzkontexts vor eine anspruchsvolle Aufgabe: Nur bei ausdrücklicher Berücksichtigung der hinzugekommenen Merkmale wie geplantem Verbindungsabbau, Relokation, Ressourcenknappheit, hohen Kommunikationskosten usw. kann es gelingen, verlässliche und somit robuste Anwendungssysteme zu schaffen. Mobilität muss demnach als wichtiger Aspekt

in den Entwurf verteilter Softwarearchitekturen einbezogen werden. Erprobte Konzepte müssen neu bewertet und den geänderten Voraussetzungen angepasst werden. Im weiteren Verlauf dieser Arbeit wird auf mögliche Ansätze hierzu umfassend eingegangen.

Wie kann der Begriff der Mobilität nun genauer gefasst werden? Wie gesehen kann zunächst einmal zwischen technischer und sozialer Mobilität unterschieden werden. Innerhalb der technischen Domäne ergibt sich dabei eine Reihe möglicher Definitionen bzw. Ausprägungen des Mobilitätsbegriffs. Insbesondere sind folgende Begriffe gebräuchlich:

- **Mobile Kommunikation**: Umfasst alle Aspekte der drahtlosen Übermittlung von Informationen zwischen räumlich getrennten Punkten.
- **Mobile Computing**: Behandelt Aspekte konkreter Umsetzungen von Computern und Kommunikationsgeräten, die ohne Kabelverbindungen verwendet werden können, bis hin zu Anwendungen, welche die Verwendung solcher Geräte einbeziehen.
- **Mobile Computation**: Umfasst alle Aspekte der Durchführung von Berechnungen in verteilten Systemen, bei denen die Berechnungsknoten bzw. Berechnungsdaten nicht räumlich fixiert sind, sondern zwischen den Knoten ausgetauscht werden können, also z.B. migrierender Code, mobile Agenten usw.

Das folgende Kapitel 2 befasst sich eingehend mit den verschiedenen Formen von technischer Mobilität und geht näher auf jeden der genannten Bereiche ein.

1.1 Defizite mobiler Geräte

Wesentlich ist an dieser Stelle zunächst folgendes: Technische Mobilität weist ein breites Spektrum auf. Sie umfasst die Bewegung des Anwenders genauso wie die Bewegung des Gerätes. Sie umfasst die Bewegung von Programmen wie auch von Daten, die mit diesen Programmen verarbeitet werden. Technische Mobilität spielt sich auf der Ebene der Geräte ab, auf der Ebene der Kommunikationsmedien und auf der Ebene der Anwendungsentwicklung. Und schließlich bewegt sich der Bereich der technischen Mobilität in einem Spannungsfeld zwischen Miniaturisierung zum Zwecke eines flexiblen Ortswechsels auf der einen Seite und auf der anderen Seite der Schaffung eines möglichst hohen Nutzungsgrades, der jedoch dem Miniaturisierungsbestreben entgegensteht.

Doch wo liegen die Defizite und Probleme heutiger Formen von technischer Mobilität? Auch wenn die Leistung mobiler Geräte der neuesten Generation derer von Rechenanlagen vergangener Jahrzehnte überlegen zu sein scheint, so bleibt doch festzuhalten, dass sie wahrscheinlich niemals mit der Leistung jeweils *moderner* Arbeitsplatz– oder gar Großrechner vergleichbar sein wird. Dieses bezieht sich auf die Rechengeschwindigkeit genauso wie auf die Speicher– bzw. Kommunikationsgeschwindigkeit. Ein erster Problembereich ergibt sich also daraus, dass Ansprüche und Anforderungen des Benutzers in der IT zumeist im Zusammenhang mit der Tätigkeit am Arbeitsplatzrechner entstehen, diese jedoch nicht in gleicher Weise durch mobile Geräte befriedigt werden können.

Hierzu ein Beispiel: Arbeitet ein Angestellter an seinem Arbeitsplatz mit der Textverarbeitung, so wird er die in ihrer Benutzung erlernten Fähigkeiten auch auf seinem PDA anwenden können wollen, vielleicht sogar die zuvor am Arbeitsplatz erstellten Dokumente auf das mobile Gerät übertragen und diese dort weiter bearbeiten wollen. Mag der PDA vielleicht leistungsfähiger sein als ein PC von vor zehn Jahren, so ist er höchstwahrscheinlich dennoch nicht in der Lage, die aktuelle Version der Textverarbeitung zu starten. Ohnehin hat sich die minimale Dokumentengröße in den vergangenen Jahren vervielfacht, das zu bearbeitende Dokument wird also mit großer Wahrscheinlichkeit nicht auf dem PDA Platz finden. Soll das Dokument also weiter bearbeitet werden, muss es in ein passendes Format konvertiert und dann auf dem PDA in ein speziell für das entsprechende Gerät entwickeltes Textverarbeitungsprogramm eingeladen werden. Konvertierung, Übertragung und der geringe Funktionsumfang der PDA Software schaffen Brüche, die einer effizienten Nutzung im Wege stehen.

An diesem Beispiel lassen sich jedoch auch weitere Probleme verdeutlichen. So besteht ein Bestreben in der Weiterentwicklung mobiler Geräte darin, diese bezüglich Größe und Gewicht zu reduzieren und so deren ständige Mitnahme zu vereinfachen. Während einer fortlaufenden Miniaturisierung *technisch* kaum Grenzen gesetzt sind, gibt es eine Konstante, die Beachtung finden muss: die Ergonomie der Benutzungsschnittstelle. Indem mobile Geräte immer kleiner werden, wird ihre Bedienung immer mühseliger. Gesetzt den Fall, es gelänge, einem PDA die Leistungsfähigkeit eines Arbeitsplatzrechners zu geben und diesen in seiner Größe auf wenige Kubikzentimeter zu reduzieren, so bliebe es dennoch nur sehr bedingt praktikabel, auf der miniaturisierten Tastatur (oder gar in Kurzschrift auf dem Bildschirm) längere Texte zu erstellen oder diese über das beschränkte Sichtfeld des Bildschirms zu betrachten. Wird ein Gerät andererseits mit einer ausreichend großen Tastatur bzw. einem genügend großen Bildschirm ausgestattet, so leidet die Portabilität, das Gerät ist also nur noch bedingt (bzw. nur bezüglich einer sehr weit gefassten Definition) als *mobil* zu bezeichnen. Gerade bei modernen Laptops stellt sich genau diese Frage. In ihrer Leistungsfähigkeit sind sie Arbeitsplatzrechnern tatsächlich ebenbürtig — doch sind sie aufgrund ihrer Abmaße wirklich noch als mobil zu bezeichnen? In ihrer Betrachtung zur Definition von Mobilität geht diese Arbeit unter anderem dieser Frage nach.

Noch ein weiteres Problem lässt sich aus obigem Beispiel ersehen. So stellt der Austausch von Informationen von mobilen Geräten untereinander sowie zwischen mobilen Geräten und stationären Geräten einen Punkt von hoher Relevanz für die Nutzbarkeit und Praxistauglichkeit dar. Während der Austausch von Informationen mit stationären Kommunikationspartnern grundsätzlich immer ein Problem mobiler Geräte war, ist diesbezüglich eine zunehmende Verschärfung zu beobachten: Um die Praktikabilität zu erhalten, wird oftmals so verfahren, dass die mobilen Geräte entsprechend ihres Anwendungskontextes optimiert werden. Soll ein Gerät zur Eingabe von Text genutzt werden, so wird es mit einer größeren Tastatur ausgestattet, ist lediglich die Sichtung vorhandener Texte geplant, so entfällt eine Tastatur ggf. ganz, notwendige Eingaben erfolgen über wenige Knöpfe oder die Schrifteingabe auf dem Bildschirm. Nach diesem Ansatz erfolgt idealerweise eine Partitionierung des Anwendungsspektrums des Benutzers, wobei jedem Anwendungsgebiet ein Gerät zuzuordnen ist, das sodann in seinen Fähigkeiten optimal ausgenutzt wird: Das Mobiltelefon zur Sprachkommunikati-

on, das digitale Adressbuch zur Verwaltung von Telefonnummern, der Pager zum Empfang von Textbotschaften usw. Hierdurch wird die weitgehende Minimierung eines jeden Geräts und seine Anpassung an den Anwendungsfall möglich, doch erkauft man sich diese Optimierung durch eine Abkehr von der universellen Einsetzbarkeit des Geräts. Nur indem man eine große Zahl verschiedener Geräte bei sich trägt, kann man sicherstellen, dass alle auftretenden Bedarfe abgedeckt werden können. Außerdem muss gewährleistet werden, dass die Daten, die auf einem Gerät gespeichert sind, von einem anderen Gerät nutzbar sind. Dieses ist jedoch häufig nicht der Fall. Adressen, die auf dem PDA gespeichert sind, können nicht direkt für den Wahlvorgang auf dem Mobiltelefon genutzt werden. Kurznachrichten, die vom Pager aufgefangen werden, können nicht ohne weiteres im PDA bearbeitet werden usw. Und die geschilderte Problematik verstärkt sich nochmals, wenn der Arbeitsplatz in das Szenario einbezogen wird: Dokumente, die mit dem PC geschrieben wurden, können nicht direkt auf den PDA übertragen werden, Telefonnummern, die auf dem PDA erfasst wurden, nicht direkt zum Wählen mit dem Bürotelefon verwendet werden usw. Mit solch einer zunehmenden Spezialisierung auf der Geräteseite kommt es zu immer größeren Problemen in der übergreifenden Datennutzung sowie dem Datenaustausch.

Als wesentlicher Problembereich wird die *Integration* mobiler Geräte erkennbar. Bestehende Konzepte zur nahtlosen Kopplung von Geräten in einem gemeinsamen Anwendungskontext müssen hinsichtlich ihrer Unterstützung mobiler Szenarien überprüft und angepasst werden. Der Zugriff auf entfernte Anwendungen in einem heterogenen Umfeld muss auch unter Einsatz mobiler Geräte ermöglicht werden. Den damit verbundenen Fragestellungen gilt in dieser Arbeit das Hauptaugenmerk.

1.2 Integrationsebenen

Was bedeutet Integration im Kontext technischer Mobilität? Die oben genannten Beispiele geben für eine erste Charakterisierung einen Anhaltspunkt: Das vornehmliche Ziel der Integration liegt zumeist im Austausch von Daten zwischen Mobilgerät und Zentralrechner, zwischen Arbeitsplatzrechner und Mobilgerät oder auch zwischen verschiedenen Mobilgeräten. Daten sollen jeweils auf dem Gerät erfasst werden können, das hierzu am besten geeignet ist und dennoch auf allen anderen verwendeten Geräten genutzt werden können, wenn es der Anwendungsfall verlangt. Im Unternehmen sollen Daten im zentralen Rechensystem erfasst und gespeichert werden können und sodann allen Mitarbeitern für ihre Arbeit vor Ort zur Verfügung stehen. Das mobile Gerät soll in beliebige betriebliche Abläufe einbezogen werden können. Schließlich sollen die mit dem mobilen Gerät erfassten Daten wieder in die betrieblichen Abläufe zurückfließen können, um dann im weiteren Verlauf zur weiteren Verarbeitung am Arbeitsplatz nutzbar zu sein. Gemeinsam sollen alle beteiligten Geräte und alle verwendeten Anwendungen einen kontinuierlichen und homogenen Informationsraum bilden.

Es ist offensichtlich, dass der Integrationsbegriff so viele Aspekte betrifft, wie sich Anwendungen für mobile Geräte finden lassen. Ein Teil dieser Arbeit ist der Untersuchung existierender Ansätze für die Integration mobiler Geräte gewidmet. Dabei lässt sich zeigen, dass die in der Literatur beschriebenen Integrationsverfahren sich anhand des OSI Referenzmodells (vgl. [Ker92]) einordnen lassen.

Es wird sich zeigen, dass jede seiner Schichten sich zur Behandlung bestimmter Integrationsaspekte nutzen lässt und sich zu jeder der definierten Ebenen Integrationsansätze in der Literatur finden lassen.

Wie noch genauer ausgeführt werden wird, bietet insbesondere die Anwendungsschicht großes Potential bei der Integration mobiler Geräte. So kann auf dieser Ebene entschieden werden, welche Daten Anwendungsbezug besitzen und daher zu übertragen sind. Auch kann auf dieser Ebene ein Integrationsansatz gewählt werden, bei dem die eigentliche Verarbeitung von Anwendungsdaten nicht mehr auf dem Gerät selbst erfolgt, sondern lediglich die Benutzungsschnittstelle der Anwendung dargestellt wird, die eigentliche Anwendung mitsamt ihrer Daten jedoch auf der Seite des zentralen Rechensystems verbleibt. Solche Ansätze sollen in dieser Arbeit genauer betrachtet werden.

Ein wichtiges Merkmal der meisten mobilen Geräte, das auch im Zusammenhang mit deren Integration eine Rolle spielt, ist die in diesem Bereich vorherrschende „Geschlossenheit“ der Systeme. Geschlossenheit soll in diesem Zusammenhang besagen, dass eine nachträgliche Modifikation durch den Anwender oder einen von ihm beauftragten Programmierer nicht ohne weiteres möglich ist. Geschlossene Geräte verfügen entweder nicht über die (entsprechend dokumentierten) Schnittstellen zu deren Programmierung oder sie sind technisch nicht dazu ausgelegt, nachträglich eingespielte Programme lokal zu speichern und abzuarbeiten. Wenngleich solche Geräte heutzutage oftmals über die Fähigkeit verfügen, auf entfernt abgespeicherte Daten zuzugreifen und diese darzustellen (wie z.B. WAP Telefone), so sind die Möglichkeiten zur Integration auf Anwendungsebene hierdurch dennoch stark eingeschränkt.

Sollen die Vorteile des Ansetzens auf der Anwendungsebene genutzt werden, so muss also ein Weg gefunden werden, wie dieses geschehen kann, ohne dass ein Eingreifen in das Mobilgerät selbst notwendig ist. Ein Weg, der hierzu beschritten werden kann, besteht darin, die verwendete Anwendung so zu modifizieren, dass die von ihr erstellten Ausgaben ein Format besitzen, das von dem mobilen Endgerät verstanden und dargestellt werden kann. So ist es zunehmend populär, XML zur Definition von Datenformaten zu verwenden, mit denen dann die zu übertragenden Informationen formuliert werden können. In das richtige Format überführt, können sie dann beispielsweise im WAP Browser des Mobiltelefons oder im Web Browser des PDAs angezeigt werden. Nachteilig ist hierbei jedoch, dass eine explizite Modifikation der Anwendung notwendig ist. Eine Berücksichtigung dieses Ansatzes ist somit zwar bei der Neuentwicklung von Anwendungen ernsthaft zu erwägen. Für bestehende Anwendungen stellt dies jedoch ein ähnliches Problem dar wie die Anwendungsentwicklung für das Mobilgerät. Auch hier ist der Aspekt der Geschlossenheit zu erkennen. Die Behandlung von Geschlossenheit bzw. der erreichbare Grad an Transparenz sind in der Literatur unter dem Stichwort der „Awareness“ bekannt. Der Erläuterung dieser Begrifflichkeit wird in dieser Arbeit gesonderter Raum geschenkt.

1.3 Fragestellung und Gang der Untersuchung

Insgesamt ergibt sich die zentrale Fragestellung dieser Arbeit folgendermaßen:

Ist es möglich und dabei praktikabel, eine Integration auf Anwendungsebene zu schaffen, die sowohl für die Anwendung als auch für das zu integrierende Gerät transparent ist? Wie sieht ein entsprechendes Konzept aus und welche Möglichkeiten zur praktischen Umsetzung lassen sich vor dessen Hintergrund herleiten?

Dieser Frage widmet sich diese Arbeit in zwei Teilen. In ihrem ersten Teil wird zunächst auf Grundlagen eingegangen, die im Rahmen der Integration mobiler Geräte berücksichtigt werden müssen. Es werden verschiedene Definitionen für Mobilität einander gegenübergestellt, um den Hintergrund der Untersuchung einzugrenzen. Des Weiteren wird ein Klassifikationsschema für Integrationsansätze eingeführt und dann anhand von konkreten Beispielen dargestellt, wie sich die Integration mobiler Geräte auf den verschiedenen Ebenen dieses Schemas gestaltet. Im weiteren Verlauf des ersten Teils wird besonders detailliert auf existierende Ansätze eingegangen, die der Anwendungsebene zuzurechnen sind. Ausgehend von Erkenntnissen über deren Aufbau werden dann die notwendigen Teilkonzepte hergeleitet, die ein Ansatz mit den laut Fragestellung geforderten Merkmalen aufweisen muss. Die Herleitung ist dabei ihrerseits in zwei Teile aufgespalten: Zum einen die Integration von Dienstnehmern, mobilen Geräten also, die vornehmlich zum lesenden Zugriff auf das zentrale Anwendungssystem Verwendung finden. Zum anderen die Integration von Diensterbringern, d.h. mobilen Geräten, die überwiegend zur Einspeisung von Informationen in das Anwendungssystem dienen.

Der zweite Teil der Untersuchung dient der Diskussion konkreter Umsetzungsmöglichkeiten. Hier wird die eigentliche Integrationsumgebung vorgestellt, die auf der Basis der zuvor beschriebenen Bausteine entstanden ist. Es werden zunächst allgemein die funktionalen Schwerpunkte dieser Umgebung beschrieben, und es wird das Zusammenspiel der Komponenten untereinander erläutert. Im Anschluss wird die entstandene Integrationsumgebung entsprechend der im Bereich verteilter Systeme üblichen Klassifikationen eingeordnet und erörtert, ob es sich hierbei beispielsweise um *n–Tier Software*, um Middleware oder gar um Meta–Middleware handelt. Auch diese Begriffe werden im Verlauf der Arbeit herausgearbeitet. Schließlich wird auf konkrete Implementationsaspekte im Detail eingegangen und erläutert, auf welche Weise systemtechnische Probleme im Zusammenhang mit der Integration mobiler Geräte als Diensterbringer sowie als Dienstnehmer gelöst werden können. Abschließend erfolgt eine Bewertung der erarbeiteten Integrationsumgebung.

Teil I

Integration mobiler Geräte in verteilte Umgebungen

Kapitel 2

Grundlagen mobiler Systeme

In diesem Kapitel wird die im Zuge der Integration mobiler Systeme in bestehende Informationssysteme auftretende Problematik vorgestellt, die sich mehr und mehr als eines der komplexesten Probleme im Bereich der mobilen Systeme herausgestellt hat und die mittlerweile in einer Vielzahl von Projekten und Publikationen Eingang gefunden hat (vgl. [JWW+95]). Wenngleich mobile Systeme von ihrem technischen Aufbau nahezu durchgehend dem klassischen Von–Neumann–Ansatz folgen, so ergeben sich vor dem Hintergrund ihrer Ausrichtung auf überwiegend spezielle Anwendungsgebiete eine Reihe von Problemen, die für ein normales Computersystem nicht zu erwarten sind. Entsprechend wäre es verfehlt, mobile Systeme lediglich als miniaturisierte Standardsysteme zu betrachten und zu deren Einbindung auf Algorithmen zurückzugreifen, wie sie im Bereich der verteilten Systeme entwickelt wurden, wie beispielsweise Imielinski in [IK96] ausführt:

> *Mobile Computing is not a „scaled–down" version of the established and well–studied field of distributed computing. The nature of wireless communication media and the mobility of computers combine to create fundamentally new problems in networking, operating systems , and information systems. Furthermore, many of the applications envisioned for mobile computing place novel demands on software systems.* [IK96], S. xxi.

In diesem Sinne ist es also notwendig, herauszuarbeiten, was Mobilität ist, worin die Eigenheiten und Probleme einer mobilisierten Datenverarbeitung liegen und auf welche Weise den Problemen begegnet werden kann. Es wird demgemäß im Folgenden aufgearbeitet, was unter Mobilität bzw. einem mobilen System als solchem zu verstehen ist, wobei der Mobilitätsbegriff zunächst allgemein beleuchtet wird, quasi als Grundlage für die Abgrenzung des in dieser Arbeit betrachteten Bereiches der Mobilität von anderen Bereichen, die außer Acht gelassen werden sollen.

Im Anschluss daran wird besprochen, nach welchen Gesichtspunkten sich Taxonomien für mobile Systeme erarbeiten lassen und durch welche inhärenten Eigenschaften bzw. Einschränkungen die so klassifizierten mobilen Systeme gekennzeichnet sind. Hauptziel dieses Abschnittes ist es entsprechend, eine einheitliche Begrifflichkeit für alle weiteren Ausführungen dieser Arbeit zu vermitteln.

Um den Gegenstand dieser Arbeit, die Integration mobiler Geräte also, besser verstehen zu können, muss, in ähnlicher Weise wie auf Aspekte der Mobilität generell, auf die zu unterstützenden Geräte eingegangen werden. Der Bereich der mobilen Geräte zeichnet sich, bedingt durch die für Mobilgeräte notwendige Spezialisierung auf das jeweilige Anwendungsgebiet, durch eine überdurchschnittlich ausgeprägte technologische Diversifikation aus. Eine Analyse der domänenspezifischen Integrationsproblematik setzt demgemäß eine genaue Kenntnis der Geräteklassen und deren Charakteristik voraus. Abschnitt 2.4 soll diese vermitteln, indem hier anhand der zuvor eingeführten Taxonomie ein Überblick mobiler Geräte gegeben wird, wie sie derzeit auf dem Markt verfügbar sind und auf die klassenspezifischen Eigenschaften eingegangen wird, die im Rahmen ihrer Nutzung für den Zugriff auf Informationssysteme zum Tragen kommen.

2.1 Soziale vs. physische Mobilität

Es ist an dieser Stelle zunächst notwendig, den Begriff der Mobilität näher zu beleuchten und anhand allgemeiner Definitionen zu fundieren.

Uneingeschränkte Mobilität bildet von jeher eines der elementaren Grundbedürfnisse des Menschen. Der Wunsch nach freier Ortswahl ist gewissermaßen untrennbar mit unserer Geschichte verbunden, wie sich anhand der nomadischen Wurzeln unserer Vorfahren belegen lässt. Weite Teile der Erde wurden nur durch über Jahrtausende anhaltende Wanderungsbewegungen bevölkert und auch in der Neuzeit gilt das Recht des Einzelnen auf Mobilität als wichtiger Faktor für den Grad an Lebensqualität.

Primär wird der allgemeine Mobilitätsbegriff in *regionale*, *horizontale* bzw. *physische* Mobilität als der Änderung der räumlichen Position auf der einen Seite sowie *soziale* bzw. vertikale Mobilität als sozialem Auf– oder Abstieg auf der anderen Seite unterteilt (siehe [Bro86a], S. 149).

Im Zusammenhang dieser Arbeit soll nur die räumliche, physische Mobilität von Computersystemen betrachtet werden, während die etwaigen soziologischen bzw. sozioinformatischen Aspekte der Mobilität unbeachtet bleiben. Für eine eingehende Betrachtung dieser Themenkomplexe sei auf entsprechende Literatur verwiesen wie z.B. [Lam01], eine Sammlung von Aufsätzen, in der u.A. Peter Sloterdijk auf die sozialen bzw. kulturellen Implikationen der Mobilität eingeht.

2.2 Mobilität in der Datenverarbeitung

Die mobile Datenverarbeitung lässt sich grob als Spezialisierung der Datenverarbeitung in verteilten Systemen bezeichnen. Berechnungen werden dabei nicht an zentraler Stelle durchgeführt, sondern können an wechselnden Orten innerhalb

eines Netzwerkes oder, allgemeiner, an frei wählbaren Positionen im Raum stattfinden. Im Gegensatz zur herkömmlichen verteilten Verarbeitung ändern dabei jedoch nicht nur Berechnungen (d.h. Programme, Daten, Kontrollflüsse usw.) ihre räumliche Position. Vielmehr können auch die Berechnungsknoten selbst ihren Standort wechseln. Der statische Aufbau des Systems weicht zunehmend einer dynamischen, sich in hoher Frequenz ständig wechselnden Konstellation von Knoten und Verbindungen.

2.2.1 Portabilität vs. Mobilität

In seiner einfachsten technischen Form lässt sich Mobilität durch die Relokation der Rechenmaschine erzielen. In [Per98] wird hierbei eine Unterscheidung in *portable computing* auf der einen Seite und *mobile computing* auf der anderen Seite beschrieben. Sowohl Daten als auch Programme lassen sich durch die Entkopplung des Systems vom Netzwerk, dessen Transport an einen neuen Ort und die dortige Kopplung an das Netzwerk bedingt mobilisieren. Perkins bezeichnet dieses als *portable computing*. Ein wesentliches Merkmal des Portable Computing ist hiernach das Ausbleiben von Berechnungen *während* des eigentlichen Ortswechsels und die Notwendigkeit von Anpassungen *nach* dem Ortswechsel. Dieser passiven Form der Mobilität steht eine Form der Mobilität gegenüber, bei der die Funktionstüchtigkeit des Computersystems von der Relokation nicht beeinträchtigt wird. Insbesondere können Berechnungen während des gesamten Vorgangs der Abkopplung, Bewegung und Ankopplung des Gerätes weiterlaufen. In [Per98] wird dieses als *mobile computing* bezeichnet.

Die Definition von Perkins bezieht sich primär auf die Netzwerkkommunikation des betrachteten Rechnersystems. Während für den portablen Computer von einer Rekonfiguration nach dessen Einbindung in eine Netzwerkinfrastruktur an neuer Stelle auszugehen ist, zeichnet sich das mobile System dadurch aus, dass es bei Anschluss an ein *beliebiges* Netzwerk die infrastrukturellen Gegebenheiten selbst zu erschließen in der Lage ist und sich gegebenenfalls auf die geänderte Situation einstellen kann. Als Fundament für Mobilität muss gemäß Perkins somit zunächst sichergestellt sein, dass ein Computersystem ohne Benutzerintervention in der Lage ist, seine eindeutige Adresse innerhalb eines Netzwerkes zu beziehen, die Diensterbringer innerhalb des Netzwerkes zu identifizieren sowie seine geänderte Position an das Netzwerk mitzuteilen und so sicherzustellen, dass eine eventuell zuvor unterbrochene Interaktion mit einer Gegenstelle wieder aufgenommen werden kann. Zur Realisierung dieser elementaren Form von Mobilität existiert mittlerweile eine Reihe zum Teil komplementärer Protokolle, auf die im Rahmen dieses Kapitels noch näher eingegangen wird.

2.2.2 Hardware– vs. Softwaremobilität

Es ist offensichtlich, dass sich der zuvor genannte Begriff der Mobilität zunächst einmal nur auf die Mobilität des Computers *relativ* zum Netzwerk beschränkt, welchem er zugeordnet ist. Das Computersystem selbst und die hierauf laufenden Anwendungen werden dabei im Sinne einer *Black Box* Sicht nicht näher betrachtet. In einer Verfeinerung dieses noch recht grobgranularen Klassifikationsansatzes schlägt Cardelli das folgende Schema vor, bei welchem nicht das Gerät,

sondern vielmehr die Anwendungsprogramme sowie die von ihnen verarbeiteten Daten in den Vordergrund gestellt werden (vgl. [Car97] bzw. [RCDD98]). Anwendungsmobilität ist hiernach einer von drei Kategorien zuzuordnen:

Control Mobility : Eine Verlagerung des Kontrollflusses einer Berechnung zwischen mehreren Berechnungsknoten innerhalb eines Netzwerkes, z.B. im Rahmen eines entfernten Prozeduraufrufs.

Link Mobility : Variante des *Remote Procedure Calls*, bei der Referenzen zwischen Netzwerkknoten verlagert werden. Hierbei wird die Referenz auf einen Kommunikationsendpunkt im Rahmen eines RPCs an einen entfernten Knoten übergeben und von diesem sodann genutzt.

Data Mobility : Die Verlagerung von Daten über ein Netzwerk mittels Argumenten oder Rückgabewerten entfernter Prozeduraufrufe (RPCs), wobei die Struktur der Daten unmittelbar auf die Zielplattform angepasst wird (sog. *on–line portability*).

In [Car99] wird darüber hinaus noch der Begriff der Object Mobility eingeführt, welcher die Fähigkeit zur Relokation von Objekten zwischen unterschiedlichen Rechenknoten innerhalb eines Netzwerkes bezeichnet.

Den von Cardelli vorgeschlagenen Varianten von Mobilität in verteilten Systemen liegt jeweils die Annahme zugrunde, dass primär das lauffähige bzw. laufende Programm betrachtet wird, nicht jedoch der ausführende Netzwerkknoten. So können Programme ihre räumliche Position verlagern, können Daten sowie Kontrollflüsse innerhalb eines Netzwerkes bewegt werden, ohne dass sich dabei jedoch die physikalische Struktur der eingesetzten Ausführungsplattform ändern würde. Diese bleibt vielmehr über den gesamten Betrachtungszeitraum statisch. In der Literatur wird diese Form der Mobilität als *Mobile Computation* oder auch virtuelle Mobilität bezeichnet (vgl. u.a. [Car99]).

Dieser gegenüber steht die räumliche Verlagerung der eigentlichen Rechensysteme im Zuge der Programmausführung. Daten und Programme verbleiben hierbei auf dem zu bewegenden Netzwerkknoten, während dieser an eine andere Position verbracht wird. Je nach Unterstützungsgrad kann der Knoten auf die Änderung bzw. den Verlust seiner Konnektivität reagieren und auch während des Relokationsschritts mit der Programmausführung fortfahren. Diese Form der Mobilität wird allgemein als *Mobile Computing* bzw. physische Mobilität bezeichnet (vgl. u.a. [Car99]).

Dem steht schließlich ein Betrachtungswinkel auf das Gebiet der Mobilität gegenüber, der nicht die Migrationsbewegung eines laufenden Programmes oder die räumliche Verlagerung des ausführenden Rechners betrachtet, sondern die Anforderungen an die Kommunikationsnetzwerke, die sich hieraus ergeben. Dieser als *Mobile Communication* bezeichnete Aspekt der Mobiltät wird beispielsweise in den Veröffentlichungen von Cox, Goodman und Zorzi in den Mittelpunkt gerückt (vgl. [Cox99] bzw. [Zor00]). Wesentliche Aspekte in diesem Themenfeld sind die einer mobilen Kommunikation zugrunde liegenden Funksysteme, die effiziente Verwendung des begrenzten Frequenzraumes durch geeignete Aufteilung der abzudeckenden Fläche in Zellen, der Entwurf von Kodierungsverfahren, die eine größere Robustheit gegenüber Störungen ermöglichen usw (vgl. [MP92],

[PM95], [Ray97], [Goo97], [Gib99], [Sch00] usw.). Weder Mobile Computation noch Mobile Communication werden in dieser Arbeit näher betrachtet.

2.3 Merkmale mobiler Datenverarbeitung

Mobile Anwendungskontexte stellen ganz spezielle Anforderungen an die elektronische Datenverarbeitung, die sich zum Teil erheblich von denen unterscheiden können, welche sich für herkömmliche, eher statische Umgebungen identifizieren lassen. Existieren zwar gewisse Parallelen zwischen der *verteilten* und der mobilen Verarbeitung, so lassen sich doch grundsätzliche Unterschiede feststellen, die eine Neubewertung einer Vielzahl von Problemen und deren Lösungsansätzen notwendig erscheinen lassen. Die Unterschiede und speziell für mobile Systeme und deren Integration geltenden Probleme wurden in der Literatur mittlerweile ausführlich diskutiert (siehe etwa [HHC+99], [Sat95], [FZ94], [IB94]), wobei sich bestimmte Klassen von Problemen als in besonderem Maße charakteristisch für mobile Systeme herausgestellt haben. Diese sollen im Folgenden näher behandelt werden.

Imielinski und Badrinath identifizieren folgende Problemkategorien, anhand derer eine weitere Klassifikation erfolgen kann (vgl. [BI93], S. 5):

1. Mobilität, Verbindungsabbruch und Skalierbarkeit
2. Geändertes Informationsmedium und limitierte Ressourcen

Mobilität umfasst nach Ansicht der Autoren alle Probleme, die sich aus der nunmehr dynamischen gegenüber einer statischen Konstellation innerhalb des Verbundes von Computersystemen ergeben. Dieses können Aspekte sein, welche die Relokation des eigentlichen Mobilgerätes selbst betreffen, dieses können jedoch genauso auch Probleme dergestalt sein, wie sie sich durch die Relokation von Daten *innerhalb* der Menge betrachteter Computersysteme (mobil bzw. statisch) ergeben.

Verbindungsabbrüche sind all diejenigen Probleme, die sich durch den Abbruch der Kommunikationsverbindung zwischen dem mobilen Gerät sowie der Integrationsinfrastruktur ergeben. Während diese auch in herkömmlichen verteilten Systemen auftreten und diese sich grundsätzlich in ihrem Effekt in mobilen Anwendungskontexten nicht unterscheiden, so besitzen Verbindungsabbrüche im Bereich mobiler Systeme zwei Facetten, die in verteilten Systemen nicht zu erwarten sind: Zum einen treten sie mit einer wesentlich höheren Wahrscheinlichkeit auf, da das Kommunikationsmedium weniger verlässlich ist. Zum anderen treten neben die *ungeplanten* Unterbrechungen durch Verbindungsfehler et cetera die sich durch die Eigenschaften mobiler Systeme ergebenden häufigen *geplanten* Unterbrechungen, z.B. zwecks Ressourcenschonung oder vor dem Verlassen des durch das Funknetz abgedeckten Bereichs.

Es ist zu vermuten, dass die immer noch zunehmende Popularität mobiler Systeme zu komplexen Infrastrukturen enormer Größe führen wird. Aufgrund ihrer unterschiedlichen Charakteristik können die Ansätze zur Schaffung skalierbarer Infrastrukturen jedoch nicht ohne weiteres aus dem Bereich verteilter Systeme

übernommen und auf mobile Anwendungskontexte übertragen werden. Die Sicherstellung von *Skalierbarkeit* mobiler Infrastrukturen umfasst die Behandlung all derjenigen Probleme, die sich im Zusammenhang mit der Organisation von Benutzergruppen, mit der Datenhaltung, der Datenorganisation, der Datenverbreitung usw. ergeben.

Ähnliches gilt auch für das geänderte Informationsmedium. Die Tatsache, dass Benutzer im Zugriff auf Informationen nicht mehr an bestimmte Orte gebunden sind, an denen sie auf Informationen zugreifen können, führt zu einem geänderten Anspruch an die Informationen, die erfragt werden sollen. Sind in verteilten Systemen zumeist komplexere und umfassende Datenbestände zuzugreifen, so ist in mobilen Szenarien meist ein Bedarf nach kleineren Informationseinheiten vorhanden, die sich eher auf bestimmte Situationen oder Lokationen beziehen. Entsprechend muss für mobile Kontexte eine neue Herangehensweise bei der Aufbereitung von Daten, bei deren Segmentierung und Gruppierung entwickelt werden, die sich zum Teil grundsätzlich von bisherigen Ansätzen unterscheidet.

Einen elementaren Einfluss auf die Informationsaufbereitung haben schließlich die vergleichsweise limitierten Ressourcen, die in mobilen Geräten zum Tragen kommen. Die geringere Größe von Mobilgeräten führt zu einer geringeren Anzeigefläche, zu beschränkten Eingabemöglichkeiten, zu eingeschränkter Energieversorgung, eingeschränkter Bandbreite des Übertragungskanals usw. Diesem muss anhand einer angepassten Herangehensweise bei der Erstellung von Benutzungsschnittstellen, bei der Datenaufbereitung und –lokalisierung Rechnung getragen werden.

In einer genaueren Betrachtung der Probleme von Mobilität, Verbindungsabbrüchen und Skalierbarkeit identifizieren Badrinath und Imielinski weiterhin die Aspekte der *Topologie* und der *Datenlokation* als Problembereiche von Relevanz für mobile Kontexte:

Topologie : Verteilte Systeme gehen von einer festen Lokation für alle beteiligten Rechnerknoten aus, die sich während der gesamten Lebenszeit des Knotens nicht ändert oder allenfalls in einer sehr niedrigen Frequenz. Entsprechend besitzen verteilte Systeme eine statische *logische* Struktur bzw. Topologie (z.B. Ring, Baum, Netz usw.). Imielinski et al. merken dazu an, dass herkömmliche Algorithmen für verteilte Systeme überwiegend auf dieser logischen Struktur beruhen und in mobilen Systemen daher nicht uneingeschränkt zur Anwendung kommen können. In diesem Zusammenhang prägen die Autoren den Begriff des *Konfigurationsmanagement* als der Behandlung von Problemen, die durch die dynamische Struktur mobiler Netzwerke entstehen ([BI93], S. 12).

Datenlokation : Daten werden innerhalb eines mobilen Anwendungsszenarios nicht immer an derselben Stelle zu finden sein, selbst wenn sie auf demselben Gerät verbleiben. Bereits bei einer gleichbleibenden Topologie ergibt sich so die Notwendigkeit zu einer im Vergleich zu verteilten Systemen geänderten Herangehensweise, da in Fragen des Datenzugriffs und der Datenreplikation nicht von einer statischen Datenhaltungsstruktur ausgegangen werden kann. Imielinski et al. fassen Lösungen zu den hiermit im Zusammenhang stehenden Problemen unter dem Schlagwort des *Lokationsmanagements* zusammen ([BI93], S. 12)

2.3.1 Ressourcenknappheit

Die vergleichsweise geringe Ausstattung mit Ressourcen verschiedener Art stellt letztlich den fundamentalen Unterschied zwischen mobilen Systemen und herkömmlichen verteilten Systemen dar. Während in verteilten Systemen die Verfolgung von Benutzerzielen ganz im Vordergrund stehen kann, müssen für mobile Systeme die Optimierungskriterien der Gerätegröße, des Gerätegewichts und der Ausnutzung der vorhandenen Energie beachtet werden. Entsprechend sind besondere Algorithmen notwendig, um die stark begrenzten Ressourcen besser auszunutzen. Dieses kann sich in der Nutzung optimierter Berechnungsverfahren zu Reduktion der Rechenzyklen genauso zeigen, wie in der Nutzung besonderer Benutzungsschnittstellen, die trotz geringerer Grafikauflösung eine effiziente Bedienung des Geräts bzw. der Anwendung ermöglichen.

2.3.2 Eingeschränkte Sicherheit

Mobile Geräte sind gleich in mehrerer Hinsicht einem größeren Sicherheitsrisiko ausgesetzt als herkömmliche verteilte Systeme. Dieses kann sich zum einen auf das Gerät selbst, zum anderen auf die Daten beziehen, die auf dem Gerät gespeichert bzw. durch dieses verarbeitet werden. Im Gegensatz zu herkömmlichen Computersystemen besteht eine wesentlich höhere Wahrscheinlichkeit von Diebstählen, von Gerätedefekten durch Unfälle oder auch Unachtsamkeit. Schließlich ist die Datenübertragung zwischen dem Gerät selbst und dem Netzwerk, mit welchem das Gerät verbunden ist, deutlich höheren Sicherheitsrisiken ausgesetzt, die sich beispielsweise durch Abhören des Datenstroms o.ä. zeigen können. Das Risiko von Datenverlusten kann reduziert werden, indem auf eine lokale Speicherung der Daten auf dem mobilen Gerät weitgehend verzichtet wird. Im Falle eines Geräteschadens werden so nur mögliche Konfigurationsdaten in Mitleidenschaft gezogen, während die in der Informationsinfrastruktur gespeicherten persönlichen weiterhin existieren [FZ94].

2.3.3 Variierende Kommunikationsbedingungen

Im Gegensatz zu herkömmlichen verteilten Computersystemen spielt die drahtlose Datenübertragung bei mobilen Geräten eine wesentliche Rolle und mit ihr auch die wesentlich höhere Wahrscheinlichkeit von Verbindungsabbrüchen im Zuge der Datenkommunikation. Neben den *ungewollten* Abbrüchen, wie sie durch Kommunikationsfehler verursacht werden, ergeben sich für mobile Systeme auch *gewollte* Abbrüche, beispielsweise zur Reduktion des Energieverbrauchs oder bei Verlassen des vom Kommunikationssystem versorgten Gebiets.

Während drahtlose Kommunikationssysteme mittlerweile zum Teil Geschwindigkeiten erreicht haben, die vormals auch von festverbundenen Kommunikationsmedien nicht erbracht werden konnten, stehen sie diesen in ihrer Übertragungsrate in der Regel doch um mehrere Größenordnungen zurück. Insbesondere wirkt sich in mobilen Netzwerken problematisch aus, dass die zur Kommunikation genutzte Frequenz nicht von mehreren Teilnehmern gleichzeitig verwendet werden kann, wie es etwa beim herkömmlichen Ethernet der Fall ist. Entspre-

chend muss die ohnehin begrenzte Bandbreite bei einer Nutzung durch mehrere Kommunikationspartner aufgeteilt und so für alle reduziert werden.

2.3.4 Eingeschränkte Benutzungsschnittstelle

Mobile Geräte können aufgrund der Forderung nach minimalem Gewicht und minimaler Größe derzeit nicht mit Bildschirmen ausgerüstet werden, die eine zufriedenstellende Darstellung herkömmlicher Anwendungsschnittstellen erlauben würden. Sie besitzen eine geringere Auflösung, sind zumeist monochrom gehalten und weisen nur in seltenen Fällen horizontale bzw. vertikale Größenverhältnisse auf, die denen von herkömmlichen Computersystemen entsprechen würden. Benutzungsschnittstellen können daher nicht einfach übernommen werden, sondern müssen explizit für den Einsatz im mobilen Kontext entworfen werden.

2.3.5 Heterogenität

Gemäß der bereits angedeuteten Problematik ihrer Ressourcenarmut werden mobile Systeme meist auf bestimmte Anwendungskontexte hin optimiert. Dieses hat zu einer Vielzahl miteinander inkompatiblen und teilweise miteinander konkurrierenden Ansätzen geführt, die jeweils lediglich notwendige Hilfsmittel für einen bestimmten Anwendungsfall zur Verfügung stellen. Während in herkömmlichen Systemen die Bereitstellung von Kompatibilitätsschichten zur Adaption an andere Systeme üblich geworden ist, fallen diese bei mobilen Geräten aufgrund fehlender Systemressourcen zumeist weg.

2.3.6 Hohe Marktdynamizität

Der Bereich der mobilen Systeme hat sich zu einem Markt entwickelt, dessen Dynamik als bedeutendes Merkmal eingeschätzt werden muss. Die steigende Heterogenität zueinander inkompatibler Systeme und Ansätze stellt eine Herausforderung für die Anwendungsentwicklung dar. Während nämlich in bisherigen verteilten Systemen neu auf dem Markt erscheinende Systeme zumeist untereinander kompatibel waren und somit die Daten, Anwendungen, Methoden beim Übergang auf ein neues System übertragen werden konnten, ist dieses für mobile Systeme nurmehr in Ausnahmefällen möglich.

2.3.7 Eingeschränkter Nutzungskontext

Während die überwiegende Zahl der im Alltag verwendeten herkömmlichen Computersysteme als Generalisten zu bezeichnen sind, sie also für eine Vielzahl von Anwendungsfällen tauglich sind, ist dieses bei mobilen Systemen nicht der Fall. Um trotz der oben genannten Einschränkungen zu handhabbaren, kleinen Geräten zu kommen, die problemlos mit sich geführt werden können, weisen mobile Computer oftmals bereits in ihrem Design eine Einschränkung auf bestimmte Nutzungskontexte auf, welche ihre Verwendung in anderen Kontexten erschwert oder gar unmöglich macht. Erst die jüngste Generation von Persönlichen Digitalen Assistenten (PDAs) hat sich so weit entwickelt, dass eine gewisse Flexibilität gegeben ist.

2.4 Klassifikationskriterien für mobile Geräte

Wie kaum ein anderer Bereich der Computertechnik ist die Mobiltechnologie durch einen hohen Diversifikationsgrad geprägt. Während für konventionelle Systeme wie zum Beispiel Mainframe–Systeme, Workstations und insbesondere Arbeitsplatzsysteme zumeist deren möglichst *universelle* Einsetzbarkeit von übergeordnetem Interesse ist, so stellt sich die Situation für mobile Computersysteme im Vergleich nahezu orthogonal dar. Insofern, als diese Systeme vor allem hinsichtlich ihrer Handhabbarkeit in einem durch die Mobilität ihres Benutzers geprägten Umfeld bewertet und optimiert werden, musste die Generizitätsforderung zu Gunsten einer Optimierung auf den jeweiligen Anwendungskontext hin aufgegeben werden. Gilt für einen Arbeitsplatzrechner das Gebot, seine Leistung so weit zu steigern, dass er für eine möglichst große Anzahl *denkbarer* Anwendungsfälle einzusetzen ist, versucht man im Gegensatz dazu für Mobilgeräte, deren Leistung so weit zu senken, dass sie für den *konkreten* Kontext ausreichend ist, dabei jedoch ein Minimum an Ressourcen verbraucht, ein möglichst geringes Gewicht hat und eine minimale Anzahl möglichst robuster Komponenten enthält.

Im Entwurf eines Klassifikationsschemas für mobile Geräte stellt sich zunächst die Frage, anhand welcher Eigenschaften der Geräte dieses geschehen soll. Als mithin offensichtlichstes Merkmal lässt sich die Größe des Gerätes heranziehen. Weiser schlägt hierzu eine grobe Einteilung in drei Klassen vor (vgl. [Wei91], [Wei93] sowie [IK96], S. 49 ff.):

Inch–Size Scale : Geräte in einer Größenordnung bis zu wenigen Zentimetern. Beispiele hierfür wären kleine PDAs, Pager, Kommunikatoren oder erweiterte Mobiltelefone. Geräte dieser Klasse lassen sich überallhin mitführen, da sie sich ob ihres Größenfaktors auf das einfachste am Körper tragen bzw. in die Kleidung integrieren lassen.

Foot–Size Scale : Geräte in einer Größenordnung bis etwa DIN A4, wie zum Beispiel größere PDAs, Sub–Notebooks, elektronische Notizbücher oder elektronische Bücher (sog. eBooks). Geräte dieser Klasse bieten ein breiteres Anwendungsspektrum, da sie bezüglich ihrer Benutzungsschnittstelle weiteren Raum lassen. Andererseits sind sie nicht jederzeit mitzuführen und somit für Bedinungen ungeeignet, in welchen eine minimale Größe entscheidend ist.

Yard–Size Scale : Geräte in einer Größenordnung bis zu einigen Metern. In diese Kategorie fallen nach Weiser beispielsweise elektronische Wandtafeln. Geräte dieser Klasse werden vornehmlich im Bereich sogenannter intelligenter Gebäude [Oli92] zu finden sein, wo sie sich homogen in die Gebäudetechnik einpassen lassen.

Die hier vorgestellte Klassifikation entstammt dem Projekt ParcTab, in welchem am *Palo Alto Research Center* (PARC) der Firma XEROX Untersuchungen auf dem Gebiet der ubiquitären, d.h. *allgegenwärtigen* Computer durchgeführt wurden. Es ist offensichtlich, dass der Begriff der Allgegenwärtigkeit dabei nur zum

Teil mit dem in dieser Arbeit verwendeten Mobilitätsbegriff übereinstimmt[1]. So sind mit Geräten der Klasse der *Yard Scale* Systeme keine tragbaren Computer im eigentlichen Sinne gemeint, wenngleich diese in begrenztem Rahmen gleichwohl mobil sein könnten. [2] Da sich jedoch die generellen Charakteristika ubiquitärer Systeme insbesondere bezüglich der Benutzerinteraktion mit mobilen Systemen vergleichen lassen, ist dieses Schema als Ausgangspunkt geeignet anzusehen.

Andererseits lässt sich feststellen, dass der Interaktion mit der Umgebung eine wichtige Rolle in der Bereitstellung von Infrastrukturen für mobile Systeme zukommt: *„[...] we intend that the nomadic environment include the concept of an intelligent room.“* [Kle95]. Erst anhand von Rückmeldungen über den Ort, an dem sich das mobile Gerät befindet, und dessen *Wertigkeit* bezüglich der Wertmaßstäbe des Benutzers kann in geeigneter Weise reagiert werden, z.B. indem ortsabhängige Informationen oder Funktionen bereit gestellt werden (vgl. z.B. [DMCB98], [Fan99], [HS99]). Eine solche Interaktion wiederum ist nur auf der Basis einer geeigneten technischen Grundlage möglich, eben dem *Intelligent Building*.

2.4.1 Anbindung als Klassifikationskriterium

Ein weiterer Ansatz zur Klassifikation von Mobilgeräten nach Schiller [Sch00] bezieht sich weniger auf die Größe des betrachteten Gerätes, sondern zieht vielmehr die Qualität der Kommunikationsanbindung als wesentliches Kriterium heran. Schiller unterscheidet Geräte danach, ob sie verdrahtet bzw. nicht verdrahtet sind und weiterhin, ob sie dabei statisch sind oder mobil. Durch Kombination der beiden Merkmale ergeben sich folgende vier Klassen, anhand derer das Klassifikationsschema aufgespannt wird:

Verdrahtet / Statisch : In diese Gruppe fallen Computer im herkömmlichen Sinne, also PCs, Workstations, Mainframes usw. Diese Systeme sind über Kabelverbindungen an ein Netzwerk angeschlossen und dabei überwiegend an einen bestimmten Ort gebunden. Sie sind also gleichsam verdrahtet und ortsfest, d.h. statisch.

Verdrahtet / Mobil : Diese Gruppe umfasst den derzeit gängigen Stand der Kommunikation mobiler Computersysteme. So sind die meisten Laptops und insbesondere PDAs als mobil zu bezeichnen, da der Anwender sie problemlos bei sich führen kann. Diese Geräte verfügen jedoch über keine speziellen Fähigkeiten zur drahtlosen Kommunikation. Um eine Verbindung zu einem verteilten System aufzubauen, werden sie mittels eines gewöhnlichen Netzwerkkabels oder auch einer Dockingstation an das Netzwerk angeschlossen, um dann wie ein herkömmlicher PC zu kommunizieren. Sie sind also mobil, in ihrer Kommunikation jedoch verdrahtet.

Drahtlos / Statisch : Eine weitere Klasse wird durch all diejenigen Geräte gebildet, die zwar ortsgebunden sind, deren Kommunikation jedoch mittels

[1] Eine ausführlichere Betrachtung ubiquitärer Computersysteme und eine Gegenüberstellung zu mobilen Systemen erfolgt in Abschnitt 2.4.2.7.

[2] So ist eine elektronische Schreibtafel (sog. *White Board*) zwar oftmals auf Rollen zu bewegen, sie ist nicht jedoch für den Einsatz unterwegs gedacht.

drahtloser Kommunikationstechniken vonstatten geht. In diese Gruppe fallen insbesondere Arbeitsplatzrechner, die unter Verwendung eines drahtlosen Kommunikationsstandards wie z.B. IEEE 802.11 miteinander vernetzt sind. Die Kombination drahtloser Technologien mit statischen Computersystemen bietet sich beispielsweise dort an, wo Kabelsysteme aus baulichen oder zeitlichen Erwägungen nicht zum Einsatz kommen können.

Drahtlos / Mobil : In der Kombination aus drahtloser Kommunikation mit mobilen Computersystemen ergibt sich die vierte Kategorie des von Schiller vorgeschlagenen Schemas. Geräte dieser Kategorie können jederzeit vom Benutzer bei sich getragen werden und spontan mit ihrer Umgebung über drahtlose Kommunikationsprotokolle interagieren. Beispiel hierfür wären PDAs, die über das Bluetooth Protokoll mit einem Knotenpunkt des verteilten Systems in ihrer Reichweite Kontakt aufnehmen können.

Die in dieser Arbeit betrachteten Algorithmen und Technologien lassen sich im Wesentlichen im Bereich der beiden mobilen Kategorien ansiedeln.

2.4.2 Geräteklassen am Beispiel

Anhand konkreter Beispiele soll verdeutlicht werden, auf welche Weise mobile Geräte in Klassen eingeteilt werden können. Es ist zu erwarten, dass sich im Laufe der Zeit eine Verschiebung ergeben kann, sofern neue Gerätetypen auf den Markt kommen. Die exemplarischen Beispiele der folgenden Aufzählung wurden jedoch mit Bedacht so gewählt, dass sie mit großer Wahrscheinlichkeit auch in Zukunft als Anhaltspunkt für die Einordnung von mobilen Systemen dienen können.

2.4.2.1 Portable Computer

In seiner einfachsten Form lässt sich physische Mobilität durch eine Relokation von Computersystemen erzielen. Eine Maschine wird hierzu vom Netzwerk getrennt und an anderer Stelle wieder an das Netzwerk angeschlossen. Als Bedingung gilt dabei zunächst nur, dass das System im weitesten Sinne *beweglich* ist und dass die manuelle Rekonfiguration durch das verwendete Betriebssystem unterstützt wird.

Portabilität setzt keine andere Unterstützung durch die Hardware voraus als deren geringe Größe bzw. praktikable Handhabbarkeit. Vor allem werden keine besonderen Anforderungen an die Interaktion mit dem Benutzer gestellt, also etwa die Steuerung per Stift [CG96] oder gar Sprache [SS98]. Auch auf Seiten der Software stellt sich das portable Gerät im Wesentlichen wie ein Arbeitsplatzrechner dar. Da portable Computer keine Software–seitige Unterstützung für den Ortswechsel oder die Nutzung als autonomes System besitzen, kommen auf portablen Computern handelsübliche Betriebssysteme zur Anwendung. Insgesamt kann also zunächst einmal jeder Personalcomputer insbesondere durch leichte Änderungen an seinem Produktdesign zu einem portablen System gemacht werden.

Portable Systeme bieten den Vorteil, dass bestehende Anwendungen ohne konzeptionelle oder technische Anpassungen eingesetzt werden können. Andererseits sind sie nur sehr bedingt als mobile Systeme zu bezeichnen, da sie hierfür keine *aktive* Unterstützung anbieten.

2.4.2.2 Miniaturisierte PCs

Der Wunsch nach verbesserter Unterstützung einer *autonomen* Nutzung von Computersystemen hat zur Entwicklung miniaturisierter PCs geführt. Hierzu wurden Bauteile höher integriert sowie externe Komponenten wie Bildschirm oder Drucker in das System integriert. Viele dieser Systeme besitzen darüber hinaus einen Akku als eigene Energieversorgung, die ein Arbeiten auch *während* eines Ortswechsels möglich macht.

Miniaturisierte PCs bieten zumeist keine explizite Unterstützung für die Nutzung als Mobilgerät. In ihrer Benutzungsschnittstelle ähneln sie konventionellen PCs, wobei aufgrund der geringeren Abmaße zum Teil gewisse Einschränkungen bezüglich ihrer Interaktionsfähigkeiten bestehen. Grundsätzlich unterscheidet sich auch die auf miniaturisierten PCs verwendete Betriebssystem– und Anwendungssoftware nicht von derjenigen, wie sie auf üblichen Arbeitsplatzsystemen verwendet wird.

Miniaturisierte PCs kommen vor allem dort zum Einsatz, wo zwar eine gewisse Portabilität notwendig ist, um bestimmte Aufgaben auch außerhalb der Büroumgebung durchführen zu können, die jedoch die Benutzungsschnittstelle eines herkömmlichen Computersystems bedingen, wie beispielsweise Texterfassung oder Fahrzeugentwurf.

2.4.2.3 Laptops, Notebooks und Subnotebooks

Laptops, Notebooks und Subnotebooks stellen derzeit die größte Gruppe mobiler Geräte dar. Sie stellen die konsequente Fortführung des bereits für miniaturisierte PCs geltenden Integrationsbestrebens dar, welches vor dem Hintergrund weiterentwickelter Technik zu sehr kleinen und leichten Geräten geführt hat.

Laptops Laptops stellen in ihren Abmaßen die größten Systeme dieser Klasse dar. Sie stehen Arbeitsplatzrechnern in ihrer Leistungsfähigkeit mittlerweile nicht mehr wesentlich nach und werden in bestimmten Bereichen bereits als vollwertiger Ersatz angesehen. Neben ihrer Prozessorgeschwindigkeit und ihrem RAM– und Festplattenspeicher ist es nicht zuletzt die hohe Bildschirmauflösung, die Laptops auch für alltägliche Büroarbeiten einsetzbar gemacht hat. Dank leistungsfähiger Akkus können Laptops über mehrere Stunden ohne externe Stromversorgung betrieben werden und während dieser Zeit über Funk mit einem Netzwerk verbunden sein.

Notebooks Notebooks bilden eine Variante der Laptops. Sie weisen eine ähnliche Leistungscharakteristik auf, sind in ihren Abmessungen jedoch etwas kleiner und somit für den mobilen Einsatz noch besser geeignet. Wie Laptops weisen auch Notebooks eine für Computer klassische Benutzungsschnittstelle auf

und sind für die Mehrzahl der Anwendungsprogramme geeignet. Da sie ähnlich wie Laptops mit üblichen Betriebssystemen ausgestattet sind, besitzen sie keine über deren Fähigkeiten hinausgehende Unterstützung für den Einsatz in mobilen Szenarien.

Subnotebooks Subnotebooks sind die kleinsten auf den Prinzipien herkömmlicher Arbeitsplatzrechner basierenden Mobilsysteme. Wie auch Notebooks und Laptops integrieren sie Bildschirm, Tastatur und Energieversorgung, sind dabei jedoch nochmals kleiner in ihren Abmessungen. Als Erweiterung der aufgrund ihrer geringen Größe zum Teil schwer zu bedienenden Tastatur und Maus bieten einige Subnotebooks die Möglichkeit, per Stift auf dem drucksensitiven Bilschirm Menüpunkte auszuwählen oder Befehle einzugeben.

Insgesamt zeichnet sich diese Klasse an mobilen Geräten dadurch aus, dass sie zwar Dank ihrer geringen Abmaße einen hohen Grad an Portabilität aufweisen und aufgrund der eingebauten Stromversorgung auch während einer Relokation benutzt werden können, sie jedoch fast ausschließlich mit herkömmlichen Betriebssystemen verwendet werden und somit keine proaktive Unterstützung für den Einsatz in mobilen Kontexten besitzen. Insofern weisen sie nur einen graduellen Unterschied zu den zuvor beschriebenen portablen bzw. miniaturisierten PCs auf.

2.4.2.4 Persönliche Digitale Assistenten (PDAs)

Anfang der neunziger Jahre wurde begonnen, das Konzept einer universellen Einsatzmöglichkeit, wie es durch den Personalcomputer verkörpert wird, mit dem Konzept, ein Computersystem ausschließlich auf die Verwaltung persönlicher Daten eines einzelnen Benutzers abzustimmen, wie es bereits zuvor mittels so genannter Organizer möglich war, zu kombinieren. Die hieraus resultierenden *Personal Digital Assistants* (PDAs) oder auch „Palmtops“ (siehe [BAI93]) zeichneten sich durch ein vergleichsweise offenes, durch Programme von Drittanbietern erweiterbares Design aus, ähnelten in ihren Abmaßen und ihrem Design jedoch eher den typischen Organizern. Im Gegensatz zu in vorigen Abschnitten beschriebenen Systemen basieren PDAs auf einem eigenständigen Design, das sich sowohl auf die Hardware (vgl. z.B. [Cul94] bzw. [SW94]), als auch auf die Software [WSW+94] bezieht.

Bedingt durch ihre minimale Größe werden nur wenige PDAs mit einer Tastatur ausgerüstet. Üblicher ist hingegen die Benutzung mit einem Stift auf einem druckempfindlichen LC–Display und auch diejenigen Geräte, die über eine Tastatur verfügen, besitzen zumeist die Möglichkeit der Interaktion mittels Stift. Die Hardware zeichnet sich zumeist dadurch aus, dass sie auf die ansonsten üblichen Festplattenspeicher verzichten. Alle Daten werden in persistenten RAM–Speichern abgelegt, was zu einer wesentlichen Gewichtsersparnis führt, andererseits jedoch in einer vergleichsweise geringen Speicherkapazität resultiert.

Im Vergleich zu Notebooks besitzen PDAs ein vergleichsweise gut auf die speziellen Bedingungen des mobilen Einsatzes abgestimmtes Betriebssystem. Dieses bietet neben einer inhärenten Unterstützung für Eingabegeräte wie Stift oder gar Sprache zumeist eine explizite Berücksichtigung der beschränkten Ressourcen des Gerätes. Daten sind oftmals über spezielle Dateisysteme organisiert, die

eine minimale Redundanz ermöglichen und so zu einer effizienten Datenhaltung beitragen. Des Weiteren können PDAs zumeist den Zustand laufender Anwendungen persistieren, sofern sie abgeschaltet werden sollen. Über Infrarot– oder Funkschnittstellen (wie z.B. IrDA oder Bluetooth) können sie ohne die Notwendigkeit einer externen Netzwerkinfrastruktur Daten mit anderen PDAs austauschen. Im Gegensatz zu herkömmlichen Systemen wird dabei oft von *Dateien* als kleinsten übertragbaren Einheiten abgerückt und an ihrer Stelle auf intuitivere Einheiten wie beispielsweise *Adressen* oder *Visitenkarten* zurückgegriffen. Modernere Systeme sind schließlich bereits mit Funknetzmodulen ausgerüstet, über die jederzeit auf entfernte Daten zugegriffen werden kann.

2.4.2.5 Mobiltelefone

Mobiltelefone haben unter den mobilen Geräten den höchsten Verbreitungsgrad erreicht. In vielen Ländern haben sie einen so hohen Durchdringungsgrad erreicht, dass sie nicht länger bewusst wahrgenommen werden (vgl. [Cox99], S. 15-2 ff.). Mobiltelefone werden weltweit von Privatpersonen genauso wie von Geschäftsleuten für die mobile Kommunikation eingesetzt und stellen in ihrer Größe, Verfügbarkeit und Handhabbarkeit nahezu das Ideal eines Mobilgerätes dar. Es ist vor allem diese Klasse von Mobilgeräten, welche den Begriff der *Wireless Personal Communication Systems* [Goo97] geprägt hat.

Mobiltelefone haben in der Mehrzahl eine sehr eingeschränkte Benutzungsschnittstelle. Ihre Anzeige ist auf wenige Zeichen bzw. Zeilen beschränkt und auch Geräte mit höheren Grafikfähigkeiten erreichen derzeit selten Auflösungen von über 10 Prozent derjenigen eines Laptops. Das Tastenfeld besitzt selten mehr als zwölf Tasten, wobei diese zumeist drei– bis vierfach überbelegt sind. Mobiltelefone sind überwiegend *geschlossene* Systeme, deren Betriebssystem und Anwendungssoftware vom Hersteller vorinstalliert ist und nicht von Drittanbietern modifiziert werden kann.

Mobiltelefone dienen primär der Sprachkommunikation und sind in ihrem Design auf diesen Anwendungsfall hin optimiert. In Verbindung mit einem Laptop oder PDA können viele Mobiltelefone darüber hinaus auch zur drahtlosen Übertragung von Daten genutzt werden, wobei das Mobiltelefon in diesem Falle als Modem fungiert. Mit Hilfe des sog. *Short Message System* (SMS) können schließlich direkt auf dem Telefon kurze Nachrichten verfasst und an SMS–fähige Gegenstellen geschickt werden. Auf der Basis von Kurznachrichten lassen sich Mobiltelefone in bestehende Informationssysteme einbinden, wobei auf dem Telefon einfache Befehle formuliert und an den Informationsdienst geschickt werden. Dieser interpretiert die Befehle, führt sie aus und sendet eventuelle Ergebnisse zurück an das Telefon (vgl. z.B. [SJ98]). Aufgrund ihrer fehlenden Ausdrucksmächtigkeit und der derzeit wenig intuitiven Interaktionsfähigkeiten mobiler Telefone eignet sich diese Art der Einbindung jedoch nur für eine sehr eingeschränkte Menge von Anwendungen.

Kommunikatoren Eine Mischung aus Mobiltelefonen und PDAs stellen die so genannten Kommunikatoren dar. Bei ihnen wurde das Design des Mobiltelefon so weit modifiziert, dass neben den Fähigkeiten eines klassischen Telefons auch die Funktionalität eines digitalen Assistenten integriert werden konnte.

Entsprechend verfügen Kommunikatoren über die Möglichkeit zur Verwaltung von Adressen, zur Versendung von Emails bis hin zur Möglichkeit das World Wide Web zu nutzen (vgl. [HKR99]). Kommunikatoren sind zumeist mit PDA Betriebssystemen ausgestattet und verfügen somit einerseits über *offene* Programmierschnittstellen und können darüber hinaus oftmals direkt mit für den jeweiligen PDA entwickelten Programmen betrieben werden. Als Mobiltelefone verfügen sie über integrierte Kommunikationsschnittstellen, über welche ein direkter Zugriff auf entfernte Daten möglich ist. Als PDAs verfügen sie hingegen über eine größere Anzeige und können entweder per Stift oder über eine Tastatur bedient werden. Die Benutzung eines Kommunikators ist somit wesentlich einfacher als die Interaktion mit einem Mobiltelefon.

WAP–Telefone Um die Beschränkungen des *Short Message System* zu überwinden und eine zur Benutzung des *World Wide Web* vergleichbare, *interaktive* Datenkommunikation mittels Mobiltelefonen zu ermöglichen, wurde das *Wireless Application Protocol* (WAP) entworfen. Basierend auf einer auf die Einschränkungen mobiler Telefone abgestimmten Seitenbeschreibungssprache, der *Wireless Markup Language* (WML) [HTK98] und unter Nutzung von sog. *WAP Gateways* [MW00a] als speziellen Kommunikationsknotenpunken, ist es mit WAP–fähigen Telefonen [MW00b] möglich, flexibel auf bestehende Informationssysteme zuzugreifen.

WAP–fähige Telefone unterscheiden sich äußerlich nicht von herkömmlichen Mobiltelefonen. Sie weisen dieselben Außenmaße auf und besitzen dieselben eingeschränkten Interaktionsmöglichkeiten. Der einfache Zugriff auf WML–Dokumente lässt sie jedoch dort zu einer guten Alternative zu Kommunikatoren oder PDAs werden, wo die Interaktion mit dem Informationssystem primär lesend ist und die Nutzung des Gerätes nicht ohne Verbindung zum Funknetz zu erfolgen hat.

2.4.2.6 Wearables

Eine besondere Rolle unter den mobilen Geräten spielen die so genannten *wearables*, Computersysteme also, welche von ihrem Benutzer ständig am Körper getragen werden, bzw. welche zum Teil sogar in seine Kleidung integriert sind [Hum00]. Wie auch persönliche digitale Assistenten dienen Wearables zur Unterstützung eines *bestimmten* Benutzers, dabei unterscheiden sich PDAs und Wearables jedoch in einigen wesentlichen Merkmalen. Als wichtigstes Merkmal lässt sich feststellen, dass PDAs oder Laptops zumindest für kürzere Zeitspannen die volle Aufmerksamkeit des Benutzers binden und somit für gewisse Zeit zu einem Mittelpunkt seines Handelns werden. Wearables sollen hingegen lediglich bei der Ausführung einer Handlung *unterstützen*, ohne dabei jedoch die Aufmerksamkeit von eben dieser Handlung abzulenken: *... Traditional computing paradigms are based on the notion that computing is the primary task. Wearable computing, however, is based on the notion that computing is NOT the primary task. [Man98].* Nach Mann gelten als wesentlich für Wearables insbesondere die folgenden Eigenschaften (vgl. [Man98]):

Konstanz („Constancy“) : Im Gegensatz zu Notebooks oder PDAs gilt für Wearables die *permanente* Verfügbarkeit als oberstes Designkriterium.

Während PDAs und, mehr noch, Laptops beim Start zunächst ein Betriebssystem starten oder eine Anwendungssoftware laden müssen, bevor sie einsatzbereit sind, muss ein Wearable *jederzeit* ohne Zeitverzögerung nutzbar sein.

Unterstützung („Augmentation“) : Wearables sollen den Benutzer in seiner Arbeit unterstützen, ohne dabei seine volle Aufmerksamkeit zu benötigen. Während ein PDA mit beiden Händen bedient werden muss, soll ein Wearable möglichst mittels verbaler Kommandos steuerbar sein und anhand von Audiosignalen seinen Zustand mitteilen können (vgl. [SS98]). Während es nicht möglich ist, eine weitere Aufgabe zu erledigen, während ein Laptop betrachtet wird, kann dieses bei Wearables durch die Einspielung der Anzeige in das Blickfeld des Benutzers vermieden werden, bzw. sogar die Sicht des Benutzers angepasst werden, um vorhandene visuelle Defizite zu kompensieren (vgl. [SMR+97]).

Vermittlung („Mediation“) : Insofern, als sie direkt am Körper getragen werden, können Wearables unsere Sinne von der Außenwelt abschirmen und somit filtern, was über die Sinne aufgenommen wird. Während PDAs lediglich dem aktiven Zugriff auf Daten dienen können, können Wearables auch Informationen zurückhalten, die in einem bestimmten Augenblick nicht aufgenommen werden sollen.

Während Wearables technisch im Wesentlichen miniaturisierten PCs entsprechen, sind es vor allem die Ein– und Ausgabegeräte auf der einen Seite sowie die Anwendungen auf der anderen Seite, die sie von PDAs oder Laptops unterscheiden. Als Eingabegeräte werden Sprachsteuerungen oder mit der Hand zu bedienende Spezialgeräte benutzt. Als Ausgabegeräte fungieren zumeist auf dem Kopf zu tragende Bildschirme, deren Signal mittels Brille in das Sichtfeld des Benutzers eingeblendet wird [BKM+97] [SS99]. Wearables unterbrechen somit nicht den Interaktionsfluss, sondern werden in ihn einbezogen.

Der Benutzer soll einen Wearable Computer nicht nur zwecks Zugriff auf Informationsquellen nutzen, sondern generell in seiner Wahrnehmung und seinen Handlungen durch ihn *unterstützt* werden. Anwendungen für Wearables sollen demgemäß Information nur so weit aufbereiten, wie dieses für die audiovisuelle Präsentation notwendig ist. Im Gegensatz zu den in der *künstlichen Intelligenz* vorherrschenden Ansätzen soll im Wearable Computing jedoch nicht versucht werden, die menschliche Intelligenz zu *emulieren*. *„[...] Rather than attempting to emulate human intelligence, as it is a common goal of research in Artificial Intelligence (AI), the goal of wearable computing is to produce a synergistic combination of human and machine, in which the human performs tasks that it is better at, while the computer performs tasks that it is better at. [...]“* [Man98]. Der Mensch soll also vielmehr in seiner Willensbildung unterstützt werden. Mann prägt hierzu den Begriff der *Humanistic Intelligence (HI)*, die aus der durch Wearable Computing realisierten Synergie zwischen Mensch und Maschine erwächst. Letztlich soll durch den Einsatz von Wearables der Entscheidungsprozess gefördert und die aktive Gestaltung der Zukunft unterstützt werden [PR99].

Affective bzw. Situated Computing Die Nutzung der durch biometrische Sensoren aufgenommenen Informationen *über* den Benutzer des Wearable Computers wird in einer speziellen Form des *wearable computings*, des sogenannten *Affective* oder auch *Situated Computing* betrachtet [PH97]. Hierbei werden Parameter wie Hautwiderstand, Herzfrequenz oder Atemfrequenz des Anwenders vom Computer aufgenommen und analysiert. Entsprechend eines zuvor festgelegten Programms kann sodann eine Reaktion anhand der erfassten Messwerte erfolgen und quasi eine Rückkopplung zwischen Mensch und Maschine erfolgen. Beispielsweise könnte beruhigende Musik eingespielt werden, sofern die Sensoren Stress–Symptome aufgenommen haben, um den vom Computer erkannten Stress abzubauen [PH97]. Neben den bereits für PDAs bekannten *Lokationskontext* [HHS+99] tritt hierbei ein *Situationskontext* [SSR+99]. Computer können Emotionen über ihre Sensorik aufnehmen und entsprechend unterstützend oder gar lenkend eingreifen.

2.4.2.7 Ubiquitäre Systeme

Ubiquitäre Systeme stellen keine Form des Mobile Computing im eigentlichen Sinne dar. Insbesondere in der populären Literatur erscheinen die Begriffe austauschbar, andererseits werden sie in der Fachliteratur teilweise gar als zueinander orthogonal bezeichnet [RMW99]. Nach einer frühen Definition von Weiser [Wei91] bilden ubiquitäre Systeme eine Obermenge, unter welcher sich das Mobile Computing als eine Spezialisierung subsummieren lässt. Weiser führt hierzu folgendes an: *„Ubiquitous Computing in this context does not just mean computers that can be carried to the beach, jungle or airport. Even the most powerful notebook computer, with access to a worldwide information network, still focuses attention on a single box.“*

Nach Weiser ist es vor allem ein Wechsel der Betrachtungsebene, der für ubiquitäre Systeme in Anspruch genommen werden muss. Im Gegensatz zu herkömmlichen Computersystemen sollen ubiquitäre Systeme nicht als Computer in den Vordergrund treten, sondern vielmehr in ihrer Funktion als Träger von Information erscheinen. Und für diesen Zweck kann möglicherweise — nicht jedoch zwangsläufig — ein mobiles System dienen. Weiser hierzu: *„[...] Ubiquitous computers will also come in different sizes, each suited to a particular task. [...]“*. Primär wird im Falle ubiquitärer Systeme also der *Anwendungsfall* und nicht das technische Werkzeug *per se* betont. Als mögliche Größen für diese Systeme werden von Weiser die bereits unter 2.4 beschriebenen Kategorien *Inch–Size*, *Foot–Size* sowie *Yard–Size* vorgeschlagen.

Als ein wesentliches Merkmal des *Ubiquitous Computing* lässt sich die Eigenschaft feststellen, sich nicht in einem einzelnen Hilfsmittel zu manifestieren. Es ist vielmehr die Kombination aus einer Vielzahl stationärer sowie mobiler Geräte, die in ihrer Gesamtheit zu einem nicht länger bewusst wahrgenommenen, allgegenwärtigen Werkzeug werden. Weiser prägt hierzu den Begriff der *embodied virtuality*, welche durch deren geeignete Aggregation zur Schaffung von Computer Systemen führt, die über die Summe ihrer Komponenten hinauswachsen [Wei91].

Ubiquitäre Systeme lassen sich in zwei Hauptgruppen unterteilen. Auf der einen Seite stehen die *stationären Systeme*, die Informationen aus einem permanenten

Netzwerk beziehen und in geeigneter Form an den Benutzer weiterleiten können. Stationäre Systeme können hierbei der *direkten* Interaktion dienen, wie es beispielsweise bei fest montierten Sichtgeräten oder bei elektronischen Schreibtafeln der Fall ist (vgl. [IK96]). Gemäß ihres an herkömmlichen Möbeln orientierten Designs werden diese Systeme auch als *Roomware* bezeichnet [HS99]. Die sie enthaltenden Räume werden in der Literatur zum Teil unter dem Begriff des *Intelligent Room* erwähnt [BCKP95].

Stationäre Systeme können auch der *indirekten* Interaktion dienen. So können Sender und Empfänger Informationen mit mobilen Geräten austauschen [AGS+93] oder diese mit Informationen über Lokation und Kontext des Ortes versorgen, an welchem sie sich befinden [HHS+99]. Hierbei bilden sie eine Plattform, auf der die eigentliche Interaktion mit dem Benutzer stattfindet.

Auf der anderen Seite stehen die *mobilen Systeme*, mit denen der Benutzer interagiert. Mobile Systeme müssen dabei nicht unbedingt Geräte sein, die ständig mit sich getragen werden. Als wichtiges Beispiel hierfür mag etwa das sog. *Pad* dienen, das in verschiedenen Projekten betrachtet wurde [BBB+94], [Bro99], [JWDH95]. Pads können wie Notizblöcke mit sich geführt werden und zur Datenein– bzw. ausgabe dienen. Ähnlich wie Notizblöcke werden jedoch auch Pads nicht *permanent* am Körper getragen, sondern sind vielmehr primär mit einer Aufgabe assoziiert und werden nur zu deren Bearbeitung eingesetzt. Verschiedene Aufgaben können dann ggf. mittels verschiedener Pads bearbeitet werden.

2.5 Zusammenfassung

Dieses Kapitel galt einer ersten Standortbestimmung im überaus vielschichtigen Umfeld der Mobilität. Ausgegangen wurde dabei von der Unterscheidung zwischen sozialer und physischer Mobilität. Wenngleich die soziale Mobilität, also der Übergang des Menschen von einer gesellschaftlichen Position zu einer anderen, in der Vergangenheit einen enormen Einfluss auf unser Denken und Handeln hatte und auch heute noch hat, so soll sie dennoch in dieser Arbeit keine Betrachtung finden. Es geht hier ausschließlich um die räumliche Änderung, die der Mensch als Anwender der elektronischen Datenverarbeitung oder das von ihm benutzte Gerät erfahren können.

Demgemäß wurde im weiteren Verlauf dieses Kapitels auf Spielarten der Mobilität im Zusammenhang mit der Datenverarbeitung eingegangen. Es wurde der Unterschied zwischen Portabilität und Mobilität aufgezeigt und dargestellt, wie sich portable Computer von mobilen Computern unterscheiden. Es wurde weiterhin verdeutlicht, dass es neben der offensichtlichen, physischen Mobilität durch die Relokation eines Computersystems auch eine zweite, nicht direkt sichtbare Mobilität gibt, die als Mobilität der Software bezeichnet werden kann. Softwaremobilität ist eine Form der Mobilität, die zum Teil zur Unterstützung der Hardwaremobilität dient, teilweise aber auch von dieser völlig unabhängig ist. Mit der Erläuterung von Mobiler Kommunikation, von Mobile Computation und von Mobile Computing wurden drei wichtige Ausprägungen von Mobilität umrissen, die der einen wie der anderen Grundform zugerechnet werden können.

Im weiteren Verlauf wurde dann detailliert auf die besonderen Merkmale der mobilen Datenverarbeitung eingegangen, welche sie von der herkömmlichen Datenverarbeitung abheben. Insbesondere die Faktoren der Ressourcenknappheit, der eingeschränkten Schnittstelle sowie des geschlossenen Designs sorgen dafür, dass die Realisierung von Anwendungen für mobile Geräte ebenso anspruchsvoll wie interessant erscheint.

Anschließend wurden mehrere Ansätze zur Klassifikation mobiler Geräte vorgestellt. Neben der Größe als Ordnungskriterium, wie sie von Weiser vorgeschlagen wurde, sind es vor allem die von Schiller vorgeschlagene Beweglichkeit des Gerätes und seine Anbindung an ein bestehendes Netzwerk, die als Grundlage der Einordnung dienen können. Beide Taxonomien sind offensichtlich recht allgemein gehalten, bieten dafür jedoch den Vorteil, trotz der dynamischen Entwicklung verfügbarer Mobilgeräte auch weiterhin anwendbar zu bleiben.

Zum Abschluss des Kapitels wurde schließlich anhand von Beispielen versucht, die Geräteklassen zu illustrieren. In einem Spektrum zwischen portablen Computern auf der einen Seite und ubiquitären bzw. in die Kleidung des Menschen integrierten *Wearables* auf der anderen wurden mögliche Ausprägungen dargestellt, die das Mobile Computing besitzen kann.

Nachdem dieses Kapitel der ersten Annäherung an den Themenkomplex der mobilen Datenverarbeitung diente, wichtige Begrifflichkeiten geklärt, Abgrenzungen durchgeführt und Vorschläge für eine Klassifikation vorgestellt wurden, soll im nächsten Kapitel die Integration mobiler Geräte in verteilte Umgebungen als ein elementarer Baustein für deren effiziente Nutzung näher beleuchtet werden.

Kapitel 3

Integrationsstrategien für mobile Geräte

Nachdem im vorigen Kapitel ein erster Überblick auf die Themenvielfalt im Bereich der Mobilität gegeben wurde und verschiedene Ansätze zur Kategorisierung mobiler Systeme näher beleuchtet wurden, soll nunmehr näher auf deren Integration in verteilte Systemumgebungen eingegangen werden. Insbesondere wird dieses Kapitel zunächst der Behandlung der Frage dienen, *warum* es notwendig bzw. angebracht ist, mobile Systeme in Bezug auf deren Einbeziehung in verteilte Infrastrukturen gesondert zu betrachten, sich hierbei also nicht ohne weiteres der aus dem Bereich der verteilten Systeme hinlänglich bekannten Ansätze bzw. Algorithmen zu bedienen. Es werden unter anderem die für mobile Systeme spezifischen Probleme genannt sowie näher untersucht.

Eingedenk der Vielfalt möglicherweise auftretender Probleme lässt sich die Integration mobiler Systeme auf verschiedene, konzeptionell grundlegend verschiedene Arten betrachten und behandeln. Angefangen von einer Integration auf der Ebene niederer Netzwerkkommunikation, über eine Behandlung der Integrationsproblematik auf der Ebene der Bereitstellung von Programmierschnittstellen im Rahmen der Anwendungsentwicklung, der Realisierung kompletter Laufzeitumgebungen, bis hin zur Abbildung der grafischen Anwendungsschnittstelle auf das mobile Gerät, ist eine Vielzahl möglicher Herangehensweisen denkbar. Entsprechend wird dieses Kapitel die Integrationsebenen identifizieren und die für sie charakteristischen Probleme zuordnen.

Dabei soll anhand einer Einordnung in das ISO *Open System Interconnection* (kurz: OSI) Modells [Ker92] ein Überblick verschiedener Integrationskonzepte für mobile Geräte gegeben werden. Insbesondere werden Konzepte vorgestellt, die sich der Sicherungsschicht (AIRMAIL, Multi Service Link), der Netzwerkschicht (Mobile IPv4/Mobile IPv6) sowie der Transportschicht (u.a. RESOA, EBSN) zurechnen lassen. Die Darstellung von Ansätzen auf den beiden höchsten Ebenen des Modells wird dagegen auf das nächste Kapitel verschoben, in welchem dann ausschließlich Verfahren aus der Präsentations– sowie Anwendungsschicht behandelt werden.

3.1 Ziele der Geräteintegration

Als generelles Ziel der Integration mobiler Systeme in verteilte Infrastrukturen kann zunächst der Wunsch genannt werden, von *beliebigen* Orten auf die innerhalb der Infrastruktur verfügbaren Dienste und die von ihnen verwalteten Daten zuzugreifen, diese zu lesen, zu modifizieren, zu verknüpfen usw. Während in herkömmlichen Netzwerken hierfür wohldefinierte Zugriffspunkte zur Verfügung stehen, müssen bei Einbeziehung mobiler Geräte *beliebige* Zugriffspunkte unterstützt werden. Jing bringt diese Forderung auf die folgende Kurzform:

> „[...] users who carry portable devices have access to information services through a shared infrastructure regardless of their physical location or movement behavior.“ [JHE99b]

Die freie Orts– und Gerätewahl wird in der Literatur als *Transparenz* bezeichnet. Sowohl aus der Sicht des Benutzers als auch aus der Sicht der Informationsinfrastruktur soll vom benutzten Gerät und von der Art bzw. Qualität des Kommunikationskanals abstrahiert werden können. Bagrodia und Kleinrock fassen dieses im Begriff des *Transparent Virtual Networking* zusammen [BCKP95, Kle95]. Transparent Virtual Networking bezeichnet einen Zustand, in welchem Benutzern und den durch sie gestarteten Anwendungsprogrammen ein maximaler Grad an Effektivität ermöglicht wird, ohne dass dabei ihr Operationsmodus beeinflusst wird. *Vollständige* Transparenz stellt einen Idealzustand dar, der nur angenähert, nicht jedoch uneingeschränkt erreicht werden kann (vgl. auch [SKM+93]).

Variationen in z.B. der Verbindungsqualität sollen keinen grundsätzlichen Einfluss auf den Informationszugriff haben. Vielmehr soll die verwendete Infrastruktur in der Lage sein, die Daten ggf. anzupassen, um einen entsprechend der derzeitigen Situation optimalen Nutzungsgrad zu ermöglichen und die Transparenzforderung bestmöglich zu erfüllen. Kleinrock fasst die zur Annäherung des Ideals zu beachtenden Aspekte unter dem Schlagwort der *Nomadicity* zusammen, einer Verschmelzung von Aspekten des *Nomadic Computing* sowie der *Computer Communication*:

> „Nomadicity may be defined as the system support needed to provide a rich set of computing and communication capabilities and services to nomads as they move from place to place in a transparent, integrated and convenient form.“ [Kle95]

Die Integration soll aus der Sicht des Anwenders weitestgehend ohne sein Eingreifen vonstatten gehen. Wird zu verschiedenen Zeitpunkten von unterschiedlichen Orten zugegriffen, so muss das Gerät bzw. die Infrastruktur autonom in der Lage sein, die Kommunikationsverbindung anzupassen bzw. den Informationsfluss umzulenken. Eine der Grundeigenschaften mobiler Umgebungen ist die fehlende Qualitätsgarantie bezüglich der zur Verfügung stehenden Kommunikationskanäle. Bewegt sich das mobile Gerät aus dem Empfangsbereich eines bestimmten Kommunikationskanals, so kann es sein, dass die Datenübertragung auf einen alternativen Kanal umgestellt werden muss und nurmehr mit verminderter Geschwindigkeit stattfinden kann. Eventuell steht sogar kein alternatives

Übertragungsmedium zur Verfügung, und die Verbindung muss vorübergehend unterbrochen werden. Ziel der Integration muss es insofern sein, die Interaktion zwischen dem Mobilgerät und der Informationsinfrastruktur den geänderten Bedingungen anzupassen und diese in die Interaktion einzubeziehen.

3.2 Bewertungskriterien für Integrationsansätze

Es ist festzustellen, dass die Integration mobiler Systeme in verteilte Systemumgebungen eine Vielzahl spezifischer Probleme bereitet, welche als Teil einer Integrationsumgebung zur Unterstützung mobiler Systeme behandelt werden müssen. Das vorige Kapitel ist auf diese Probleme bereits im Überblick eingegangen. In der Vergangenheit wurden in einer Reihe von Forschungsarbeiten Integrationsansätze vorgestellt, die jeweils eine Teilmenge dieser Probleme behandeln. Zum einen ist dieses dadurch bedingt, dass die Menge möglicher Probleme zu groß ist, um sie in einem einzigen Konzept zu behandeln. Vor allem aber sind viele Problembereiche grundsätzlich *orthogonal* zueinander und somit nur in verschiedenen Konzepten, bzw. auf verschiedenen Ebenen anzugehen.

Bei der Beschreibung verschiedener Ansätze, wie sie in diesem Kapitel erfolgen soll, stellt sich nun die Frage nach der Systematik, nach welcher dieses zu geschehen hat. Um eine genauere Positionsbestimmung eines Ansatzes vornehmen zu können, ist ein genügend feines Bewertungsschema notwendig. Ein solches Schema sollte möglichst allgemein gehalten sein, um auch zukünftige Ansätze einbeziehen zu können. Es sollte sich außerdem möglichst auch zur Klassifikation nicht primär der Integration *mobiler* Systeme dienender Ansätze eignen, um ggf. einen Vergleich mit diesen zuzulassen.

Im Bereich der Kommunikationstechnik hat es sich bewährt, Ansätze zur Interaktion zwischen Netzwerkknoten anhand des OSI–Referenzmodells [OSI94] einzuordnen. Das OSI–Referenzmodell gruppiert die Aspekte der Kommunikation zwischen Netzwerkknoten in sieben Gruppen, die, entsprechend einer Modellsicht, als *Schichten* des Modells bezeichnet werden. Im einzelnen werden folgende Schichten definiert:

Bitübertragungsschicht : Behandlung von Problemen, die mit der Übertragung unstrukturierter Daten über eine Datenleitung zusammenhängen (z.B. Spannungspegel, Timing usw).

Sicherungsschicht : Behandlung von Problemen, die mit der verlässlichen Übertragung von Daten über die Datenleitung zusammenhängen (z.B. Organisation von Daten in Datenpaketen, Fehlerbehandlung, Flusssteuerung usw.)

Netzwerkschicht : Abstraktion von zugrunde liegenden Übertragungstechnologien und Bereitstellung von logischen Verbindungen zwischen Netzwerkknoten (z.B. Aufbau, Verwaltung und Abbau von Verbindungen)

Transportschicht : Behandlung von Problemen, die mit der Realisierung verlässlicher Ende–zu–Ende–Verbindungen zwischen Netzwerkknoten auftreten (z.B. Fehlerbehebung, Flusssteuerung usw.)

Sitzungsschicht : Aufbau, Verwaltung und Abbau von Sitzungen zwischen kooperierenden Anwendungen verschiedener Netzwerkknoten (z.B. Dialogsteuerung, Synchronisation usw.).

Präsentationsschicht : Bereitstellung von einheitlichen Datenrepräsentationen zwischen Netzwerkknoten (z.B. Verschlüsselung, Kodierung von speziellen Zeichen usw.)

Anwendungsschicht : Bereitstellung von Anwendungsdiensten, auf die innerhalb eines Netzwerkes durch kooperierende Netzwerkknoten eingesetzt werden können (z.B. Transaktionsdienst, Mail–Server usw.)

Für eine detaillierte Beschreibung des OSI–Referenzmodells sei auf entsprechend weiterführende Literatur verwiesen (z.B. [Sta87], [Ker92] oder [Tan96]). Wenngleich das OSI–Modell ursprünglich als Grundlage für konkrete Implementationen konzipiert wurde, so entspricht es doch durch seinen Umfang und seinen allgemeinen Ansatz den Ansprüchen an eine Taxonomie. Die Einordnung von Kommunikationssystemen nach dem OSI Modell stellt ein gängiges Vorgehen dar.

Bevor im Folgenden verschiedene Integrationsansätze beschrieben und den Ebenen des OSI–Modells zugeordnet werden, sollen jedoch alternative Klassifikationsmodelle angeführt werden, anhand derer eine Einordnung vonstatten gehen kann.

3.2.1 Awareness und Adaptation

Herkömmliche Computersysteme sind zumeist stationär. Sie werden an einem bestimmten Ort aufgestellt und an diesem Ort benutzt. Ortswechsel sind bei solchen Systemen eher unüblich und kommen somit vergleichsweise selten vor. Entsprechend geringe Bedeutung besitzen Informationen über die Umgebung des Computersystems auf dessen Benutzung. Da sich Ort, Umfeld, Kontext usw. im Verlaufe des Einsatzes kaum ändern, muss also nicht dynamisch auf sie eingegangen werden, um den Nutzen des Systemes zu optimieren. Wo Umgebungsparameter einen Einfluss auf stationäre Computersysteme besitzen, können sie bei der Installation berücksichtigt und entsprechend konfiguriert werden.

Im Falle von mobilen Systemen stellt sich die Situation grundlegend anders dar. Mobilgeräte besitzen die quasi *inhärente* Eigenschaft, ihren Einsatzort des öfteren zu wechseln, insbesondere bei PDAs sogar während deren Benutzung. Von umso höherer Relevanz sind demgemäß die verschiedenen Umgebungsparameter, welche sich im Zuge der Benutzung ändern können und deren Werte einen Einfluss auf die Nutzung des Gerätes besitzen können.

Die Einbeziehung der sich ändernden Umgebungsbedingungen wird in der Literatur zumeist als *Awareness* bezeichnet [SNKP94], die eigentliche Reaktion auf sich ändernde Bedingungen als *Adaptation*. Awareness und Adaptation, zusammen auch als *Adaptivity* bezeichnet, wurden bereits sehr früh als die wesentlichen Funktionalitäten einer jeden Integrationsumgebung im mobilen Kontext erkannt [JHE99b, Kat94, Sat95].

Awareness bezieht sich auf eine Vielzahl von Aspekten des mobilen Umfeldes, die sich im Zuge der Nutzung eines mobilen Gerätes ändern können und deren Beachtung einen Mehrwert für diese Nutzung schaffen kann. Ressourcen lassen sich entsprechend ihrer Relevanz für die einzelne Anwendung bzw. für das gesamte Computersystem entweder als *generisch* oder als *typspezifisch* bezeichnen [NPS95]. Generische Ressourcen besitzen eine Bedeutung für *alle* innerhalb des Gerätes bzw. der Umgebung laufenden Anwendungen. Noble et al. nennen als Beispiel hierfür unter anderem den verfügbaren Speicher, die zur Verfügung stehende Batteriespannung oder die Geschwindigkeit der Funknetzverbindung.

Typspezifische Ressourcen sind hingegen nur für bestimmte Anwendungen von Bedeutung. Die Werte der zu diesen Ressourcen verwalteten Parameter sind nur im Kontext dieser Anwendung zu verwenden und könnten im Kontext einer anderen Anwendung bzw. eines anderen Gerätes eine gegensätzliche Bedeutung besitzen. Noble et al. führen als Beispiel den Fall einer Datenbank an, die nur eine bestimmte Anzahl von Malen pro Tag abgefragt werden kann. Der Parameter, welcher die Anzahl der bereits gestellten Anfragen verwaltet, ist typspezifisch und nur im Kontext dieser Anwendung gültig.

Die im Folgenden angeführten Aspekte stellen weitere in der Literatur diskutierte Umgebungsparameter dar, welche von einer Integrationsinfrastruktur beachtet werden können:

Context–Awareness : Gerade im Zusammenhang mit ubiquitären Systemen stellt der *logische* räumliche Kontext einen wichtigen Umgebungsparameter dar. Nicht die Position des Geräts selbst steht im Vordergrund, sondern die Personen oder anderen Geräte, die sich in räumlicher Nähe zum mobilen Gerät befinden bzw. die logische Wertigkeit des Ortes, d.h. der Name des Ortes, die Funktion des Ortes, die Nähe zu Räumen oder Orten mit bestimmter Funktion usw. Ursprünglich geprägt wurde der Begriff der Context Awareness im Rahmen des Xerox ParcTab Experiments [WSA+95], in welchem Computer verschiedener Größe zur Untersuchung einer Umgebung für ubiquitäre Computer genutzt wurden. Die genannten Definitionen wurden vornehmlich im Rahmen dieses Projektes erarbeitet (vgl. z.B. [WSA+96], S. 47). Kidd et al. beschreiben in [KAA+99] (siehe auch [AE97]) ihren Ansatz für ein *Aware Home* (dt. „Bewusstes Haus“), in welchem Sensoren dafür genutzt werden, entsprechende Informationen aufzunehmen und zur Steuerung der Hauselektronik zu verwenden. Pham et al. beschreiben in [PSGP99] die Nutzung von Kontextinformation zur Aggregation von in einem Raum vorhandenen Computersystemen zur Unterstützung des mobilen Geräts.

Location–Awareness : Von sehr hohem Wert für die Integrationsumgebung ist das Wissen über den Ort, an welchem sich das mobile Gerät befindet. Informationen über die Lokation können dazu genutzt werden, ortsbezogene Daten auf dem Gerät darzustellen, die in einer zentralen Datenbasis gespeichert sein können [BHJJ97, Maa97]. Location–Awareness betont die absolute, *räumliche* Position der betrachteten Computersysteme bzw. die Position der Benutzer dieser Systeme. Diese wird oftmals mittels satellitengestützter Peilung (GPS) ermittelt und dann zur Datenabfrage genutzt. In der Literatur ist in diesem Zusammenhang auch die Einschränkung bzw.

Filterung von Informationen durch Einführung einer *Sichtbarkeitsgrenze* bekannt. Hohl et al. berichten in [HKL+99] von der Infrastruktur *Nexus*, welche die Nutzung von Datenbanken zur Speicherung ortsabhängiger Informationen mit der räumlichen Partitionierung in so genannten *Augmented Areas* kombiniert, und so die Sichtbarkeit von Informationen, die mittels eines mobilen Geräts an einem Ort aufgenommen werden können, auf ein bestimmtes Gebiet beschränkt.

Situation–Awareness : Eine weitere spezielle Form der Context Awareness stellt die so genannte *Situation Awareness* dar [SSR+99]. Im Falle der Situation Awareness wird primär die *relative* Position eines Gerätes zu einem anderen Gerät betrachtet, wobei im Gegensatz zur Context Awareness oft ein vom Benutzer definiertes Profil zum Einsatz kommt. Dahlberg berichtet von einem System, bei welchem der Anwender Aufgaben definieren kann, die anderen Benutzern zugeordnet werden. Meldet der in einem PDA integrierte Sensor die Nähe eines einer Aufgabe zugeordneten Benutzers, so wird der Anwender darauf akustisch und durch Darstellung der Aufgabe auf dem PDA hingewiesen (vgl. [BDL99, DLS99, RDLH99]).

Network–Awareness : Insbesondere für Integrationsinfrastrukturen ist die Qualität der Kommunikationsverbindung von hoher Relevanz. Lässt im Rahmen einer Interaktion die Qualität der Verbindung des Mobilgerätes zur Infrastruktur nach, so muss hierauf ggf. in geeigneter Weise reagiert werden. *Network–Awareness* stellt einen der grundlegenden Parameter dar, die zur Realisierung einer robusten Interaktionsplattform beachtet werden müssen. Eine mögliche Reaktion auf die verminderte Kommunikationsgeschwindigkeit könnte in der Herabsetzung der Dienstgüte liegen (*Quality of Service*). Eine andere Reaktion könnte in der Anpassung der Infrastruktur selbst liegen. Ranganathan berichtet von der Infrastruktur *Sumatra*, in der einzelne Komponenten die Qualität der Netzwerkverbindung untereinander überwachen und gegebenenfalls mittels Migration die Kommunikationsstrecken optimieren können [RASS97].

Energy–Awareness : Die Energiequelle stellt eine sehr wertvolle Ressource für mobile Geräte dar. Ein ökonomischer Umgang mit der vorhandenen Energie ist für den erfolgreichen Einsatz im mobilen Kontext essentiell. Es hat sich gezeigt, dass die Auswahl des für die Lösung eines Problems einzusetzenden Algorithmus unter Umständen eine Einsparung bis zu dreißig Prozent der Energie bedeuten kann [FS99]. Entsprechend kann die vorhandene Energie von der Integrationsinfrastruktur als Parameter einbezogen und entsprechend gehandelt werden. Ähnliche Ansätze werden auch von Weiser beschrieben [WWDS96], wo durch den Einsatz geeigneter Scheduling–Algorithmen zu einer Reduktion des Energieverbrauchs beigetragen wurde.

Die genannten Parameter stellen nur einen Ausschnitt dessen dar, was ein mobiles Gerät beachten *kann*, um sich optimal in den Anwendungskontext einzufügen bzw. einen möglichst hohen Nutzen trotz der eigenen Beschränktheit in seiner Umgebung zu erzielen. Eine wichtige Frage ist es nun, welche Parameter betrachtet werden müssen und vor allem, ob dieses auf Ebene des Benutzers, auf

der Anwendungsebene, durch eine in das Betriebssystem integrierte Funktionalität oder auf der Ebene der Hardware geschehen muss. Weiterhin gilt es zu klären, was auf der Basis der betrachteten Parameter geschehen soll, wie also eine geeignete Reaktion auf die Parameter auszusehen hat.

Die Anpassung des Computersystems auf sich ändernde Umgebungsparameter wird zumeist als *Adaption* bezeichnet (vgl. [dLWZ00, SNKP94, NPS95, Nak96, AS00] usw.). Steigt ein Parameter über einen bestimmten Schwellwert oder sinkt er unter diesen, so wird das dem Mobilgerät in geeigneter Form signalisiert. Das Gerät kann nun anhand einer zuvor festgelegten Strategie handeln und sein Verhalten gemäß der geänderten Situation *anpassen*[1]. Grundsätzlich sind zwei gegensätzliche Strategien zur Anpassung zu unterscheiden, wobei sich Integrationsansätze in einem Spannungsfeld zwischen diesen Polen einordnen lassen (siehe Abb. 3.1:

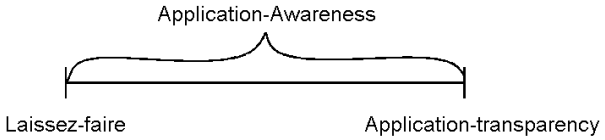


Abbildung 3.1: Application–Awareness im Spannungsfeld zwischen *Laissez–faire* und *Application–Transparency* (vgl. [SNKP94])

- Die Anwendung benennt die Umgebungsparameter, deren Werte für die Erfüllung des Anwendungsziels kritisch sind. Sie tut dieses beispielsweise durch Registrierung bei einer Dienstinstanz innerhalb des zugrunde liegenden Betriebssystems. Ändert sich ein registrierter Parameter, so wird die Anwendung über den geänderten Parameter bzw. über dessen neuen Wert informiert. Sie kann sodann eine nur in ihrem Kontext definierte Strategie einschlagen, um auf den veränderten Zustand zu reagieren. Umgebungsparameter werden also auf Anwendungsebene *bewusst* wahrnehmbar, die Anpassungsstrategie kann als *application–aware*, in ihrem Extrem auch als *laissez–faire* Strategie bezeichnet werden [SNKP94, JHE99a]. Welling nennt als Beispiel für dieses Vorgehen eine Anwendung, die von der aufwendigen Darstellung ihrer graphischen Benutzungsschnittstelle zu einer einfachen Textschnittstelle übergeht, wenn die Batterieleistung unter einen bestimmten Wert sinkt [WB97].

- Die Anwendung wird nicht über die sich ändernden Parameter informiert. Gegebenenfalls notwendige Anpassungen an die sich ändernden Umgebungsbedingungen werden innerhalb einer der Anwendung zugrunde liegenden Infrastruktur behandelt, während die Anwendung nur *indirekt* über deren Auswirkungen auf die Umgebungscharakteristik von dem geänderten Parameter erfährt, auf diese jedoch nicht selbst *aktiv* reagiert. Da die Anwendung von Änderungen der Umgebung und eventuell somit notwendigen Anpassungen kein Wissen erhält, wird diese Art der Adaption als *application–transparent* bezeichnet [SNKP94, JHE99a]. Beispielhaft

[1] *to adapt*: anpassen, angleichen an

> für dieses Form wäre eine Anwendung, die ihre Ressourcen in einem Dateisystem organisiert, das mittels Replikation und Caching auf dem Mobilgerät aktualisiert wird, sofern die verwendete Infrastruktur eine Netzwerkverbindung signalisiert. Änderungen der Daten werden durch die Anwendung an der lokalen Kopie durchgeführt und es obliegt der Infrastruktur, gegebenenfalls hierbei entstehende Inkonsistenzen zwischen dem entfernten Original und der Kopie aufzulösen [Rat98].

Sowohl für die *anwendungstransparente* als auch für die *anwendungsbewusste* Adaption wird in der Literatur argumentiert. Ist es auf der einen Seite von Vorteil, wenn die Anwendung nicht gezwungen ist, Strategien zu formulieren und zu implementieren, mit denen sie auf die sich ändernden Umgebungsparameter reagieren will, so ist auf der anderen Seite eine *optimale* Reaktion nur dann zu gewährleisten, wenn die reagierende Instanz über die Konsequenzen ihrer Handlung auf das Anwendungsziel im klaren ist. Dieses ist jedoch nur durch die Anwendung selbst zu gewährleisten. Kein generischer Ansatz wird das notwendige Kontextwissen besitzen, an Stelle der Anwendung und ohne deren Wissen zu handeln.

Grundlegende Argumente zu Gunsten einer anwendungsbewussten Adaption sind aus [SRC84] zu erschliessen. In dieser Publikation diskutieren Saltzer et al. die Vorteile einer anwendungsnahen Behandlung von auftretenden Ereignissen im Rahmen einer Netzwerkkommunikation wie z.B. die Behandlung von Fehlerfällen, die Verschlüsselung von Daten usw. Die Autoren vertreten die Meinung, dass eine Behandlung dieser Aspekte durch niedere Schichten einer Infrastruktur nur in Spezialfällen von Vorteil sei, da sich für ihre Behandlung oftmals keine *allgemeingültigen* Strategien formulieren ließen, welche für die Menge auf der Basis der Infrastruktur ablaufender Anwendungen gültig wären, bzw. genügende Effizienz aufweisen würden. Darüber hinaus seien Behandlungsstrategien in niederen Schichten zum Teil sogar kontraproduktiv, da sie auf Ebene der Anwendungen und sodann mit genügendem Kontextwissen ohnehin wiederholt werden müssten. An Stelle einer Behandlung durch die Infrastruktur sei eine Behandlung auf Ebene der Anwendungen vorzuziehen (sog. *End–To–End Caretaking*), wobei die Einflussnahme durch die zugrunde liegende Infrastruktur sich auf ein Minimum zu beschränken habe, entsprechend dem *end–to–end Prinzip*, wonach das Folgende gilt: „*. . . a function or service should be carried out within a network layer only if it is needed by all clients of that layer, and it can be completely implemented in that layer*“ [SRC98].

Die Ausführungen von Saltzer et al. lassen sich auf die Behandlung von Umgebungsparametern in Integrationsinfrastrukturen für mobile Systeme anwenden. In der überwiegenden Zahl der Fälle wird eine Reaktion auf sich ändernde Parameter, wie z.B. die Kommunikationsgeschwindigkeit, auf einer niederen Ebene nicht allen Anwendungen gerecht werden. Besitzen einzelne Anwendungen darüber hinaus eigene Strategien zur Behandlung von Parametern, so kann es auch hier dazu kommen, dass die eingeleiteten Maßnahmen sich gegenseitig beeinflussen oder gar behindern. Stellt die Infrastruktur hingegen nur die Mechanismen zur Verfügung, Umgebungsparameter aufzunehmen oder über deren Änderung informiert zu werden, so kann eine jede Anwendung in der für sie optimalen Weise reagieren.

Während anwendungsbewusste Behandlung von Umgebungsparametern für die

Mehrzahl der Fälle von Vorteil ist, bieten auch anwendungstransparente Ansätze zum Teil einen Nutzen. Sie eignen sich insbesondere für die Integration von Anwendungen, die sich der Berücksichtigung von Umgebungsparametern entziehen. Dieses sind vor allem *geschlossene* Anwendungen, zu deren Quellen kein Zugriff möglich ist und die nicht unter Einbeziehung der mobilen Umgebung entwickelt wurden. Im Falle solcher Anwendungen ist die unmittelbare Behandlung von Änderungen ihrer Umgebung nicht möglich und muss vielmehr durch eine unterstützende Infrastruktur übernommen werden, welche Anpassungen in einer für die Anwendung *transparenten* Art durchführt.

Neben einer besseren Unterstützung geschlossener Anwendungssysteme weist die anwendungstransparente Adaption des Weiteren den Vorteil auf, dass sie die unter Umständen sehr spezifischen Parameter einer bestimmten Computerumgebung von der Anwendungsentwicklung entkoppelt. So besteht die Gefahr, dass Anwendungen, die auf die Berücksichtigung *bestimmter* auf einem Gerät zur Verfügung stehender Parameter hin optimiert wurden, nur unter großem Aufwand auf eine andere Plattform übertragbar sind. Werden notwendige Anpassungen jedoch in einer zugrunde liegenden Infrastruktur behandelt und so von der Anwendung fern gehalten, so besteht dieses Problem nicht. Die Anwendung kann auch auf anderen Plattformen ausgeführt werden, die unter Umständen gänzlich andere Umgebungsparameter beachten können oder über abweichende Methoden verfügen, die Parameter auszuwerten.

3.2.2 Agilität und originalgetreue Wiedergabe

Zwei Aspekte sind im Rahmen von Integrationsbemühungen von besonderem Interesse. Wie schnell ist die Infrastruktur bzw. Anwendung in der Lage, auf die sich ändernden Umgebungsparameter zu reagieren? Und: Wie weit kann die tatsächlich erreichte Dienstqualität dem theoretischen Maximum angenähert werden? Es ist nachvollziehbar, dass beide Aspekte miteinander in Beziehung stehen und dass die Qualität einer Integrationsinfrastruktur von beiden gleichermaßen abhängt.

Die Reaktionsgeschwindigkeit der Infrastruktur wird in der Literatur als *Agilität* bezeichnet [NSN+97, Nob00]. Agilität beschreibt die Zeit, welche zwischen der Änderung eines Umgebungsparameters und der Meldung über diese Änderung an die in der Infrastruktur organisierten Anwendungen verstreicht. Während die Zeitspanne im theoretischen Fall Null sein kann, wird in realen Softwaresystemen dieser Idealwert nicht erreicht. Sowohl die Aufnahme des Parameterwertes als auch die Auswertung, ob der sich ändernde Wert eine Schwelle überschritten hat, nehmen Zeit in Anspruch. Ebenso geht die Signalisierung an die registrierten Anwendungen nicht verlustfrei vonstatten. Auch hier verstreicht auf dem Signalisierungsweg gewisse Zeit, bevor das Signal die Anwendung erreicht. Je nach Güte, d.h. Effizienz, der Infrastruktur kann also ein signifikantes Zeitquantum zwischen Änderung eines Parameters und der Meldung hierüber an die Anwendung vergehen.

Agilität stellt ein Gütekriterium von grundlegender Relevanz für die Bewertung adaptiver Integrationsinfrastrukturen dar. Noble et al. hierzu in [NSN+97]: „[...] *a key property of an adaptive system is the speed and accuracy with which it detects and responds to changes in resource availability.* “ Nur die verzögerungsar-

me Verarbeitung sich ändernder Umgebungseinflüsse ermöglicht es Anwendungen, in geeigneter Form hierauf zu reagieren und so die Dienstqualität auch dann noch zu gewährleisten, wenn die Änderungsfrequenz ähnlich hoch ist, wie dieses zum Beispiel im Falle der Netzwerkbandbreite in mobilen Szenarien zu erwarten ist.

Wird beispielsweise in einem solchen Szenario die durch den Übergang auf ein weniger leistungsfähiges Übertragungsmedium zurückgegangene Datenrate nicht umgehend an eine Anwendung zur Sprachübertragung weitergeleitet, so kann die fehlende Anpassung auf Seiten der Anwendung zu einem vollständigen Abbruch der Übertragung führen. Kann die Anwendung hingegen genügend schnell[2] informiert werden, so kann sie die Sprachqualität gegebenenfalls anpassen, bevor ihr Puffer bereits übertragener Daten sich vollständig geleert hat.

Die vertretbare Latenz der Infrastruktur kann dabei von Anwendung zu Anwendung variieren. Während im Falle von Sprachübertragung bereits geringe Verzögerungen zu inakzeptablen Auswirkungen auf die Dienstgüte führen können, sind diese im Falle einer Videoübertragung bereits deutlich geringer. Für ein mobiles Informationssystem schließlich ist die Latenz kaum noch erheblich. Für dieses ist es nurmehr notwendig, *überhaupt* benachrichtigt zu werden, wenn die Netzwerkverbindung abreißt, ohne dass hierbei jedoch besondere enge Zeitlimits erhoben werden müssten.

Insgesamt stellt die Agilität also das Maß dafür dar, für Umgebungen welcher Charakteristik *adaptive* Integrationsinfrastrukturen sich eignen. Umgebungen mit hoher Dynamik lassen sich nur mit Infrastrukturen begegnen, die einen hohen Grad an Agilität aufweisen, während weniger veränderliche Umgebungen einen geringeren Grad an Agilität voraussetzen. Noble et al. fassen diese Tatsache wie folgt zusammen: „*Agility is thus the property of a mobile system that determines the most turbulent environment in which it can function acceptably.*“

Wie in den vorigen Abschnitten beschrieben, wird die Dienstgüte direkt von der Agilität einer Infrastruktur beeinflusst. Ändert sich ein von der Infrastruktur überwachter Umgebungsparameter und über– oder unterschreitet dieser einen zuvor definierten Schwellwert, so wird dieser Umstand an registrierte Anwendungen gemeldet, die hierauf in einer für sie geeigneten Form reagieren können.

Grundsätzlich hängt die geeignete Reaktion von zwei Punkten ab:

Art des Umgebungsparameters : Wird die Netzwerkbandbreite betrachtet und fällt diese unter einen definierten Schwellwert, so muss die Anwendung entsprechend durch eine Änderung ihres Verhaltens darauf reagieren, die zu einer Reduktion der verbrauchten Bandbreite führt, d.h. zum Beispiel eine geringere Datenrate zur Folge hat. Wurde hingegen ein Unterschreiten des Batteriestandes unter einen bestimmten Schwellwert gemeldet, so muss dafür gesorgt werden, dass der Energieverbrauch gesenkt wird, beispielsweise durch Abschaltung nicht notwendiger Systemkomponenten.

Art der Anwendung : Jede Anwendung besitzt ein spezifisches Repertoire an Reaktionsvarianten, die sie gemäß ihres Anwendungsziels und ihres

[2] Es ist Sache des Anwendungsentwicklers, die konkrete Ausprägung von genügender Geschwindigkeit zu definieren. Diese wird sich im Allgemeinen primär durch den Anwendungskontext ergeben.

Anwendungskontexts auf gemeldete Parameteränderungen hin anwenden kann. Die reduzierte Datenrate kann bei einer Videoanwendung dazu führen, dass bei gleichbleibender Bildqualität nurmehr jedes zweite Bild übertragen wird, während eine Audioanwendung die Übertragungsqualität von Stereo auf Mono reduzieren kann.

Der Typ des Parameters und der Typ der Anwendung stellen also die primären Kriterien für die Art der Reaktion dar. Dabei legt ersterer die *Klasse* von Reaktionen fest, die möglich sind. Letzterer bestimmt die konkrete *Ausprägung* der Reaktion, die von einem Anwendungsprogramm gezeigt werden wird (siehe Tabelle 3.1).

Parametertyp	**Anwendungstyp**	**Klasse**	**Reaktion**
Übertragungsrate	Videospieler	Datenreduktion	Auslassung von Einzelbildern
Übertragungsrate	Audiospieler	Datenreduktion	Verminderung der Tonqualität
Übertragungsrate	Börsendienst	Datenreduktion	Reduktion der Aktualisierungsfrequenz

Tabelle 3.1: Beispiel für verschiedene Reaktionsklassen

Die im Festnetz gespeicherte Version einer Datei bzw. der nahezu verzögerungsfreie Zugriff hierauf aus dem Festnetz wird üblicherweise als *Referenz* für die erreichbare Dienstgüte angesehen. In mobilen Umgebungen wird ein Anwendungssystem nur in Ausnahmefällen die Qualität der Referenzkopie erreichen und stattdessen gemäß den zuvor gemachten Reaktionsklassen eine Minderung der Dienstgüte herbeiführen. In der Literatur wird der Grad, bis zu welchem die Qualität der Referenzkopie erreicht wurde, als *Fidelity* (dt. Wiedergabegüte) bezeichnet [NPS95, Nob00]. Je näher die Dienstqualität auf dem Mobilgerät dem Original kommt, desto höher ist die derzeitige *Fidelity*.

Wie angedeutet hängt die Bewertung der Wiedergabegüte vornehmlich vom Anwendungskontext ab. Jede Anwendung kann ihre eigenen Dimensionen für die Bewertung der Wiedergabegüte besitzen, und ihr Bewertungsmodell stellt gewissermaßen ihr Alleinstellungsmerkmal dar. Dabei gibt es grundsätzlich keine Beschränkungen in der Menge möglicher Kriterien. Gemäß Noble et al. existiert allerdings die *Konsistenz* als ein *allen* Anwendungen gemeinsames Bewertungskriterium.

Die Güte einer Integrationsstrategie lässt sich jederzeit daran ermessen, wie konsistent die Dienstdarstellung auf dem Mobilgerät im Vergleich zum Original ist und welchen Einfluss die veränderlichen Umgebungsparameter auf die Konsistenz der Kopie haben. Verteilte Dateisysteme wie CODA [Bra98], Ficus [PGJH90] oder Bayou [DPS+94] können zwar mittels lokal verwalteter Kopien die Datennutzung auch dann noch ermöglichen, wenn eine Unterbrechung der Kommunikationsverbindung signalisiert wurde. Mit zunehmender Dauer der Unterbrechung ist die lokale Replik jedoch mit immer größerer Wahrscheinlichkeit vom entfernt gespeicherten Original verschieden. Die Wiedergabegüte stellt

dabei das Maß dar, wieweit die Kopie sich vom Original unterscheidet (vgl. [NSN+97], Kap. 2.2).

Die Wiedergabegüte hängt direkt von der Agilität einer Infrastruktur ab. Kann die Änderung von Umgebungsparametern ohne größere Verzögerungen an die Anwendung gemeldet werden, so kann diese umgehend mittels geeigneter Reaktionen dafür sorgen, dass den geänderten Bedingungen Rechnung getragen und somit die optimale Nutzung der zur Verfügung stehenden Ressourcen erreicht wird. Nur die optimale Nutzung der zur Verfügung stehenden Ressourcen wiederum ermöglicht eine verlustarme Datenrepräsentation auf dem Mobilgerät und ermöglicht es somit, eine optimale Wiedergabegüte zu erreichen.

Die gemeinsame Betrachtung von Agilität und Wiedergabegüte erlaubt die Bewertung und Klassifikation sowohl von applikationsbewussten als auch von vollständig anwendungstransparenten Integrationsinfrastrukturen für mobile Geräte. Anhand dieser beiden Maßeinheiten lässt sich ermessen, wie gut es der Umgebung gelingt, Änderungen der Umgebungsparameter aufzunehmen und diese an die Anwendungen weiterzuleiten, die in die Integrationsinfrastrukur integriert sind (Agilität). Des Weiteren lässt sich ermessen, wie sehr die Anwendungen in der Lage sind, die empfangenen Signale über Parameteränderungen umzusetzen und anhand geeigneter Anpassungen auf eine Weise zu reagieren, die zu möglichst geringen Beeinträchtigungen der Anwendungsqualität führt (Wiedergabegüte). Gelingt es in einer optimalen Abstimmung zwischen Anwendung und Integrationsumgebung beide Aspekte zu maximieren, so ist eine für den Anwendungsfall ideale Integration die Folge. Je weniger es jedoch gelingt eine diesbezügliche Abstimmung zu finden, desto geringer ist die Qualität der Integration.

Im Falle anwendungstransparenter Integrationsinfrastrukturen muss die Begrifflichkeit insofern variiert werden, als für die Dienstqualität nicht länger der Dialog zwischen Anwendung und Infrastruktur relevant ist. Die Agilität ist entsprechend nicht mehr als bestimmendes Qualitätsmerkmal zu sehen. Da die Anwendung über die sich ändernden Umgebungsparameter nicht informiert wird und sie somit nicht selbständig auf diese reagieren kann, hat die Reaktionsgeschwindigkeit der Infrastruktur keinen direkten Einfluss mehr auf ihr Handeln. Auch die Wiedergabegüte wird nicht mehr durch die Anwendung selbst beeinflusst, da auch in diesem Falle die sich ändernden Umgebungsparameter nicht genutzt werden. Eine Änderung der Dienstqualität ergibt sich demgemäß nurmehr aus den Anpassungen durch die Infrastruktur. Entsprechend ist für anwendungstransparente Integrationsansätze keine Dienstgüteoptimierung durch eine optimale Abstimmung zwischen Anwendung und Infrastruktur zu erlangen.

3.3 Integrationsansätze im Überblick: eine OSI–Sicht

Im Folgenden soll nun also anhand der Schichten des OSI–Modells ein Überblick zu verschiedenen Integrationsansätzen gegeben werden. Dabei werden jeweils ein oder mehrere Ansätze als Stellvertreter der jeweiligen OSI–Schicht umrissen, wobei auf spezielle Charakteristika des ausgewählten Ansatzes eingegangen wird. Der Abschnitt beschränkt sich in seiner Betrachtung vornehmlich

auf die mittleren Schichten des Modells. Hardwarenahe Integration steht nicht im Mittelpunkt dieser Arbeit, während auf Ansätze der Präsentations– sowie Anwendungsschicht eingehend im folgenden Kapitel eingegangen wird.

3.3.1 Bitübertragungsschicht

Die Bitübertragungsschicht stellt die niedrigste Ebene dar, auf der Integration mobiler Geräte betrieben wird. Alle höheren Integrationsansätze basieren letztlich auf der drahtlosen Übertragung von Daten und somit auf Festlegungen hinsichtlich Frequenzen, Kodierungsverfahren et cetera, wie sie auf dieser Ebene laut OSI definiert werden.

Da in dieser Arbeit nur *softwareseitige* Integrationsansätze betrachtet werden, sollen die physikalischen Grundlagen daher hier nicht weiter vertieft werden. Es sei lediglich auf [BB97] verwiesen, wo über die Integration mobiler Geräte in die Andrew Architektur der Carnegie Mellon University berichtet wird. Im beschriebenen Projekt wurde eine verteilte Systeminfrastruktur entworfen, in welcher so genannte „Webpads“ als mobile Sichtgeräte den Zugriff auf die Datenbestände innerhalb eines zentralen Informationssystems ermöglichen. In der zitierten Abhandlung wird näher auf Details des Projektes und die Verfahren auf Ebene der Bitübertragung eingegangen, die bei der Integration der Webpads zum Einsatz kommen.

3.3.2 Sicherungsschicht

Die Sicherungsschicht stellt die niedrigste Ebene zur softwareseitigen Integration mobiler Geräte in eine verteilte Umgebung dar. Integration auf dieser Ebene drückt sich primär in der Anwendung optimierter Sendeverfahren (*Automatic Repeat Request, ARQ*) sowie Fehlerkorrekturverfahren (*Forward Error Correction, FEC*) mit besonderer Eignung für mobile Übertragungskanäle aus. Primäres Ziel ist es mithin, die Übertragung ansonsten uninterpretierter Daten in ihrer Effektivität zu steigern und robuster gegen die Fehler des drahtlosen Übertragungsmediums zu machen und so für höhere Schichten verlässlich verfügbar zu machen.

Im Gegensatz zu Verfahren auf höheren Schichten werden dem Einsetzen auf dieser Ebene folgende Vorteile zugeschrieben (vgl. [XP99]):

- Verzicht auf komplexe Softwareagenten, wie sie von Ansätzen höherer Schichten (insbesondere Transport– sowie Sitzungsschicht) benötigt werden.
- Keine Festlegung auf bestimmte Übertragungsprotokolle wie UDP oder TCP, sondern Flexibilität zum Einsatz mit beliebigen höherschichtigen Protokollen.
- Die Möglichkeit zur Ausnutzung spezieller Eigenschaften des physikalischen Übertragungsmediums bzw. der Sicherungsschicht.

- Die Möglichkeit zur Isolation eines bestimmten Verfahrens auf einen Verbindungskanal zwischen Mobilgerät und Basisstation und somit die Möglichkeit zur Kombination verschiedener Verfahren in einem einzigen Gesamtszenario.

Eine Vielzahl von Verfahren setzt auf dieser Ebene an, wobei vornehmlich die Optimierung von TCP/IP Datenübertragungen betrachtet wird. Stellvertretend sollen an dieser Stelle zwei Ansätze skizziert werden, nämlich das von Ayanoglu et al. vorgeschlagene AIRMAIL [APL+95] sowie *Multi Service Link* von Xylomenos und Polyzos [XP99].

3.3.2.1 AIRMAIL

Der *Asymmetric Reliable Mobile Access in Link Layer*, kurz AIRMAIL, beruht im Kern auf zwei Optimierungsstrategien. Zum einen wird in der Kommunikationsbeziehung zwischen Mobilgerät und Basisstation die Last soweit als möglich auf die Basisstation als leistungsfähigeren Kommunikationspartner übertragen. Darüber hinaus wird die Effizienz des Übertragungskanals mittels besonderer Kodierungstechniken gegenüber herkömmlichen Verfahren gesteigert.

Zum Einsatz kommt hierbei zunächst die selektive Neusendung verlorener Datenpakete mittels *Automatic Repeat Request* (ARQ). ARQ ist ein Verfahren, bei dem Datenpakete bei Empfang durch die Empfangsstation aktiv quittiert werden. In seiner Grundform wird zu jedem eingehenden Paket eine Empfangsbestätigung an den Sender zurück übermittelt. Anhand dieser Empfangsbestätigung kann der Sender die erfolgreiche Übertragung des Pakets feststellen und mit dem nächsten Paket fortfahren. Erhält er zu einem Paket keine Empfangsbestätigung, so kann er dieses als Scheitern der Übertragung werten und das betroffene Paket erneut senden.

Im Sinne der asymmetrischen Aufteilung von Verantwortlichkeiten wird dieses Verfahren in AIRMAIL insofern variiert, als die Basisstation hinsichtlich der Etablierung des Steuerungsprotokolls stärker in die Pflicht genommen wird. Die Aktivität des Mobilgeräts wird hingegen auf ein Minimum beschränkt. Wie bei ARQ üblich werden dazu empfangene Datenpakete regelmäßig durch die Basisstation quittiert und der Empfang an das Mobilgerät gemeldet. Das Mobilgerät ist also jederzeit darüber informiert, ob die ausgesendeten Pakete erfolgreich empfangen wurden und welche Pakete im Störungsfall erneut gesendet werden müssen. Im Gegensatz zu herkömmlichem ARQ wird hingegen auf die Verwaltung von Zeitnehmern im Mobilgerät verzichtet. Fehlerfälle kann es somit nicht selbständig durch ausbleibende Empfangsbestätigungen erkennen, was nach den Untersuchungen von Ayanoglu et al. jedoch keinen wesentlichen Einfluss auf die Zuverlässigkeit des Protokolls hat. In entgegengesetzter Richtung, d.h. bei Empfang von Datenpaketen durch das Mobilgerät, wird selbiges in Abwandlung zum ARQ von der Verpflichtung zur sofortigen Bestätigung von empfangenen Paketen befreit. Lediglich der Erhalt einer ganzen Gruppe von Paketen muss quittiert werden.

Zusätzlich verwaltet die Basisstation eine Reihe von Zeitnehmern, die an die Stelle fortlaufender Quittungen treten. Nachdem die Basisstation eine Gruppe von Datenpaketen ausgesandt hat, wird ein Zähler gestartet. Trifft die Quittung

zu dieser Gruppe nicht vor Ablauf des Zählers ein, so wird die Übertragung als gescheitert angesehen und muss wiederholt werden. Trifft die Quittung hingegen rechtzeitig ein, so war die Sendung offenbar erfolgreich, der Zähler kann verworfen werden.

Durch die Modifikation des verwendeten ARQ Schemas kann die im Zusammenhang mit dem Datenaustausch zwischen Mobilgerät und Basisstation auftretende Belastung deutlich in Richtung der Basisstation verschoben werden. Die Anzahl der vom Mobilgerät zu übertragenden Datenpakete wird durch Verzicht auf die Mehrzahl der Empfangsquittungen deutlich reduziert. Da außerdem die Basisstation ausschließlich für die Verwaltung von Zeitnehmern verantwortlich ist, kann auch der Berechnungsaufwand gegenüber herkömmlichen ARQ Verfahren deutlich reduziert werden.

Zur weiteren Effizienzsteigerung wird bei AIRMAIL die Kombination mehrerer Varianten der *Forward Error Correction* (Bit–, Byte– sowie Packet–Level–FEC) angewendet. FEC basiert auf der redundanten Erweiterung zu übertragender Daten mittels Steuerungsdaten. Auf Bit–Ebene sind dieses Steuerbits, auf Byte–Ebene Prüfsummen (CRCs) und auf Paket–Ebene regelmäßig eingestreute Kontrollpakete. Auf der Empfängerseite können die Steuerungsinformationen dazu verwendet werden, die Vollständigkeit der erhaltenen Daten zu überprüfen. Werden dabei Übertragungsfehler festgestellt, so lassen diese sich bis zu einem bestimmten Grad anhand der erhaltenen Steuerinformationen ausgleichen.

In AIRMAIL werden alle drei Verfahren eingesetzt, wobei jedoch kein automatisches Verfahren zur Erkennung des optimalen Kodierungsverfahrens existiert, sondern dieses explizit einzustellen ist. In [APL+95] wird für schmalbandigen Außeneinsatz (*narrowband outdoor*) die Verwendung von Bit– bzw. Byte–Level–FEC vorgeschlagen, für den breitbandigen Inneneinsatz (*broadband outdoor*) hingegen Packet–Level–FEC.

Zusammengenommen ergibt sich durch die Kombination von ARQ– und FEC–Verfahren in AIRMAIL bei einer asymmetrischen Aufgabenverschiebung in Richtung der Basisstation ein deutlicher Vorteil gegenüber herkömmlichen Verfahren. Die Effizienz des Übertragungskanals wird ebenso gesteigert wie dessen Robustheit. Durch selektive Neusendung sowie Rekonstruierbarkeit übertragener Datenpakete kann außerdem die Auslastung der Funkstrecke signifikant reduziert werden. Dabei wird die Berechnungs– und Kommunikationslast weitgehend vom Mobilgerät genommen, so dass dieses bezüglich seiner Ressourcen geschont werden kann.

3.3.2.2 Multi Service Link

Ebenso wie AIRMAIL basiert auch das Multi Service Link auf einer Kombination verschiedener Kodierungsverfahren zur effizienteren Ausnutzung der Kommunikationsstrecke zwischen Mobilgerät und Basisstation. Motivation für den Ansatz ist die Beobachtung, dass verschiedene Nutzlasten höherer Übertragungsschichten ebenso zu unterschiedlichen Reaktionen auf Fehler der Datenübertragung führen wie Variationen der zugrunde liegenden Sende-/Empfangsstationen. In einem Beispiel zeigen Xylomenos und Polyzos, dass Übertragungen im 2,4 GHz WaveLAN zwischen einer Sende-/Empfangseinheit auf PCMCIA Basis und einer entsprechenden Gegenstelle auf einer ISA Karte je nach eingenommener Rolle

als Sender oder Empfänger zu signifikant unterschiedlichen Übertragungsraten führen. Ähnliches gilt auch für die Nutzlast: Ströme von TCP Paketen reagieren in anderer Weise auf Übertragungsfehler als UDP Ströme; die Auswirkungen auf Videoübertragungen unterscheiden sich deutlich von denen auf Dateiübertragungen mittels FTP [XP99].

Diverse Optimierungsansätze auf Ebene der Sicherungsschicht sind nun in der Lage, die Auswirkungen von Übertragungsfehlern auf einzelne Konstellationen aus physikalischem Übertragungsmedium und Nutzlast auf Netzwerkebene in unterschiedlicher Weise zu beeinflussen. In Messreihen lässt sich zeigen, dass *Selective Repeat* [Bra87] sich vorteilhaft auf FTP Übertragungen auswirkt, *Karn's RLP* [Kar93] bzw. *Forward Error Correction* (siehe oben) deutliche Vorteile bei Videoübertragungen zeigen usw. (vgl. [XP99] für eine ausführliche Betrachtung).

Als integrativer Ansatz auf eben dieser Ebene ist das Multi Service Link nun in der Lage, aus dem zugrunde liegenden Übertragungsmedium Rückschlüsse auf den optimalen Mechanismus zu ziehen und sodann diesen zur Optimierung des *lokalen* Übertragungsweges zwischen Basisstation und Mobilgerät zu verwenden. Zusätzlich bietet das Multi Service Link höherschichtigen Anwendungen die Möglichkeit zu deren Registrierung. Das ermöglicht über die Behandlung von Eigenschaften der Bitübertragungsschicht hinaus außerdem die gesonderte algorithmische Behandlung jeweiliger Nutzlasten. Dies setzt jedoch ein explizites Wissen über das Multi Service Link durch die höherschichtige Anwendung voraus und weicht somit den oben angeführten Vorteil der Isolation auf die Sicherungsschicht auf.

Insgesamt bieten Integrationsansätze auf Ebene der Sicherungsschicht einiges Potential zur Effizienzsteigerung bei Datenübertragungen über fehlerbehaftete Funkschnittstellen. Das Einsetzen auf dieser Ebene bietet gegenüber Ansätzen der Netzwerk– und Transportschicht den Vorteil der lokalen Eingrenzung auf die direkte Verbindung zwischen Basisstation und Mobilgerät sowie der Möglichkeit zur unmittelbaren Reaktion auf Fehlerfälle des Übertragungskanals. Integrationsansätze auf dieser Ebene können also punktuell auf die Anforderungen der spezifischen Verbindungsstrecke hin optimiert werden. Gleichzeitig stellen sie jedoch auch ein essentielles Fundament für Ansätze höherer Ebenen dar. Denn nur auf der Basis des durch sie geschaffenen robusten und somit verlässlichen Kommunikationskanals sind komplexe Protokolle wie z.B. das im nächsten Abschnitt beschriebene Mobile IP überhaupt realisierbar.

3.3.3 Netzwerkschicht

Das Netzwerkprotokoll IP stellt seit seiner Einführung 1980 das wichtigste Kommunikationsprotokoll des Internet dar. Die überwiegende Zahl der an das Internet angeschlossenen Computersysteme tauschen Daten auf der Basis von IP aus und auch die Verbindungsrechner zwischen einzelnen Teilnetzen nutzen IP zur Interaktion. Unter TCP/IP wird jeder Knotenpunkt durch eine eindeutige Adresse, die so genannte IP–Adresse identifiziert. IP besitzt einen Adressraum von 32 Bit und erlaubt demgemäß die direkte Adressierung von insgesamt über 4 Milliarden (2 hoch 32) Knotenpunkten. Die 32 Bit einer jeden IP–Adresse werden in Gruppen zu je acht Bits (sog. *Octets*) zusammengefasst, so dass IP–Adressen sich in der Form A.B.C.D darstellen lassen, wobei jede der Teilzahlen

einen Wertebereich zwischen 0 und 255 besitzt.

Um die topologischen Gegebenheiten miteinander verbundener Teilnetze besser abbilden zu können, wurde über den beschriebenen Adressraum die Struktur der so genannten *Subnetze* gelegt. Ein Subnetz stellt jeweils die logische Gruppierung eines zusammenhängenden Bereiches der insgesamt verfügbaren Adressen innerhalb von IP dar. Zur Vereinfachung der Vergabe und Organisation von IP–Subnetzen wurden vier Gruppen von Subnetzen definiert, die jeweils als eine *Klasse* bezeichnet werden.

Subnetze der jeweiligen Klasse können nicht willkürlich aus dem gesamten Adressraum von IP ausgewählt werden. Sie ergeben sich vielmehr als Resultat der im Folgenden beschriebenen Bitoperationen über die vier Oktets des Adressierungsschemas von IP. Die Berechnung der Größe der Netze, deren Menge und Lokation innerhalb des Adressraumes wird dabei in zwei Teilschritten festgelegt. Zunächst wird anhand einer Berechnung über das erste Oktet eine Partitionierung der mittels des ersten Oktets darstellbaren 256 Zahlen in die Klassen A, B, C sowie D durchgeführt.

Die Anzahl der Netze einer jeden Klasse ergibt sich sodann als Permutation der führenden Oktets des Adressierungsschemas, wobei im Fall der Klasse A nur das erste Oktet herangezogen wird, im Falle der Klasse B das erste und zweite Oktet und so weiter. Die Anzahl verfügbarer Adressen innerhalb eines jeden Subnetzes einer Klasse ergibt sich schließlich aus der Permutation der dabei verbleibenden Oktets. Für die Klasse A also drei Oktets, für die Klasse B zwei Oktets et cetera.

Die Partitionierung der Subnetze im ersten Oktet des Adressierungsschemas von IP erfolgt folgendermaßen:

Klasse A : Enthält das erste Bit des ersten Oktets eine Null, so wird der Rest des Oktets als Subnetz–Anteil eines Subnetzes der Klasse A gezählt. Es ergeben sich für die Klasse A entsprechend 126 Subnetze, die durch eine Zahl zwischen 1 und 126 im ersten Oktet eingeleitet und identifiziert werden.

Klasse B : Enthält das erste Bit eine Eins, das zweite Bit jedoch eine Null, so wird der Rest des Oktets gemeinsam mit dem zweiten Oktet als Subnetz–Anteil eines Subnetzes der Klasse B gezählt. Es ergeben sich somit 16,384 (2 hoch 14) verschiedene Netze der Klasse B, die durch Zahlen zwischen 127 und 192 im ersten Oktet eingeleitet werden.

Klasse C : Enthalten die ersten beiden Bits des ersten Oktets eine Eins und das dritte Bit eine Null, so wird der Rest des Oktets sowie die folgenden beiden Oktets als Subnetz–Anteil eines Subnetzes der Klasse C gewertet. Es ergeben sich für die Klasse C entsprechend 2,097.152 (2 hoch 21) verschiedene Netze der Klasse C, welche durch Zahlen zwischen 193 und 223 im ersten Oktet eingeleitet werden.

Klasse D : Die verbleibenden Adressen ab 224.0.0.0 werden nicht als zu Subnetzen im eigentlichen Sinne gehörig angesehen, sondern als Multicast–Adressen. Demgemäß gelten für sie keine Subnetz–Berechnungen.

Eines der wesentlichen Ziele bei der Entwicklung von IP bestand darin, ein Kommunikationsprotokoll zu schaffen, das sich durch besondere Robustheit gegen Störungen oder Ausfälle einzelner Netzwerkknoten auszeichnet. Datenpakete sollen auch dann noch zwischen Start– und Zielknoten ausgetauscht werden können, wenn einzelne, zuvor zur Weiterleitung von Datenpaketen verwendete Verbindungsknoten, zwischen diesen beiden Knoten ausfallen oder sich die Netzwerkstruktur zwischen den beiden Knoten unvorhergesehen ändert. Entsprechend stellt die Weiterleitung von Datenpaketen, das so genannte *Routing*, eine der wichtigsten Eigenschaften von IP dar.

Rechnerknoten besitzen in IP kein Wissen über alle Knoten, die zwischen ihnen und ihren Kommunikationspartnern liegen. Entsprechend werden in Datenpaketen lediglich Start– und Zieladresse kodiert, während keine weiteren Informationen über den Weg der Pakete durch das Netz festgelegt werden. Diese Informationen werden vielmehr zur Transportzeit als Teil der Routing–Algorithmen von IP festgestellt und genutzt.

Im Zusammenhang mit der Weiterleitung von Datenpaketen zwischen einzelnen Netzwerkknoten spielen zwei besondere Knotentypen eine besondere Rolle: *Router* sorgen für die Weiterleitung von Datenpaketen zwischen einzelnen Subnetzen und *Gateways* für die Übertragung von Datenpaketen in ein bestimmtes Subnetz, für das sie verantwortlich sind.

Router verfügen über eine Tabelle aus Paaren von Subnetz–Adressen und Gateways, an die Datenpakete weitergeleitet werden, die für eines der ihnen bekannten Subnetze bestimmt sind. Daneben kennen sie einen so genannten *Default Router*, an den sie alle Pakete weiterleiten, zu denen sie kein verantwortliches Gateway feststellen konnten. Wird einem Router ein Datenpaket zugeleitet, so isoliert er zunächst den in der Zieladresse kodierten Subnetz–Anteil. Danach durchsucht er die interne Routing–Tabelle nach einem korrespondierenden Eintrag für das erkannte Subnetz. Wurde ein für das Subnetz verantwortliches Gateway gefunden, so wird das Datenpaket an den entsprechenden Netzwerkknoten weitergeleitet. Ansonsten reicht der Router das Paket an den ihm bekannten Default Router weiter, der dann seinerseits versucht, das Paket zuzustellen.

3.3.3.1 IPv4 und Mobilität

Während Robustheit von Beginn an eines der wichtigsten Ziele von IP war, spielte die mögliche Mobilität der Netzwerkknoten keine Rolle. In IP–Netzen besitzt jeder Knotenpunkt eine Adresse, die er im Normalfall während seiner gesamten Lebenszeit beibehält. Die Netzwerkadresse identifiziert ihn eindeutig gegenüber allen übrigen Rechnerknoten im IP–Adressraum und legt in der zuvor bereits beschriebenen Weise gleichzeitig seine Zugehörigkeit zu einem bestimmten Subnetz fest. Da die Topologie des Internet über die Subnetze festgelegt ist, steht somit die Lokation eines jeden Rechners zunächst einmal fest: Eine Änderung des Subnetzes, in welchem ein Rechner positioniert ist, ist nicht möglich, ohne dass sich hierbei die IP–Adresse ändern würde und somit eindeutige Identifikation verloren ginge. Ändert ein Mobilknoten beim Übergang in ein anderes Subnetz seine Adresse gemäß des veränderten Subnetz–Anteils, so ist er für bisherige Kommunikationspartner nicht mehr adressierbar.

Wie ebenfalls beschrieben basiert die Weiterleitung von Datenpaketen zwischen Startknoten und Zielknoten auf der Nutzung von Subnetz–Informationen, anhand derer Router und Gateways den jeweils für das Paket zuständigen Folgeknoten bestimmen können. Behält ein mobiler Netzwerkknoten seine IP–Adresse beim Wechsel zwischen zwei Subnetzen bei, so können Datenpakete nicht länger an ihn weitergeleitet werden. Da in IP die Adresse untrennbar mit dem Subnetz verbunden ist, werden Pakete letztlich immer an das Subnetz weitergeleitet, das in einer Adresse kodiert ist, ungeachtet dessen, an welches Netzwerk der Zielknoten mittlerweile angeschlossen ist.

Zur Unterstützung von Mobilität in TCP/IP basierten Netzen sind entsprechend folgende Aspekte zu betrachten:

Adressierung : Wie können Rechner adressiert werden, wenn sie sich nicht in ihrem Heimat–Subnetz befinden, sondern vorübergehend durch ein anderes Subnetz mit dem Internet verbunden sind?

Routing : Wie können Datenpakete ihren Weg von einem Startknoten zu einem mittlerweile verschobenen Endknoten finden?

Konzeptionell müssen zur Behandlung der Mobilitätsproblematik auf Ebene der Netzwerkschicht folgende Punkte beachtet werden. Wird ein Mobilgerät aus seinem Heimat–Subnetz entfernt und in einem fremden Netzwerk angeschlossen, muss zunächst eine neue IP–Adresse vergeben werden, unter der das Gerät im geänderten Subnetz erreichbar ist und zu der auch aus dem Internet mittels herkömmlicher Routing–Mechanismen Datenpakete weitergeleitet werden. Bevor dieses geschehen kann, ist es für das Mobilgerät notwendig, die Infrastruktur des besuchten Netzes zu erkennen und die Rechner zu identifizieren, welche für die Vergabe von IP–Adressen zuständig sind, für das Routing von Datenpaketen usw. Dieser erste Schritt wird als *Entdeckung* (engl. *discovery*) bezeichnet.

Will ein Kommunikationspartner mit dem Mobilgerät Kontakt aufnehmen, so wird dieses in Unwissenheit der geänderten Adresse zunächst unter Verwendung der vorigen Zieladresse geschehen. Entsprechend müssen Pakete, die im Heimat–Netzwerk ankommen und dort nicht zugestellt werden können, an den neuen Aufenthaltsort des Mobilgerätes weitergeleitet werden. Damit dieses möglich ist, muss das Mobilgerät eine neue IP–Adresse erfragen und diese sodann an eine Weiterleitungsinstanz im alten Netzwerk melden, damit diese eintreffende Pakete ordnungsgemäß an den neuen Zielpunkt umleiten kann. Dieser zweite Schritt wird als *Registrierung* (engl. *registry*) bezeichnet.

Nachdem die Weiterleitungsinstanz den neuen Aufenthaltsort des Mobilgeräts kennt, kann es die eingegangenen Datenpakete an diese neue Zieladresse weiterleiten. Da das ursprünglich falsch adressierte Datenpaket dabei in ein neues, an die neue Lokation adressiertes Datenpaket integriert wird, wird dieser letzte Schritt als *Tunneln* (engl. *tunneling*) zum Mobilgerät bezeichnet. Auf dem Rückweg kann das Mobilgerät die im erhaltenen Datenpaket kodierte Absenderadresse verwenden und die Antwort so direkt zurück an den Absender schicken. Hierbei kann auf eine besondere Berücksichtigung der Mobilität verzichtet werden.

Zusammengenommen besteht die Unterstützung von Mobilität unter Verwendung von Mobile IP aus drei wesentlichen Abschnitten:

- Entdeckung
- Registrierung
- Tunnelung

Die Integration dieser drei Phasen in das bestehende IPv4 wurde in verschiedenen Ansätzen untersucht (vgl. [IDJ91], [LP97], [MS93] u.a.). Allen Varianten von Mobile IP gemeinsam ist ein abstraktes Mobilitäts–Managementmodell (vgl. [Per98]).

Wie beschrieben stellt die Zuweisung einer temporären IP–Adresse eine Notwendigkeit innerhalb des IP–Protokolls dar, ohne welche Datenpakete nicht an den tatsächlichen Aufenthaltsort eines Mobilgerätes weitergeleitet werden könnten. Auf abstrakter Ebene besitzt jeder mobile Knoten eines IP–Netzwerkes also zunächst einmal eine unveränderliche *Identifikationsadresse*, auch als *Heimatadresse* bezeichnet, sowie eine veränderliche IP–Adresse, die so genannte *Fürsorge–* oder auch *care–of–Adresse*. Erstere ist lediglich im Heimat–Subnetz des Mobilknotens gültig und ändert sich nicht auf dessen Initiative. Letztere gibt jeweils den derzeitigen Aufenthaltsort eines Mobilknotens wieder und ändert sich nach dessen Relokation.

Die tatsächliche Adresse eines Mobilknotens ergibt sich wie beschrieben immer aus dem Tupel von Heimatadresse und Care–Of–Adresse. Trifft ein Datenpaket im Heimatnetz ein, das an einen abwesenden Mobilknoten adressiert ist, so muss zunächst anhand dieses Tupels geprüft werden, ob zu dem Knoten eine Care–Of–Adresse bekannt ist, an welche das Paket weitergeleitet werden kann. Alle Erweiterungen des IP–Protokolls besitzen hierzu ein Ortsverzeichnis (*location directory*), in welchem diese Tupel gespeichert sind.

Befindet sich im Location Register für den Mobilknoten ein Eintrag, der seine neue Zieladresse enthält, so müssen die Datenpakete an diese weitergeleitet werden. Dieses bedeutet zunächst einmal, dass die Datenpakete im Heimatnetzwerk abgefangen, an die Care–Of–Adresse neu adressiert und dann an diese neue Zieladresse weitergeleitet werden müssen. Die drei Teilschritte des Abfangens, Readressierens und Weiterleitens werden abstrakt in einer *Readressierungsfunktion* zusammengefasst. Alle bekannten Ansätze für IP–Mobilität verfügen über eine solche Readressierungsfunktion.

Nach Eintreffen des Datenpakets an der Care–Of–Adresse muss das Datenpaket wieder in seinen ursprünglichen Zustand zurückversetzt und die Care–Of–Adresse wieder durch die eigentliche Zieladresse ersetzt werden. Diese Rückwandlung wird konzeptionell als *inverse Readressierung* bezeichnet. Die inverse Readressierung kann, je nach Implementation, durch eine vorgeschaltete Dienstinstanz innerhalb des Zielnetzes erfolgen, sie kann jedoch auch als Teil der Paketverarbeitung durch den Mobilknoten selbst erfolgen.

Gemäß des abstrakten Modells lässt sich IP–Mobilität mittels Mobile IP somit in die folgenden Schritte aufspalten:

- Eintragung der Care–Of–Adresse im Location Directory
- Readressierung für den Mobilknoten eintreffender Datenpakete

- Weiterleitung der Datenpakete an die Care–Of–Adresse
- Invertierte Readressierung im Zielnetzwerk
- Auslieferung der Datenpakete an den Mobilknoten

Unterschiede zwischen den prototypischen Implementationen von Mobile IP ergeben sich vor allem darin, welche Instanzen innerhalb der Infrastruktur die genannten Aufgaben übernehmen oder wie die Kommunikation im Detail stattfindet.

Die Realisierung der Columbia Universität [MS93] fasst die Vergabe von Care–Of–Adressen, das Location Directory, die Readressierung und Weiterleitung der Datenpakete sowie die inverse Readressierung der Datenpakete in so genannten *Mobile Support Routern* (MSR) zusammen. Anhand eines speziellen Kontrollprotokolls, dem *Mobile Internet Control Protocol* (MICP), werden Mobilknoten über die Adresse des jeweils verfügbaren MSR informiert oder mit einer temporären IP–Adresse versorgt. MSRs stehen untereinander im Kontakt und informieren sich gegenseitig über Relokationen unter den Mobilknoten. Sie sorgen demgemäß für die ständige Aktualisierung des Location Directory, ohne dass der Mobilknoten hiermit belastet würde.

Im Gegensatz dazu verzichtet die Implementation von Sony [TYT91] auf spezielle Dienstinstanzen und integriert die notwendigen Funktionen in ohnehin in jedem Netzwerk vorhandene Gateways bzw. Router. Neben dem Location Directory übernimmt der Sony Router die Readressierung bzw. inverse Readressierung neben seinen herkömmlichen Funktionen zur Weiterleitung von Datenpaketen in das lokale Netzwerk und zur Übertragung von Datenpaketen an entfernte Router. Sony Router analysieren alle Datenpakete, die durch sie hindurchgeleitet werden nach dem Vorhandensein spezieller IP–Optionen, mit denen ein Mobilknoten den Router seines Heimatnetzwerks über seine geänderte Adresse informiert. Findet ein Router solche Pakete, so übernimmt er sie in sein lokales Location Directory. Neben Paketen, die direkt an ihn adressiert waren, nimmt er auch solche Pakete auf, die ihn auf ihrem Weg zu einem Zielrouter passieren. Für jedes Datenpaket, das auf dem Weg von einem Startknoten zu einem Zielknoten ist, vergleicht der Sony Router die Zieladresse mit seinem internen Location Directory und modifiziert dynamisch die Zieladresse, falls er eine hiervon abweichende Adresse gespeichert hat. Auf diese Weise können Pakete bereits vor dem Eintreffen im Heimat–Subnetz des Mobilknotens an einen neuen Zielort umgeleitet werden.

Die Implementation der *Internet Engineering Task Force* (IETF) [Per96] fasst die notwendigen Funktionen in so genannten *Agents* zusammen. Die Readressierung im Heimat–Subnetz und die Verwaltung des Location Directory übernimmt der *Home Agent*. Die Vergabe von Care–Of–Adressen, die inverse Readressierung sowie die Weiterleitung von Datenpaketen an den Mobilknoten obliegt dem *Foreign Agent*, der sich im Subnetz befindet, in welchem der Mobilknoten zu Gast ist. Die jeweilige Rolle nimmt der Agent in Bezug auf den Kommunikationspartner ein. Kommt ein Mobilknoten in ein fremdes Netzwerk, so erfragt er beim dortigen Agent eine Care–Of–Adresse. Der Agent handelt in diesem Falle als *Foreign Agent* des Mobilknotens. Die erhaltene Care–Of–Adresse übergibt der Mobilknoten sodann an den Agenten seines Heimatnetzes, der in diesem Falle als sein *Home Agent* auftritt und die Adresse in sein Location Directory einträgt.

Kommt ein Datenpaket im Heimat–Subnetz an, so leitet der Agent das Paket in seiner Rolle als Home Agent an die Care–Of–Adresse weiter, wo das Paket vom dortigen Agenten in dessen Rolle als Foreign Agent an den Mobilknoten übergeben wird.

In einer Variation des IETF Mobile IP übernimmt der Foreign Agent darüber hinaus auch die Nutzung der Care–Of–Adresse selbst. Wird nach dem zuvor geschilderten Vorgehen die Care–Of–Adresse vom Foreign Agent an den Mobilknoten vergeben und von diesem zu seiner Kommunikation genutzt (sog. *collocated care–of–address*), so agiert im variierten Fall der Foreign Agent selbst als Zielknoten. Seine Adresse wird an den Home Agent zurückgemeldet und er fungiert auch als Ziel der umgeleiteten Kommunikation (*foreign agent care–of–address*). Perkins beschreibt schließlich in [Per98] die Nutzung von DHCP zur Verwaltung der Care–Of–Adressen außerhalb des eigentlichen Foreign Agents.

3.3.3.2 Triangle Routing und andere Anomalien

Gemäß des im letzten Abschnitt beschriebenen Konzepts von Mobile IP läuft der Zugriff auf Mobilknoten, die sich vorübergehend in einem fremden Netzwerk befinden, immer nach demselben Schema ab:

1. Der Dialogpartner sendet seine Datenpakete an die Heimatadresse des Mobilknotens
2. Mittels Readressierung werden die Datenpakete an die *Care–Of–Adresse* des Mobilknotens in einem fremden Netzwerk gesendet
3. Die Datenpakete werden invers readressiert und an den Mobilknoten ausgeliefert
4. Der Mobilknoten sendet seine Antwort an die Adresse des Dialogpartners zurück. Mittels IP–Routing wird diese direkt an das Ziel übermittelt

Da die Datenpakete beim Versand durch den Dialogpartner mit dessen eigener IP–Adresse gekennzeichnet werden und diese nach der inversen Readressierung beim Mobilknoten wieder zur Verfügung steht, muss der Mobilknoten seine Antworten nicht über den Umweg seines *Home Agents* senden. Er kann als Zieladresse direkt die IP–Adresse des Dialogpartners verwenden. Andererseits ist es nicht ohne weiteres möglich, dass er den Dialopartner über seine temporäre *Care–Of–Adresse* informiert. Nachdem jener seine Datenverbindung zur Heimat–Adresse des Mobilknotens geöffnet hat, wird er diese für die Dauer der Kommunikation mit dem Mobilknoten beibehalten. Die dynamische Änderung der IP–Adresse zur Laufzeit ist in der derzeitigen Konzeption von IP nicht vorgesehen und wird deswegen von IP–Implementationen nicht unterstützt. Die Antwort auf das vom Mobilknoten beim Dialogpartner eintreffende Datenpaket wird also wiederum den Umweg über dessen Home–Agent nehmen. Das so entstehende Kommunikationsdreieck wird in der Literatur als *Triangle Routing* beschrieben (vgl. [Per98]). Triangle Routing führt offensichtlich zu einer signifikanten Reduktion der Kommunikationsperformanz zwischen Sender und Empfänger, da jedes Datenpaket den Weg zum Home–Agent nimmt und von dort an den Mobilknoten weitergeleitet wird.

Das Phänomen des Triangle Routing lässt sich für den Aufbau einer Kommunikationsverbindung zwischen Sender und mobilem Empfänger nicht verhindern. Datenpakete werden zunächst immer an die Heimatadresse des Mobilknotens gesendet und dort im Bedarfsfalle an dessen Care–Of–Adresse weitergesendet. Auch für die weitere Kommunikation ist diese Anomalie zunächst nicht zu vermeiden. Damit der Dialogpartner die eintreffenden Datenpakete dem Mobilknoten zuordnen kann, setzt dieser als Absendeadresse seine Heimatadresse in das Datenpaket und *nicht* die temporäre Care–Of–Adresse. Dieses ist auch aus dem Grund notwendig, da nur so sicherzustellen ist, dass die hierauf folgenden Datenpakete des Dialogpartners den Mobilknoten auch dann erreichen, wenn er das Fremdnetz verlässt, um in einem weiteren Fremdnetz eine neue Verbindung aufzubauen. In diesem Falle kann der Home–Agent die Datenpakete an die neue Care–Of–Adresse weiterleiten, während sie im vorherigen Fremdnetz nicht länger zugestellt werden könnten und verloren gehen würden.

Triangle Routing lässt sich ohne tiefgreifende Änderungen in der IP–Implementation nicht verhindern und ist somit derzeit nur dann zu vermeiden, wenn beide Kommunikationspartner über einen speziellen Mobile IP Protokollstack verfügen. Für die kommende Version 6 des IP–Protokolls wurde jedoch von Beginn an auf die Unterstützung von Mobilität geachtet und es wurden notwendige Techniken zur dynamischen Readressierung und zum Caching von Adressen direkt integriert. Auf diese Techniken wird in Abschnitt 3.3.3.3 näher eingegangen.

Die zuvor beschriebenen Konzepte von Mobile IP basieren auf der idealisierenden Vorstellung, dass die Netzwerke, an welche sich Mobilknoten anmelden, direkt miteinander kommunizieren können. Der Austausch von Care–Of–Adressen basiert genauso hierauf wie die Weiterleitung von Datenpaketen vom Home Agent an den Mobilknoten. Gerade wenn als Verbindungsglied zwischen den Netzwerken das Internet zum Einsatz kommt, ist die bidirektionale Kommunikation zwischen allen beteiligten Netzwerkknoten offensichtlich nicht zu gewährleisten. *Firewalls* an den Schnittstellen zwischen lokalen Netzen und dem Internet verhindern zumeist die Weiterleitung von nicht autorisiertem Datenverkehr in das lokale Netzwerk und schützen dieses so vor möglichen Angriffen. Auch in entgegengesetzter Richtung werden Firewalls oftmals aktiv. Um eine direkte Kommunikation mit dem Internet zu unterbinden, werden Datenpakete meist nicht uneingeschränkt weitergeleitet, sondern lediglich die Kommunikation einiger weniger Rechner (z.B. Mailserver oder Webserver) erlaubt.

Das hinter Mobile IP stehende Konzept der Weiterleitung von Datenpaketen ist grundsätzlich nicht kompatibel zur Nutzung von Firewalls. Mobile IP beruht auf einer ungehinderten Kommunikation zwischen Mobilknoten und deren Heimatnetz, während Firewalls die externe Kommunikation auf ein Minimum beschränken sollen. Die Freischaltung von externem Datenverkehr für einzelne IP Adressen innerhalb des lokalen Netzes bzw. die Freischaltung für die Interaktion mit bestimmten externen IP Adressen stellt ein potentielles Risiko dar, welches üblicherweise nicht hinzunehmen sein wird.

Auch aus einem weiteren Grunde stellen Firewalls ein fundamentales Problem für die Kommunikation zwischen Dialogpartner, Home Agent und Mobilknoten dar. Um die Vortäuschung einer falschen IP Adresse, das so genannte *IP Spoofing*, zu verhindern, werden Firewalls und sogar Gateways zunehmend dahingehend

konfiguriert, dass sie Datenpakete verwerfen, deren Absenderadresse im Widerspruch zum Netzwerk steht, aus dem sie kommen. Sendet also beispielsweise ein Mobilknoten aus dem Fremdnetzwerk heraus Datenpakete, als deren Absenderadresse er seine eigene *Home Adress* einträgt, so wird dieses vom Firewall als der Versuch eines korrupten Netzwerkknotens interpretiert, aus dem lokalen Netz heraus Daten unter einer fremden Identität zu versenden. Die fraglichen Pakete werden entsprechend verworfen. Werden die Pakete andererseits unter der *Care–Of–Adresse* des Mobilknotens verschickt, so kann der Dialogpartner sie möglicherweise nicht als vom Mobilknoten kommend interpretieren bzw. wird sie nicht dem richtigen Kommunikationsendpunkt zuordnen.

Für keines der genannten Probleme gibt es derzeit eine vollständige Lösung. Der Firewall–Problematik lässt sich zum Teil durch manuelle Anpassung der Firewalls selbst begegnen, insbesondere aber in sensitiven Netzwerken wird auch diese Lösung nicht als befriedigend zu werten sein. Für das Triangle Routing gibt es schließlich nur die Lösung einer Anpassung auf Seiten der Dialogpartner, d.h. möglicherweise *aller* auf dem IP Protokoll basierenden Knoten, was bis auf weiteres als unrealistisch bewertet werden darf. Eine durchgehende Behandlung der Probleme lässt sich somit erst mit der Einführung von IPv6 und mit ihm auch Mobile IPv6 erwarten, da hier bereits beim Entwurf aller Bereiche des Kommunikationsprotokolls auf die Berücksichtigung mobiler Knoten Rücksicht genommen werden kann. Der folgende Abschnitt geht näher auf die wesentlichen in diesem Zusammenhang wichtigen Aspekte ein.

3.3.3.3 Mobile IPv6

Um den bekannten Problemen der aktuellen Version 4 von IP zu begegnen, wurde ein grundsätzliches Redesign des Protokolls durchgeführt. Verfügbare Erweiterungen zu IPv4 sowie neue Ansätze wurden hierzu in einem vereinheitlichten Protokoll, der Version 6 von IP (IPv6), zusammengefasst. Als wesentliche Merkmale von IPv6 werden in der Literatur die folgenden Punkte genannt (vgl. [Per98]):

- Wesentlich vergrößerter Adressraum
- Inhärente Signatur– und Verschlüsselungskonzepte
- Zustandsloses Protokoll zur autonomen Vergabe von IP–Adressen
- Möglichkeit zur dynamischen Renummerierung von Subnetzen
- Effizienteres Routing von Datenpaketen
- Unterstützung für mobile Netzknoten

Wie sich aus obiger Aufzählung ergibt, spielte bei der Entwicklung von IPv6 von Beginn an auch die Unterstützung von mobilen Netzwerkknoten eine wichtige Rolle. Insbesondere die automatische Vergabe eindeutiger IP–Adressen stellt im Zusammenhang mit der Unterstützung mobiler Netzwerkknoten eine wichtige

Eigenschaft von IPv6 dar. Ohne dass hierfür ein spezieller Netzwerkdienst notwendig wäre oder das Subnetz anhand manueller Konfiguration zur Vergabe freier IP–Adressen vorbereitet werden müsste, kann der Mobilknoten die Subnetz–Adresse erfragen und danach durch Kombination seiner MAC–Adresse mit der erfragten Subnetz–Adresse eine IPv6–Adresse generieren. Die automatisch generierte IP–Adresse entspricht den Anforderungen an eine gültige Adresse, da sie sowohl *eindeutig* ist (sie enthält neben der Subnetz–Adresse die für das Medium eindeutige MAC–Adresse), als auch den topologischen Gegebenheiten von IP entspricht und somit Routing–fähig ist (sie enthält die Subnetz–Adresse, anhand derer Pakete durch das IP–Netzwerk zum Subnetz geleitet werden können). Da die autonome Vergabe von IP–Adressen inhärentes Merkmal von IPv6 ist, müssen Netzwerke somit nicht speziell für die Integration von Mobilknoten modifiziert werden, was insbesondere im Falle der für IP–Netzwerke elementaren Vergabe von eindeutigen Adressen für alle Knoten kritisch sein kann. Ist andererseits die dedizierte Vergabe von Adressen gewünscht, so ist diese auch unter IPv6 möglich.

Wie im vorigen Abschnitt über die Anomalien von Mobile IPv4 beschrieben wurde, führt die Unfähigkeit von Netzwerkknoten in IPv4 zur eigenständigen Verwaltung eines Adressencaches zum Triangle Routing. Unter IPv6 kann mittels so genannter *Binding Caches* dieses Problem verhindert werden (vgl. [PJ96, PJ00]). Erhält der Dialogpartner ein Datenpaket vom Mobilknoten, das nicht dessen Heimatadresse trägt, so generiert er ähnlich dem Home Agent ein Tupel, das der Heimatadresse die Care–Of–Adresse beiordnet. Alle folgenden Datenpakete an den Mobilknoten werden nicht mehr über den Home Agent verschickt, sondern direkt an die Care–Of–Adresse geleitet.

Binding Caches stellen die wesentliche Erweiterung von Mobile IPv6 gegenüber Mobile IPv4 dar. Sie führen zu einer erheblichen Effizienzsteigerung, da Datenpakete nur einmalig den Umweg über den Home Agent nehmen müssen. Durch dynamische Anpassung des lokalen Caches kann der Dialogpartner auf die veränderte Adresse des Mobilknotens reagieren und in der Folge entsprechend optimiert arbeiten. Zu einer nochmaligen Verbesserung führt die Einführung entsprechender Protokollnachrichten, mit deren Hilfe der Mobilknoten quasi proaktiv handeln kann. Ändert sich seine Care–Of–Adresse infolge einer Relokation, so sendet er nicht nur wie bisher an seinen Home Agent, sondern zusätzlich an alle ihm bekannten Dialogpartner eine spezielle Nachricht, in welcher er sie zu einer Aktualisierung ihres Binding Caches auffordert (sog. *Binding Update*). War bisher nur der Home Agent in der Lage diese Information zu verwalten, so ist dieses mit Hilfe des Binding Caches nun auch beliebigen anderen Netzwerkknoten möglich. Binding Updates können eine Aufforderung zur Bestätigung enthalten (sog. *Binding Acknowledgement*). Ist eine solche Bestätigung im Normalfall nicht notwendig, so ist sie im Falle des Home Agent zwingend.

Schließlich ist im Gegensatz zu Mobile IPv4 (und aufgrund der zuvor genannten Eigenschaften) unter Mobile IPv6 nicht länger ein Foreign Agent notwendig. Die Vergabe eindeutiger Care–Of–Adressen kann vom Mobilknoten selbst übernommen werden, ohne dass hierfür der Dienst eines Agents notwendig wäre. Die Weiterleitung von Datenpaketen *innerhalb* des Fremdnetzes muss nicht durch den Foreign Agent übernommen werden, da die lokalen Netzwerkknoten bzw. Gateways die Care–Of–Adresse des Mobilknotens in ihrem Binding Cache zwischenspeichern können. Und war für die Weiterleitung der readressierten

Datenpakete vom Dialogpartner zum Mobilknoten unter IPv4 ein Tunnel zwischen einem Home Agent und einem Foreign Agent notwendig, so kann die Rolle des Tunnelausgangs vom Mobilknoten selbst übernommen werden. IPv6 schreibt die entsprechende Fähigkeit für *alle* Netzwerkknoten zwingend vor.

Mobile IP stellt mit seinen verschiedenen Varianten einen wesentlichen Integrationsansatz auf Ebene der Netzwerkschicht dar. Im Gegensatz zum herkömmlichen IP–Protokoll ist die Identifikation eines Netzwerkknotens nicht länger mit seiner topologischen Position verkettet, womit Mobile IP eines der Kernprobleme von IPv4 im Kontext mobiler Systemumgebungen behebt. Mobilgeräte bleiben auch nach einer Relokation noch unter ihrer ursprünglichen Adresse erreichbar, ohne dass hierfür eine zentrale Instanz etabliert werden müsste, die für die Überwachung teilnehmender Knoten aktiv wird. Wichtig ist dabei, dass Mobile IP einen *anwendungstransparenten* Integrationsansatz darstellt. Kommunikationsprozesse höherer Abstraktionsebenen nutzen zur Identifikation eines entfernten Mobilgerätes dessen unveränderliche Heimatadresse, die in jedem Fall neben der temporären Care–Of–Adresse erhalten bleibt. Eingehende Datenpakete sind so auch dann noch zuzuordnen, wenn der mobile Knoten seine Position geändert hat. Weder für den Mobilknoten noch für den stationären Knoten ist aus der Sicht der Anwendungsprogramme die Relokation ersichtlich.

3.3.4 Transportschicht

Integrationsansätze auf der Ebene der Transportschicht abstrahieren von den Problemen der Adressierung von Netzwerkknoten und von der Weiterleitung von Datenpaketen innerhalb von Netzwerken. Im Gegensatz zu den im vorigen Abschnitt besprochenen Ansätzen auf der Netzwerkebene gehen sie von der Existenz einer Netzwerkinfrastruktur aus, in welcher jedem Knoten eine eindeutige Identität zugeordnet ist und sich Datenpakete auf wohldefinierten Routen von einem Startknoten zu einem Zielknoten durch das Netzwerk leiten lassen.

Als der wichtigste Ansatzpunkt für die Realisierung von Integrationsansätzen auf dieser Ebene haben sich die Phänomene erwiesen, die gerade in IP–basierten Netzwerken zu beobachten sind, wenn einzelne oder gar alle Knoten mobil sind. Diese Phänomene sind vornehmlich auf die in IP inhärenten Mechanismen zur Vermeidung von Überlastungen bzw. Stauungen (engl. *congestion*) zurückzuführen, welche dessen Erfolg in festen Netzen ursprünglich überhaupt erst möglich machten, die jedoch gerade im Einsatz auf drahtlosen Übertragungsstrecken zu Missinterpretationen von Fehlerzuständen und in der Folge zu unangemessenen Reaktionen hierauf führen können.

Eine der grundsätzlichen Eigenschaften von IP–Netzen liegt darin, dass Datenpakete in ihnen auf dynamisch ermittelten Wegen vom Startknoten zum Zielknoten befördert werden. Größere Datenmengen werden in leichter vermittelbare Dateneinheiten segmentiert und vom Startknoten an einen sog. Router übergeben, der den weiteren Transport in Richtung des Zielknotens übernimmt. Router enthalten eine Liste ihnen bekannter Netzwerke bzw. Router, an welche sie Datenpakete anhand der in den Datenpaketen gespeicherten Zieladressen weiterleiten können. Die Datenpakete werden so von Router zu Router weitergeleitet, bis sie schließlich am Zielknoten abgeliefert werden. Gerade bei hohem Datenverkehr

kann es vorkommen, dass Router mit der Weiterleitung von Paketen in Verzug geraten und sich die Pakete bei einer solchen Vermittlungsstelle in einer Warteschlange einreihen, sich also eine Stauung (*congestion*) aufbaut. Wird in diesem Falle nicht in geeigneter Form reagiert, kann es zu einer Kettenreaktion kommen, in welcher der Startknoten oder vorherige Router die Verzögerung als Datenverlust interpretieren und hierauf durch erneutes Senden der Daten reagieren. In der Folge kann es dann zu einem totalen Zusammenbruch der Verbindungen kommen. Das beschriebene Phänomen kann auf allen Ebenen der Netzwerkkommunikation auftreten, ist aber besonders problematisch auf der Transportschicht mit der hier definierten Ende–zu–Ende Semantik (vgl. [SRC81] bzw. [SRC84] für eine Beschreibung der Ende–zu–Ende Semantik).

IP nutzt als einen wichtigen Parameter zur Behandlung des geschilderten Problems die maximale Paketgröße, die auf der Strecke zwischen Sender und Empfänger übertragen werden kann, ohne dass es zu Stauungen kommt. Entsprechend seines Zwecks wird dieser Parameter als *Congestion Window* bezeichnet. Im Zuge des Verbindungsaufbaus wird das Congestion Window auf einen bestimmten Wert festgelegt, der zunächst die Menge der übertragenen Datenpakete vorgibt. Kann die vorgegebene Anzahl an Datenpaketen übertragen werden, ohne dass es zu einer Zeitüberschreitung kommt, so wird hieraus geschlossen, dass keine Stauung vorliegt und das Congestion Window entsprechend vergrößert, bevor die nächsten Datenpakete übertragen werden. Hierbei wird nach demselben Prinzip verfahren und wiederum das Congestion Window vergrößert, wenn die Übertragung ohne Zeitüberschreitung vonstatten ging. Das Congestion Window wird nun von Übertragung zu Übertragung exponentiell vergrößert, bis entweder ein definierter Maximalwert erreicht ist, den der Empfänger zu verarbeiten in der Lage ist, oder es zu Zeitüberschreitungen kommt. Ist letzteres der Fall, so wird das Congestion Window wieder auf den Minimalwert zurückgesetzt und der Algorithmus beginnt erneut, wobei mit dem Datenpaket begonnen wird, für welches zuvor die Zeitüberschreitung aufgetreten ist. Entsprechend des Verhaltens, die Datenrate im Verlauf zu steigern, wird dieses Vorgehen als *Slow Start* Algorithmus bezeichnet [Jac88]. Obwohl diverse Variationen des Slow Start Algorithmus existieren, gilt doch für alle das oben genannte Grundprinzip.

In IP–Netzwerken wird üblicherweise die Messung von Zeitüberschreitungen genutzt, um den Verlust von Datenpaketen zu erkennen. Insbesondere aufgrund der Tatsache, dass in Festnetzen die Übertragungsqualität mittlerweile so hoch ist, dass der Verlust von Datenpaketen durch Übermittlungsfehler weitgehend ausgeschlossen werden kann, werden Zeitüberschreitungen als wichtigstes Indiz dafür gewertet, dass eine Stauung vorliegt (vgl. [Tan96] S. 536 ff.). Kann ein Router die eingehenden Datenpakete nicht mehr in dem Maße weiterleiten, wie sie bei ihm eingehen, so stauen sie sich in einer Warteschlange auf. Hält die Überlastungssituation über längere Zeit an, so kommt es letztlich zu Zeitüberschreitungen und mittels oben beschriebenem Algorithmus wird die Größe der Datenpakete auf einen Minimalwert reduziert, der es dem Router im Normalfalle möglich machen sollte, die Überlast abzubauen. Nach einem kurzzeitigen Einbruch der Übertragungsrate wird die Datenpaketgröße wieder kontinuierlich erhöht und die Stauung so zumeist abgebaut. Die Verwendung von Congestion Windows und dem Slow Start Algorithmus wird als ein wesentlicher Grund für die Skalierbarkeit des Internet angesehen. Die Art, in der das Netz autonom auf das Auftreten von Fehlerzuständen reagieren kann, wird zum Teil auch als

„Selbstheilung“ bezeichnet.

Die Nutzung von Zeitüberschreitungen zur Indikation von Stauungen und die hierauf folgende Reaktion durch Herabsetzung der Datenrate hat im Kontext mobiler Übertragung fatale Folgen. Im Gegensatz zu Festnetzen werden die Zeitüberschreitungen in diesem Falle nämlich seltener durch Überlastung von Routern verursacht als vielmehr durch ungeplante Verbindungsabbrüche, wie sie durch die verminderte Signalqualität in schlecht oder gar nicht versorgten Gebieten verursacht werden sowie durch geplante Verbindungsabbrüche, wie sie beispielsweise im Falle von *Hand–off* Prozeduren beim Übergang zwischen zwei Funkzellen auftreten. In beiden Fällen führt die Unterbrechung zwangsläufig zu Kommunikationsfehlern und in deren Folge zu Zeitüberschreitungen, auf welche IP wie beschrieben durch eine drastisch reduzierte Datenrate reagiert.

Gerade im Falle des Hand–Off ist es offensichtlich, dass die Reaktion mittels Slow Start dazu führt, dass die Kommunikationsverbindung zwischen Mobilknoten und Festnetzknoten unter Umständen nie die volle zur Verfügung stehende Bandbreite nutzen kann, sondern vielmehr nahe des minimalen Wertes verbleibt. Dieses trifft insbesondere für kleinste Funkzellen (sog. *Picocells*) zu, da es hier vermehrt zu Zellenwechseln kommt – gerade unter dem Aspekt ein paradoxes Phänomen, dass die wichtigste Eigenschaft von Picocells gerade darin liegt, dass sie die Realisierung hoher Übertragungsgeschwindigkeiten der in ihnen aktiven Mobilknoten ermöglichen. Das wichtigste Integrationsziel auf Ebene der Transportschicht liegt also primär darin, die negativen Eigenschaften von IP auszugleichen, die sich aus der Fehlinterpretation von Ereignissen innerhalb des Netzwerkes ergeben und die auf inhärente Designmerkmale der Teilprotokolle zurückzuführen sind.

Grundsätzlich müssen hierbei diverse Punkte beachtet werden, die u.a. bei Brown et al. in der folgenden Weise zusammengefasst wurden [BS97]:

- Die Performanz sollte deutlich gesteigert werden
- Die Ende–zu–Ende Protokollsemantik sollte nicht beeinträchtigt werden
- Häufige bzw. längere Unterbrechungen müssen behandelt werden
- Eine dynamische Anpassung an sich ändernde Bandbreiten sollte möglich sein
- Eine effiziente Übergabe zwischen einzelnen Zellen sollte realisierbar sein.

Insbesondere die Erhaltung der Ende–zu–Ende Protokollsemantik stellt dabei eine grundsätzliche Schwierigkeit dar, da nur ein für die Kommunikationspartner *transparenter* Ansatz diese Forderung erfüllen kann. Wie in der Literatur jedoch immer wieder berichtet wird, steht die Realisierung einer transparenten Integrationsumbegung im direkten Widerspruch zur Forderung nach einer deutlich zu steigernden Performanz (vgl. [BPSK96, BS97, SRBW99, Wol99]).

Im Prinzip lassen sich zwei Herangehensweisen identifizieren, nach denen die Integration auf der Ebene der Transportschicht realisiert werden kann [SRBW99]:

Desktop–Approach : Auf der Seite des Mobilgerätes wird eine vollständige Protokollimplementation realisiert, die in geeigneter Form auf die spezifischen Probleme der Gerätemobilität angepasst ist.

Frontend–Approach : Auf der Seite des Mobilgerätes wird lediglich eine minimale Protokollimplementation realisiert, wie sie zur Überwindung der Kommunikationsstrecke zwischen dem Gerät selbst und seiner Basisstation notwendig ist, seinem unmittelbar nächsten Kommunikationspartner im Festnetz also.

Der erstgenannte Ansatz scheint zunächst die naheliegendere Wahl darzustellen. Die vollwertige Protokollimplementation auf dem Mobilgerät ermöglicht eine weitestgehende Behandlung von Fehlerfällen durch das Gerät selbst. Auch die Forderung nach der Erhaltung der Ende–zu–Ende Semantik scheint sich einfacher gewährleisten zu lassen, da eventuell ein notwendiges Eingreifen in den Protokollfluss durch den Endpunkt selbst erfolgen kann. Schließlich zeichnet sich der Desktop–Approach dadurch aus, dass er insgesamt zu einer geringeren Komplexität der Gesamtarchitektur beiträgt, da keine Unterstützungsdienste realisiert und in die Kommunikation zwischen Sender und Empfänger eingebunden werden müssen.

Für den Frontend–Approach spricht auf der anderen Seite, dass er zu einer weniger komplexen und somit schlankeren Protokollimplementation auf dem Mobilgerät führt. Gerade unter dem Aspekt der Speicher– und Leistungsökonomie gilt dieses als wünschenswert. Ein weiterer Vorteil liegt darin, dass die Einführung eines Stellvertreters an der Nahtstelle zwischen Fest– und Mobilnetz eine natürliche Entkopplung zwischen Sender und Empfänger darstellt. Insofern, als sowohl der Stellvertreter als auch der Mobilknoten um die besonderen Charakeristika des mobilen Kontextes wissen, kann optimal hierauf reagiert werden, beispielsweise, indem zwischen Stellvertreter und Mobilknoten ein spezielles Protokoll eingesetzt wird. Hierbei ist jedoch darauf zu achten, dass die Entkopplung nicht zu einer Verletzung der Ende–zu–Ende Semantik zwischen Sender und Empfänger führt.

Die in der Literatur beschriebenen Integrationsansätze lassen sich im Wesentlichen anhand der folgenden drei Klassen einordnen (vgl. [Wol99])[3]:

- TCP Add–Ons
- Split Connections
- Protocol Booster

TCP Add–Ons stellen eine Variante dar, bei welcher das TCP Protokoll um die notwendigen Eigenschaften erweitert wird, wie sie der Einsatz in mobilen Kontexten erfordert. Im Wesentlichen wird dabei zunächst versucht, auf Protokollebene eine genauere Unterscheidung von Paketverlusten durch Stauungen

[3]Während sich das im Folgenden gesagte grundsätzlich auf beliebige Protokolle der Transportschicht anwenden lässt, beschränken sich die weiteren Ausführungen auf TCP, das *Transmission Control Protocol* von IP. Für eine detaillierte Beschreibung von TCP sei z.B. auf [Ste94] verwiesen.

gegenüber solchen durch tatsächliche Störungen auf dem drahtlosen Übertragungsweg zu ermöglichen. Anhand der somit genaueren Diagnostik ist es danach möglich, Nachrichten durch das Netzwerk an den Sender zurückzuübermitteln, welche diesen im Falle von Signalisierungsfehlern zu einem erneuten Senden der verlorenen Datenpakete veranlassen können, ohne dass es hierbei zu dem oben beschriebenen *Slow Start* kommen würde.

Liegt andererseits ein Paketverlust durch Stauungen zwischen Sender und mobilem Empfänger vor, wie er etwa durch die verminderte Übertragungskapazität auf der Funkstrecke auftreten könnte, so kann dieses zurückgemeldet werden und dann entsprechend durch die Kombination aus erneutem Senden der verlorenen Datenpakete sowie einer Verringerung der Paketgröße reagiert werden. Wird der Versand expliziter Benachrichtigungen über den Verlust von Datenpaketen zusätzlich durch die lokale Pufferung von Datenpaketen in der Basisstation erweitert, so lassen sich teilweise Übertragungsraten zwischen Sender und mobilem Empfänger erzielen, die nahe an das theoretische Maximum heranreichen (vgl. [BKVP97]). Beispiele für Realisierungen nach dem Ansatz der TCP Add–Ons sind etwa SACK [MMFR96], *Explicit Congestion Notification* (ECN) [RF99] oder *Explicit Bad State Notification* (EBSN) [BKVP97].

Wie unter anderem in [SRBW99] berichtet wird, besitzen TCP Add–Ons im Kontext mobiler Anwendungen nur begrenzten Nutzen. Zum einen führen sie zu der bereits genannten Steigerung der Komplexität der Protokollimplementation auf dem Mobilgerät. Des Weiteren müssen Fehlerzustände durch das gesamte Netzwerk vom Mobilgerät (bzw. dem letzten Knoten davor) zurück zum Sender propagiert werden, was zu einer nur mäßigen Leistung führt. Schließlich bedingt die Einführung eines erweiterten Satzes an Nachrichten auf Protokollebene, dass alle Knoten des Netzwerkes um diese erweitert werden müssen, was einem Einsatz in öffentlichen Netzwerken entgegensteht.

Split Connections führen an der Nahtstelle zwischen Festnetz und Funknetz eine Stellvertreterkomponente ein, die als Vermittlungsinstanz zwischen dem mobilen Gerät und den im Festnetz angesiedelten Kommunikationspartnern auftritt. Logisch wird dabei die Kommunikationsstrecke zwischen Sender und Empfänger in zwei Segmente aufgeteilt, eine TCP–Verbindung zwischen Festnetzknoten und Stellvertreter sowie eine weitere, ggf. auf die speziellen Anforderungen des drahtlosen Übertragungsmediums angepasste, Verbindung zwischen Stellvertreter und dem Mobilgerät selbst. Die Kernidee von Split Connections besteht darin, die Probleme der Funkübertragung vollständig vor den Netzwerkknoten zu verbergen. Hierzu agiert der Stellvertreter zunächst anstelle des Mobilknotens, indem er die an den Mobilknoten gerichteten Datenpakete zwischenspeichert und gegebenenfalls vorläufig bestätigt, bevor er sie an diesen weiterleitet. Auf diese Weise können Zeitüberschreitungen und damit auch *Slow Starts* vermieden werden, wenn der Mobilknoten vorübergehend nicht in der Lage ist, die Datenpakete anzunehmen. Die Datenpakete werden sodann vom Stellvertreter an den Mobilknoten weitergeleitet, wobei entweder das unmodifizierte TCP–Protokoll verwendet werden kann (der Vorteil durch den Einsatz eines Stellvertreters reduziert sich in diesem Falle auf die Verhinderung von Slow Starts durch Zeitüberschreitungen) oder ein optimiertes „Last–Hop–Protokoll“ zum Einsatz kommen kann. Die Verwendung eines solchen Protokolls ermöglicht beispielsweise die Datenkompression vor Übertragung oder die Ausnutzung spezieller Eigenschaften des zugrunde liegenden Funkprotokolls zwecks optimierter Datenübertragung.

Das Konzept der Split Connections kommt beispielsweise im Falle der *Remote Socket Architecture* (ReSoA) [SRBW99] sowie dem *Mobile Transmission Control Protocol* (M–TCP) [BS97] zur Anwendung.

Split Connections weisen insofern eine günstige Charakteristik auf, als sie die besonderen Eigenschaften des drahtlosen Mediums optimal ausnutzen können, ohne dass hierbei eine Modifikation auf Seiten der Festnetzknoten notwendig wäre. Die Kommunikation mit einem mobilen Empfänger verläuft somit aus der Sicht des stationären Netzknotens quasi transparent. Hierbei kann zwischen Stellvertreter und Mobilknoten ein proprietäres Protokoll zum Einsatz kommen, das die genannten Forderungen nach einer minimalen Belastung des Mobilgerätes sowie die optimale Übertragungsgeschwindigkeit ermöglichen kann. Schließlich kann das Auftreten von Slow Starts weitgehend vermieden werden, da der Stellvertreter Datenverluste auf der Funkstrecke durch erneutes Senden lokal zwischengespeicherter Datenpakete lokal behandeln und ggf. durch vorzeitiges Senden von Bestätigungen Fehlhandlungen durch den Sender verhindern kann.

Das Hauptproblem liegt bei Split Connections darin, dass die Zerlegung der Kommunikationsstrecke zwischen Sender und Empfänger zu einer Verletzung der Ende–zu–Ende Semantik führen kann. Dieses kommt insbesondere dann zum Tragen, wenn der Stellvertreter eingehende Datenpakete bestätigt, bevor er sie an den mobilen Empfänger weitergeleitet und von diesem eine Empfangsbestätigung bekommen hat. Befindet sich der Mobilknoten im Übergang in eine neue Zelle, so kann der Stellvertreter ihn unter Umständen nicht erreichen, und das Datenpaket kann entsprechend nicht zugestellt werden. Ist die Funkstrecke zwischen Stellvertreter und Mobilknoten gestört, so ist es denkbar, dass das Datenpaket, dessen Empfang bereits bestätigt wurde, mehrfach erneut vom Stellvertreter zum Mobilknoten gesendet werden muss, bevor dieser es tatsächlich empfängt. Schließlich kann durch die Einführung eines optimierten Kommunikationsprotokolls zwischen Stellvertreter und Mobilknoten eine geänderte Protokollsemantik entstehen, die ihrerseits im Widerspruch zur Ende–zu–Ende Semantik von TCP steht.

Protokoll Booster stellen schließlich eine Zwischenform von Desktop–Ansatz und Frontend–Ansatz dar. Wie bei der Split Connection wird auch hier an der Nahtstelle zwischen Festnetz und Funknetz ein Stellvertreter etabliert, der in den Kommunikationsverkehr zwischen Sender und Empfänger eingreift. Im Gegensatz zur Split Connection wird hierbei jedoch keine Entkopplung realisiert, sondern lediglich eine transparente Optimierung angestrebt.

3.3.4.1 Beispiele

Die Verwendung von *Explicit Bad State Notifications* (EBSN) [BKVP97] kann als exemplarisch für den Einsatz von TCP Add–Ons angesehen werden. Der Ansatz beruht auf der Beeinflussung von Zeitüberschreitungen, wie sie von TCP zur Identifizierung von Fehlern auf der Kommunikationsstrecke zwischen Sender und Empfänger zum Einsatz kommen. Wie bereits beschrieben legt der Sender für jedes Datenpaket eine maximale Zeit fest, bis zu der er eine Empfangsbestätigung durch den Empfänger erwartet. Geht diese nicht vor Ende des Zeitintervalls ein,

so wird das Datenpaket als verloren gewertet und es werden alle Datenpakete *einschließlich* des verlorenen Datenpaketes erneut gesendet.

Im Einzelnen wird wie folgt vorgegangen: Stellt die Basisstation einen Verbindungsabbruch zum Mobilgerät fest, so sendet sie eine *Bad State Notification* an den Sender der empfangenen Datenpakete. Diese Nachricht veranlasst den Sender zu einer Neuberechnung der Zeitzähler für die versandten Datenpakete, wodurch deren Gültigkeitsintervall zunächst vergrößert wird. Des Weiteren wird die Übertragung weiterer Pakete gestoppt, so dass es zu keiner Stauung von Datenpaketen in der Warteschlange der Basisstation kommen kann. Die Basisstation versucht nun kontinuierlich, die lokal zwischengespeicherten Datenpakete an den Mobilknoten zu übertragen und sendet bei jedem Fehlversuch eine EBSN an den Sender zurück, so dass dieser weiterhin sowohl am Verwerfen der Pakete als auch am Versand weiterer Pakete gehindert wird. Die Kommunikation zwischen Sender und Empfänger wird quasi blockiert. Ist der Empfänger schließlich wieder erreichbar, so kann dieser eine gewöhnliche Empfangsbestätigung generieren, die an den Sender zurückgeleitet wird. Bei Empfang der Bestätigung reagiert der Sender, indem er das bestätigte Datenpaket als empfangen verbucht und sodann mit der Aussendung weiterer Datenpakete fortfährt.

In [BKVP97] berichten Bakshi et al. von Labormessungen, wonach sich die Übertragungsrate durch den Einsatz von EBSN gegenüber herkömmlichem TCP um bis zu 100% steigern lässt und damit nahe an das theoretische Maximum herangeführt werden kann. Während eine Modifikation sowohl des Senders als auch der Basisstation zur Realisierung von *Explicit Bad State Notifications* notwendig ist, lässt sich diese nach Meinung der Autoren sehr gut isolieren und auf wenige Zeilen Code der TCP Implementation beschränken. Gerade in Kontexten, in welchen Sender und Empfänger einer geschlossenen Gruppe angehören und somit eine begrenzte Zahl von Netzwerkknoten zu modifizieren ist, eignen sich EBSN somit zur Optimierung von Datenübertragungen in mobilen Kontexten.

Kapitel 4

Ein generischer Integrationsansatz auf Anwendungsebene

Nachdem im vorigen Kapitel auf konzeptionelle Grundlagen der Integration mobiler Geräte eingegangen und ein Überblick über die möglichen Integrationsansätze auf niederen Ebenen gegeben wurde, soll in diesem Kapitel die Anwendungsebene als Basis für Integrationsbemühungen herausgehoben und auf diese genauer eingegangen werden. Im Gegensatz zu den anderen Ebenen ermöglicht die Anwendungsebene einen deutlich höheren Grad an Unterstützung für die Realisierung eines allgemeinen Integrationsansatzes. Während das Einsetzen auf anderen Ebenen fast ausnahmslos grundlegende Modifikationen an der vom Mobilgerät aus zu nutzenden Anwendung bzw. am Mobilgerät selbst notwendig macht, kann es auf der Anwendungsebene eher gelingen, sowohl Anwendungstransparenz als auch Gerätetransparenz zu gewährleisten.

Ziel dieses Kapitels ist es nun, ein konzeptionelles Fundament zu schaffen, auf dem eine umfassende Integrationsunterstützung für Mobilgeräte auf Anwendungsebene fußen kann. Es werden hierzu zunächst die allgemeinen Ziele eines Herangehens auf dieser Ebene herausgearbeitet, wobei zur Illustration auf bestehende Ansätze auf dieser Ebene eingegangen wird. Im Anschluss wird dann gesondert zunächst auf die Einbindung von Dienstnehmern und dann auf die Einbindung von Diensterbringern eingegangen, wobei die Schlüsselkonzepte im Vordergrund stehen sollen, die im Zuge der Integration zum Einsatz kommen. Eine Zusammenfassung rundet dieses Kapitel schließlich ab.

4.1 Die Anwendungsebene als Integrationsfundament

Grundsätzlich lassen sich bei einem Einsetzen auf Anwendungsebene verschiedene Vorgehensweisen denken. Angefangen bei der Realisierung eigener Anwendungsprotokolle, die zwischen eigens dafür entwickelten Programmen eingesetzt

werden, über verteilte Anwendungssysteme auf der Basis einer etablierten Middleware, die Verwendung von Agentensystemen, in denen mobiler Code zwischen beteiligten Endpunkten ausgetauscht wird, bis hin zu Thin–Client– oder Applikationsserversystemen, bei denen nicht der Anwendungscode auf den entfernten Endpunkt übertragen wird, sondern ausschließlich die Benutzungsschnittstelle der zu nutzenden Anwendung. Die verschiedenen Vorgehensweisen unterscheiden sich vor allem durch den Aufwand, der auf den beteiligten Kommunikationsknoten zu betreiben ist, um die Integration möglich zu machen.

In Anlehnung an das im vorigen Kapitel vorgestellte Klassifikationsspektrum zwischen anwendungsbewussten sowie anwendungstransparenten Ansätzen lässt sich ein Schema zur Bewertung von Integrationsansätzen auf Anwendungsebene verwenden, das den Aufwand für die Integration entlang zweier Achsen bewertet: Dem Grad an Anwendungstransparenz und dem Grad an Gerätetransparenz, die durch den gewählten Integrationsansatz ermöglicht werden. Je höher der Grad an Transparenz auf einer Achse ist, den die Integrationsinfrastruktur bietet, desto geringer ist der zu erwartende Aufwand, der im Zuge der Integration anfällt.

Während es nun prinzipiell erstrebenswert erscheint, möglichst weitgehende Transparenz zu erreichen, so ist zu bedenken, dass der Wunsch nach einer optimalen Anpassung an die Eigenschaften des Anwendungssystems und die zu verwendenden Geräte diesem Streben entgegensteht. Nur indem man die Merkmale der Anwendung und des Endgerätes kennt, kann man diese bestmöglich unterstützen bzw. aufeinander abbilden. Grundsätzlich gilt dieses für jede Form der Integrationsunterstützung, unabhängig davon, ob diese auf Ebene der Sicherungsschicht, der Sitzungsschicht oder eben der Anwendungsschicht ansetzt. Gerade die Anwendungsebene lässt hierfür jedoch vergleichsweise großen Gestaltungsspielraum, was den Grad an Transparenz angeht, der durch die Integrationsumgebung ermöglicht werden kann.

Transparenz kann nicht auf allen Ebenen erreicht werden. Die notwendigen Voraussetzungen für eine Interaktion müssen letztlich auf einer der Ebenen geschaffen werden. Indem sich die Anwendungsschicht auf die ihr zugrunde liegenden Schichten stützen kann, ist die Wahrscheinlichkeit sehr hoch, dass etwaige Anpassungen bereits in jenen Schichten erfolgt sind. Die Bereitstellung der für die Interaktion grundsätzlich notwendigen Kommunikationskanäle, die Realisierung von Kommunikationsprotokollen usw. durch die niederen Schichten sorgt meist für eine einheitliche Plattform, auf der sodann die eigentliche Anwendungsintegration stattfinden kann.

Aufgrund seiner besonderen Eigenschaften soll im Weiteren auf die Realisierung von Integrationsumgebungen durch Thin–Client–Systeme [SM99] eingegangen werden.

Thin Clients eignen sich im Kontext der Integration mobiler Geräte insbesondere aus folgenden Gründen:

- Es werden keine hohen Anforderungen hinsichtlich verfügbarer Ressourcen an das Endgerät gestellt
- Auf dem Endgerät müssen zumeist keine anwendungsspezifischen Programmteile aufgespielt werden. Die Interaktion findet durch einen generischen Browser statt

- Es ergeben sich kaum Plattformabhängigkeiten, da das Endgerät fast ausschließlich zur Darstellung der Benutzungsschnittstelle verwendet wird

Insbesondere der letzte Punkt führt dazu, dass Anwendungssysteme, die auf dem Einsatz von Thin Clients basieren, leichter von einer Vielzahl verschiedener Endgeräte genutzt werden können. Da jeweils nur ein kleiner Anteil des Programmcodes angepasst werden muss, wird die Portierung auf neue Plattformen enorm vereinfacht. Es wird jedoch auch deutlich, dass es sich bei Thin Clients nicht allein um eine besondere Art von Hardwareentwurf handelt. Thin Clients werden erst durch angepasstes Softwaredesign realisierbar, das die Trennung von Anwendungslogik und Präsentationslogik überhaupt erst möglich macht.

Auf der anderen Seite stößt der Einsatz von mobilen Geräten als Thin Client Systeme insofern an Grenzen, als die Nutzung in einem solchen Szenario eine ständige Kommunikationsverbindung zum entfernten Partner voraussetzt, was einerseits zu hohem Energieverbrauch auf dem Gerät führt, was darüber hinaus jedoch gerade in mobilen Kontexten das Problem inakzeptabler Kommunikationskosten mit sich bringt.

Generell zeichnen sich Thin Client Systeme dadurch aus, dass sie die Dienstleistung einer Anwendung in mehrere Subsysteme aufspalten und diese jeweils miteinander in Beziehung setzen. In dem so zustande kommenden Schichtensystem werden üblicherweise die Haltung von Daten in einer Datenhaltungsschicht und die Verarbeitung dieser Daten in einer Applikationsschicht auf Seite des Dienstes zusammengefasst und die Visualisierung der Daten in einer Präsentationsschicht auf Seite des Dienstnehmers. Neben dieser als „Dreischicht–Architektur“ bezeichneten Grundform des mehrschichtigen Systems (siehe auch Abb. 4.1) lassen sich die Applikations– und Datenhaltungsschicht auch weiter aufspalten, wodurch man zu einem 4–, 5– und schließlich allgemein zu einem n–Schichtensystem kommt.

Neben dem bereits genannten Vorteil der geringen Ressourcenanforderungen auf der Seite der Dienstnehmer bieten mehrschichtige Systeme den weiteren Vorteil einer räumlichen Entkopplung von Anwendungslogik und Visualisierungs– bzw. Präsentationslogik. Hierdurch ergibt sich eine wesentliche Vereinfachung hinsichtlich der Anpassung auf die verschiedenen möglichen Endsysteme, von denen auf den Anwendungskern zugegriffen werden soll. In Abb. 4.1 ist dieses entsprechend skizziert: Die Anwendungslogik interagiert mit einem Mobiltelefon genauso wie mit einem herkömmlichen Arbeitsplatzrechner.

Es lässt sich also zunächst feststellen, dass sich Thin Client Systeme aufgrund ihrer mehrschichtigen Architektur für den Einsatz in Integrationsszenarien eignen, in denen potentiell ressourcenschwache Endgeräte für den Zugriff auf ein zentrales Anwendungssystem verwendet werden sollen. Der Schlüssel hierfür liegt in der Aufspaltung des Anwendungssystems in verschiedene Bereiche wie Speicherung, Verarbeitung und Darstellung von Daten, wobei im Idealfall nur letzteres auf das Endgerät ausgelagert wird. In seiner Grundform ist diese Form der Integration davon abhängig, dass eine permanente Kommunikationsverbindung zwischen dem zugreifenden Endgerät und dem Anwendungssystem besteht, da eine Darstellung ohne entsprechende Verarbeitungslogik nicht möglich ist. Dieses steht einer mobilen Nutzung zwar im Wege. Wie die folgende exemplarische Beschreibung existierender Ansätze zeigt, lässt sich die Problematik durch ei-

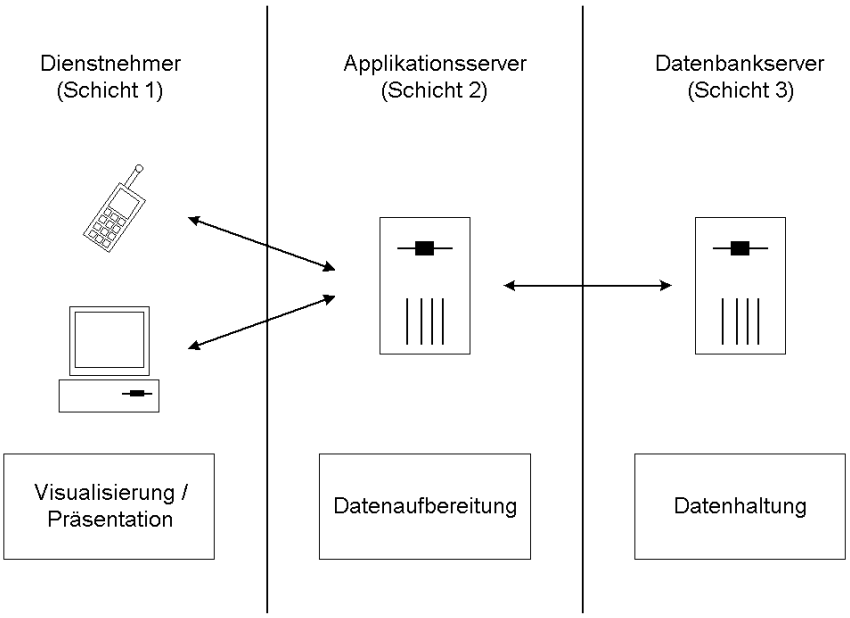


Abbildung 4.1: Aufspaltung eines Anwendungssystems in funktionale Schichten

ne geschickte Variation der Grundarchitektur jedoch abmildern, sofern dieses notwendig erscheint.

4.2 Existierende Ansätze im Überblick

Im Bereich der Thin Client Systeme lassen sich verschiedene Herangehensweisen identifizieren, nach denen die Integration von Endgeräten in ein Anwendungssystem erfolgt. Während ihnen allen gemein ist, dass sie eine Separation von Anwendungslogik und Präsentation beinhalten, weichen sie dennoch konzeptionell zum Teil stark voneinander ab. So ist zwar offensichtlich, dass auf dem Dienstnehmer eine gewisse Programmlogik vorhanden sein muss, um die Interaktion mit der Anwendungslogik auf dem Applikationsserver möglich zu machen. Gerade in deren Aufbau liegt jedoch ein wesentliches Unterscheidungsmerkmal zwischen verschiedenen Ansätzen. Auch kann nach dem Grad unterschieden werden, zu dem die Anwendungslogik auf die Abtrennung der Präsentationsschicht hin angepasst ist.

In der Literatur ist bisher keine Klassifikation möglicher Ansätze zu finden. Um die nachfolgende Gegenüberstellung verschiedener Ansätze zu vereinfachen, wird daher eine eigene Einteilung geschaffen:

Betriebssystembasierte Integration : Diese Form der Integration stellt die wohl niedrigste Abstraktionsebene in Bezug auf das Diensterbringersystem dar, auf der die Einbindung erfolgen kann. Anhand von Modifikationen am Betriebssystem des Diensterbringersystems selbst oder durch systemnahe Anwendungsprozesse wird der Zustand der Benutzungsschnittstelle des

Systems ausgelesen, in eine zur Übertragung über das Netzwerk geeignete Form gebracht und schließlich zu einem Zielsystem übertragen, wo die empfangenen Daten in ein Abbild der Benutzungsschnittstelle zurückgewandelt werden. Die Rückumwandlung bzw. Darstellung wird von einer speziellen Software übernommen, die oftmals als Viewer oder Browser des jeweiligen Ansatzes bezeichnet wird (z.B ICA Browser oder VNC Viewer). Als entfernt genutzte Anwendungen kommen bei der betriebssystembasierten Integration zumeist nicht weiter modifizierte Programme zum Einsatz, die für das jeweilige Betriebssystem entworfen wurden (z.B. MS Winword, SAP R/3 GUI o.ä.). Da der Integrationsansatz unterhalb der Anwendungsebene ansetzt, erhält die Anwendung oftmals keine Kenntnis von ihrer entfernten Nutzung.

Applikationsserverbasierte Integration : Auch bei dieser Form der Integration erfolgt die Darstellung der Präsentationsschicht durch eine spezielle Browsersoftware hindurch. Der Browser realisiert das Kommunikationsprotokoll, mit dem die Interaktion mit dem Diensterbringer erfolgt. Im Gegensatz zur betriebssystembasierten Integration stellt der Browser jedoch eine generische Umgebung zur Verfügung. Es wird kein für den Integrationsansatz spezifischer Browser verwendet, sondern eine für die Endgeräteplattform allgemein verfügbare Software, z.B. ein Web Browser oder ein WAP Browser. Auch setzt die Adaption im Diensterbringersystem nicht direkt auf der Ebene des Betriebssystems an, sondern vielmehr auf der Ebene der Anwendungen selbst. Entsprechend wird die Abstraktionsleistung auch durch das Anwendungssystem selbst erbracht. In diesem Zusammenhang wird oftmals von *Application Server* Systemen gesprochen, die abstrakt gehaltene Anwendungsdaten in eine konkrete Repräsentation umformen, welche sodann in einem Browser dargestellt werden kann. Das Betriebssystem ist nurmehr für die Bereitstellung der Daten an einer (Netzwerk–)Schnittstelle verantwortlich, über die der Browser auf die Daten zugreifen kann. Das Abstraktionsniveau ist hierbei in Richtung anwendungsnaher Schichten verschoben.

Entwurfszeit–Transformation : Ebenso wie die applikationsserverbasierte Integration setzt auch die Entwurfszeittransformation auf der Anwendungsebene an. Diese Adaptionsform orientiert sich im Wesentlichen am klassischen Anwendungsentwicklungsprozess, wobei im Gegensatz zum üblichen Vorgehen die Benutzungsschnittstelle nicht direkt in der Entwicklungsprogrammiersprache erstellt und in den Quelltext der Anwendung integriert wird. Stattdessen werden alle Aspekte der Schnittstelle in einer speziellen Beschreibungssprache abstrakt definiert und dann durch ein Übersetzungssystem in die verwendete Zielsprache übertragen, wobei eine bestimmte Zielrepräsentation vorgegeben wird. Der entstehende Quellcode wird mit dem eigentlichen Anwendungscode kombiniert und zu einem kompakten Anwendungsprogramm zusammengefügt. Das entstehende Programm kann dann entweder direkt eine Benutzungsschnittstelle für ein bestimmtes Zielsystem (bspw. PalmPilot oder Psion) besitzen und unmittelbar auf diesem System zur Ausführung kommen[1]. Wahlweise

[1] Hierbei handelt es sich dann um keine Thin Client Anwendung im eigentlichen Sinne. Andererseits kann auf diese Weise die Verarbeitung auch autonom und ohne permanente

kann jedoch auch die Ausgabe der Benutzungsschnittstelle zur Laufzeit in einer Browsersprache (z.B. HTML oder WML) erfolgen und diese durch einen Server an ein Endgerät übertragen werden, das die Schnittstelle dann in einem lokalen Browser anzeigt. In diesem Fall ähnelt der Ansatz dann der zuvor beschriebenen applikationsserverbasierten Integration. Die Entwurfszeit–Transformation wird oftmals als Ausgangspunkt für hybride Lösungen genutzt, in denen zwischen lokaler und entfernter Verarbeitung nach Notwendigkeit gewechselt werden kann.

Die Entwurfszeit–Transformation weist offensichtlich gewisse Überschneidungen mit der applikationsserverbasierten Integration auf. Tatsächlich kann die dritte Kategorie auch als grundlegende Entwurfstechnik verstanden werden, die zur Realisierung von Ansätzen aus der zweiten Kategorie Verwendung finden kann. Da die aus der Entwurfszeit–Transformation hervorgehenden Anwendungen sich jedoch oftmals deutlich vom Applikationsserver–Prinzip unterscheiden, sollen diese Ansätze in einer eigenen Kategorie zusammengefasst und beschrieben werden.

4.2.1 Betriebssystembasierte Integration

Die niedrigste Ebene, auf der eine Trennung der Anwendung von ihrer Präsentationslogik sinnvoll möglich ist, liegt auf der Ebene des Betriebssystems. Auf dieser Ebene läuft die Relokation der Präsentationsschicht und die hiermit eventuell einhergehende Transformation in eine andere Darstellungsart für die Anwendung selbst transparent ab. Stattdessen wird das Betriebssystem entsprechend modifiziert, um die grafische Ausgabe der Anwendung nicht direkt auf die im Rechnersystem vorhandene Grafikhardware zu leiten, sondern diese mit Hilfe in das Betriebssystem integrierter Bibliotheken in eine Form umzusetzen, die über ein Netzwerk zu einem entfernten Rechner weitergeleitet und dort dann in einem entsprechenden Laufzeitsystem dargestellt werden kann.

Hauptsächliche Motivation für die Entwicklung der betriebssystembasierten Integration war zunächst der Wunsch danach, Rechnersysteme mit grafischen Benutzungsschnittstellen auch in einem verteilten System benutzen zu können, ganz so, wie dieses für Terminalsysteme bereits zuvor seit langem üblich war. So wurde das X Window System (beschrieben in Abschnitt 4.2.1.1) zunächst entworfen, um einem Problem zu begegnen, das beim Übergang von Terminalsystemen zu verteilten Systemen für grafische Anwendungen auftrat: Wie sollte eine Anwendung mit grafischer Ausgabe auf einem Zentralrechner laufen, wenn die Bildschirmarbeitsplätze mit entsprechenden Grafikfähigkeiten entfernt im Netzwerk zu finden waren? Mit X11 wurde hierzu eine auf dem Client/Server Paradigma basierende Lösung gefunden.

Eine ähnliche Zielsetzung entstand später auch im Umfeld des PC Betriebssystems MS Windows. Nachdem die Rechnerleistung von Personalcomputern kontinuierlich zunahm und die Ausführung von Windows Programmen auf leistungsstarken, auf PC Hardware basierenden Zentralrechnern plötzlich denkbar wurde, musste ein Weg gefunden werden, das nur für den Einsatz auf einem Rechner bestimmte MS Windows so zu modifizieren, dass die Ausgabe einer

Netzwerkverbindung erfolgen.

Anwendung nicht nur auf dem lokalen Bildschirm, sondern auch auf entfernten Arbeitsplätzen erfolgen konnte. Einen Lösungsansatz stellte das von Microsoft als Erweiterung zu Windows NT entwickelte *Remote Desktop Protocol*(RDP) [CCM99] (siehe Abschnitt 4.2.1.2) dar, das fest in das Betriebssystem verankert wird und die gewünschte Umlenkung der Bildschirmausgabe über ein Netzwerk möglich macht.

Ansätze wie RDP (oder auch die von Citrix stammende *Independent Computing Architecture*(ICA) [GM98, CWH98]) setzen auf einer sehr niedrigen Abstraktionsebene innerhalb des Betriebssystems an. Es werden Modifikationen an den Basiselementen des Systems durchgeführt und es wird so das zuvor nur für den Einzelplatzbetrieb taugliche Betriebssystem zu einem netzwerkfähigen Mehrbenutzersystem aufgewertet.

Die Position eines „aktiven Beobachters“nimmt hingegen der Ansatz des *Virtual Network Computing* (VNC) ein (Siehe Abschnitt 4.2.1.3). Hierbei wird keine direkte Modifikation des Betriebssystems betrieben, und dennoch setzt der Ansatz unterhalb der eigentlichen Anwendungsebene an. Ein spezielles Anwendungsprogramm erstellt kontinuierlich Abbildungen der grafischen Benutzungsschnittstelle des Rechners. Die erstellten Abbildungen werden komprimiert und dann über das Netzwerk an einen Browser übertragen, wo die Bilder dekomprimiert und angezeigt werden. Auf der entgegengesetzten Strecke werden Benutzereingaben zunächst aufgefangen und dann zum Server gesandt, wo sie in die Ereignisverwaltung des Betriebssystems eingeschleust werden. Der Vorteil dieses Vorgehens gegenüber zuvor genannten Ansätzen liegt darin, dass sie auch auf Systemen zum Einsatz kommen können, die nicht speziell auf den Einsatz als mehrbenutzerfähiges Serverbetriebssystem hin optimiert wurden (z.B. Windows 3.1) und daher beispielsweise keine voreinander geschützten Laufzeitumgebungen für mehrere Anwender realisieren.

Einen sehr weit in den Bereich der Anwendungsschicht hineinreichenden Ansatz stellt schließlich das unter 4.2.1.4 dargestellte Puppeteer dar. Hierbei werden definierte Schnittstellen der Anwendungen in Richtung des Betriebssystems (OLE, COM usw.) dazu verwendet, Anwendungsprogramme in ihrer Ausführung fernzusteuern. Zwei Instanzen einer Software laufen dazu auf dem Endgerät sowie auf dem Zentralrechner, und jegliche Interaktion mit der Anwendungsinstanz auf dem Endgerät wird überwacht und in optimierter Weise an die Anwendungsinstanz auf dem Zentralrechner weitergeleitet. Soll beispielsweise ein Dokument geöffnet werden, wird der Befehl hierfür auf dem Endgerät erfasst, dieser an die zentrale Instanz geleitet und das Dokument dort geöffnet. Über die Kommunikationsverbindung zum Endgerät wird sodann der Inhalt der ersten Bildschirmseite an die entfernte Instanz zurückübertragen, alle weiteren Inhalte werden nur im Bedarfsfall nachgeladen. Auf diese Weise ergibt sich eine deutlich verbesserte Verarbeitungsgeschwindigkeit auf dem Endgerät, da nur tatsächlich benötigte Daten übertragen werden müssen.

Im Folgenden sollen die soeben kurz angerissenen Ansätze aus dem Bereich der betriebssystembasierten Integration etwas näher hinsichtlich ihrer technischen Realisierung und ihrer Vor– bzw. Nachteile beleuchtet werden.

4.2.1.1 Window System X11

Das X Window System Version 11 (kurz: X11) wurde als Ansatz entwickelt, eine grafische Schnittstelle für Unix Systeme zu schaffen. Dabei sollte der Zugriff nicht nur lokal am Rechner selbst erfolgen, sondern ähnlich wie bei klassischen Terminalsystemen von dezentraler Stelle aus. Verglichen mit zuvor üblichen Systemen sollte die Nutzung jedoch nicht nur von speziellen Sichtgeräten und über dedizierte Leitungen möglich sein. Vielmehr sollten lokale Netzwerke verwendet werden können und der Ansatz so generisch sein, dass neben dedizierten grafischen Terminals auch Unix Systeme mit lokalem Grafikbildschirm oder gar Rechner mit anderen Betriebssystemen für die Darstellung der grafischen Benutzungsschnittstelle Verwendung finden können sollten.

X11 besteht aus mehreren Komponenten. Zum Einen wird eine Programmierbibliothek (Xlib) realisiert, die zur Entwicklung von grafischen Benutzungsschnittstellen für Anwendungsprogramme zum Einsatz kommt. Diese Bibliothek realisiert die notwendigen Primitiven, um Punkte, Linien, Kreise usw. zu zeichnen und schafft so die Basis für komplexere grafische Elemente (Fenster, Menüs, Dialoge et cetera), die dann in weiteren Bibliotheken (Xt–Toolkit, Motif usw.) angeboten werden. Zum Anderen wird ein Protokoll definiert, das zum Austausch von Informationen über eine grafische Benutzungsschnittstelle zwischen Client und Server über das Netzwerk verwendet wird. Das Protokoll steht in direkter Beziehung zur Grafikbibliothek: Wurde unter Verwendung der Bibliothek eine grafische Benutzungsschnittstelle definiert, so kann die Struktur der Schnittstelle im weiteren Verlauf über das Netzwerk übermittelt werden, wobei das korrespondierende Netzwerkprotokoll zum Einsatz kommt. Gemeinsam schaffen beide Teile die Grundlage für die Entkopplung der *Generierung* der Schnittstelle von ihrer tatsächlichen *Darstellung*.

Als logischer Container für alle grafischen Ausgaben einer Anwendungssitzung gilt in X11 das *Display*. Jedes Display verfügt über einen oder mehrere *Screens*, die die tatsächliche Abbildung auf reale Bildschirme logisch abbilden, welche an jeweilige Endgeräte angeschlossen sind. Im Minimalfall besteht eine direkte Übereinstimmung zwischen Display, Screen und Bildschirm. Dieses ist dann der Fall, wenn das Endgerät über genau eine Grafikkarte verfügt und somit der logische Bildschirm (Display) genau auf einem konkreten Bildschirm (Screen) abgebildet wird. Verfügt das Terminal jedoch über mehrere Grafikkarten oder sollen Teile der Anwendungssitzung auf mehrere Bildschirme verteilt werden, so wird das Display in mehrere Screens partitioniert. Jeder Teil des Displays entspricht dann einem bestimmten Screen, der wiederum auf einen bestimmten Bildschirm ausgegeben wird. Abbildung 4.2 stellt diesen Zusammenhang schematisch dar.

Für jeden Screen muss eine spezielle Software laufen, welche über das Netzwerk Verbindung mit einem entfernten Server aufnehmen und die dort von einer Anwendung ausgehenden und nach der Übertragung über das Netzwerk eintreffenden Befehle in direkte Ausgaben auf dem zum Screen zugehörigen lokalen Bildschirm umsetzt. Die Aufteilung der Rollen in einen aktiven, auf dem Unix–Host befindlichen Part und einen reaktiven, auf dem Endgerät befindlichen Part, ist der Ursprung für das in diesem Zusammenhang verwendete, für das Client/Server Computing unübliche Benennungsschema. So wird die auf dem Endgerät (im Client/Server Sinne also dem Clientsystem) befindliche Software als X–*Server* bezeichnet. Die Anwendung hingegen, die auf dem Unix–Server

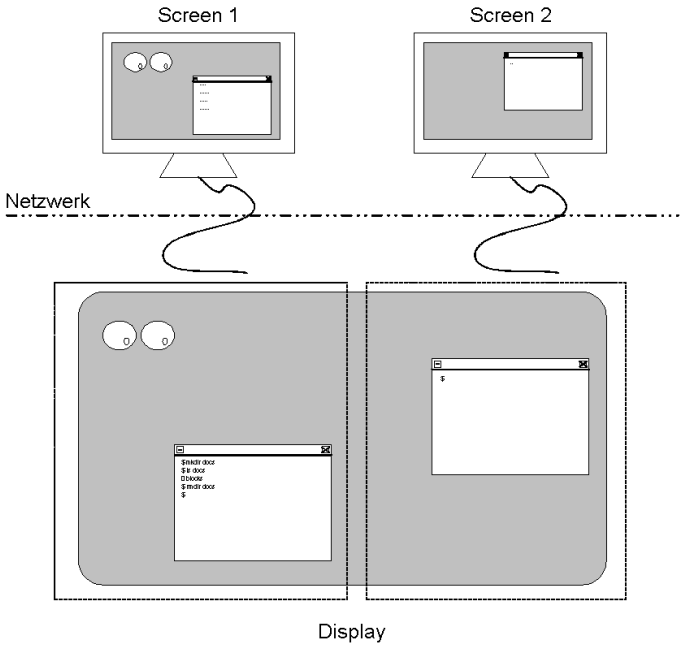


Abbildung 4.2: Zusammenhang zwischen Display und Screen im X Window System

abläuft, wird als X–*Client* bezeichnet. Wie im üblichen Fall werden auf dem Client Anweisungen erzeugt, die dann auf dem Server in Berechnungen umgesetzt werden, so werden bei X11 im X–Client Kommandos generiert, die auf dem X–Server in Ausgaben auf dem Bildschirm resultieren.

Der X–Server übernimmt im Wesentlichen folgende Aufgaben [Nye92]:

- X–Clients (d.h. Anwendungen) den Zugriff auf den lokalen Bildschirm ermöglichen
- Netzwerknachrichten des Clients verarbeiten.
- Interaktion des Benutzers aufnehmen und an hiervon betroffene X–Clients senden
- Zeichnung zweidimensionaler Grafikprimitive im Auftrag des Clients
- Komplexe Datenstrukturen verwalten, die von allen X–Clients auf dem Screen verwendet werden (Fonts, Menüs, Cursor usw.)

Dabei stellt der X–Server eine für die jeweilige Endgeräteplattform spezifische Komponente dar. Der X–Server greift direkt auf die Grafikhardware des Gerätes zu und schafft durch die Verwendung des X11 Protokolls eine Abstraktionsschicht zwischen Endgerät und entferntem Server. Die entfernte Anwendung muss nichts über die spezifischen Eigenheiten des Endgerätes wissen, genauso wie das Endgerät nichts über die der Anwendung zugrundeliegende Hardwareplattform zu wissen braucht. Es setzt lediglich die eintreffenden Befehle in

Ausgaben auf der Grafikhardware um und überträgt Benutzereingaben an die entfernte Anwendung zurück. Abbildung 4.3 stellt dieses nochmals bildlich dar.

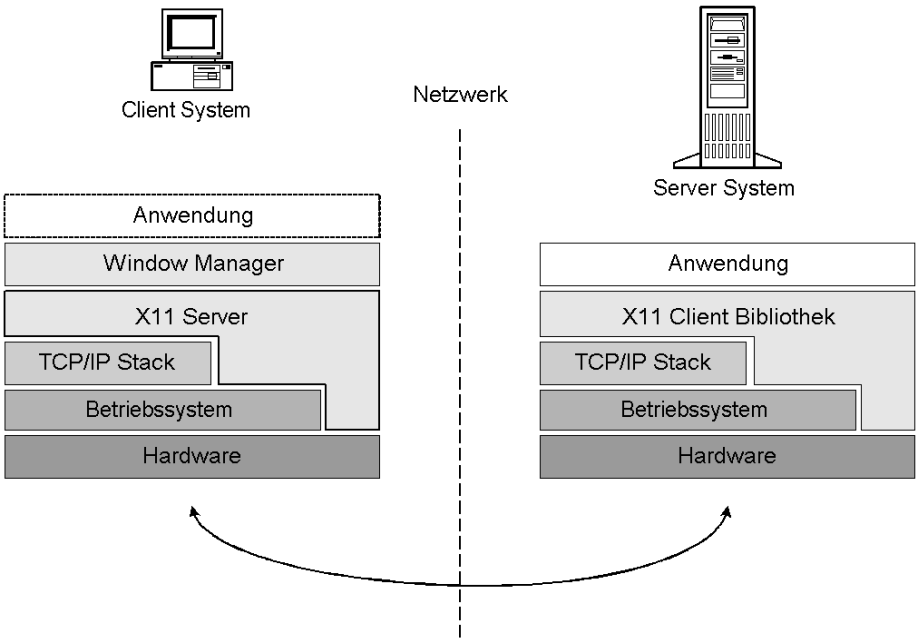


Abbildung 4.3: Schematische Darstellung Entkopplung von Client und Server durch X11

Wie aus der Abbildung ersichtlich ist, können Anwendungen direkt auf der Basisbibliothek Xlib fußen bzw. unter Verwendung einer zwischengeschalteten Bibliothek (hier Xt Toolkit) realisiert sein. Eine besondere Stellung nimmt der so genannte Window Manager ein. Der Window Manager ist eine spezielle Anwendung, die die Sitzungssteuerung eines Displays übernimmt. Er regelt den Fokus der jeweils aktiven Anwendung, organisiert die Anordnung der Fenster auf dem Screen und fügt jedem Fenster ein besonderes Systemmenü hinzu, über welches dessen Größe, Anordnung usw. abgeändert werden können. Obwohl das Konzept des Window Managers implizit für alle grafischen Systeme bekannt ist, stellt das X Window System insofern eine Besonderheit dar, als der Window Manager nicht fest mit dem System verbunden ist. Er kann vielmehr nachträglich geladen oder sogar ausgetauscht werden. Auf diese Weise kann für den jeweiligen Einsatzort (z.B. auf einem sehr eingeschränkten Mobilterminal) die optimale Darstellung erreicht werden.

Eine letzte nennenswerte Eigenheit des X Window Systems besteht darin, dass der Inhalt eines Displays ebenso auf eine ganze Anzahl von Screens aufgeteilt werden kann, wie auch verschiedene X–Clients von ganz unterschiedlichen Displays auf einem Screen zusammengefasst werden können. So kann das auf dem Bildschirm dargestellte Fenster der Buchhaltungssoftware als X–Client auf dem Unix–Host der Buchhaltung laufen, die in einem weiteren Fenster dargestellte CAD Software als X–Client auf einer leistungsfähigen Workstation in der Konstruktionsabteilung residieren, die Terminalemulation mit dem Programmeditor auf dem Unix–Host einer Partneruniversität in den Vereinigten Staaten und

schließlich (soweit es sich bei dem lokalen System um eine Unix–Workstation handelt) eine Terminalemulation in einem weiteren Fenster als X–Client auf dem lokalen Rechner laufen. Aus der Sicht des Anwenders ist dabei (außer möglicherweise hinsichtlich des jeweiligen Antwortzeitverhaltens) keine Unterscheidung möglich, wo jede einzelne Anwendung ausgeführt wird. Das X Window System ermöglicht somit aus der Sicht des Anwenders eine maximale Ortstransparenz.

4.2.1.2 Remote Desktop Protocol

Eine ähnliche Zielsetzung wie beim X Window System stand bei der Konzeption von RDP im Vordergrund. Auch in diesem Fall sollte ein entfernter Zugriff auf ein zentrales System über eine grafische Benutzungsschittstelle realisiert werden. Da RDP jedoch nicht wie X11 als Erweiterung eines Mehrbenutzersystems wie Unix, sondern aufbauend auf Microsoft Windows als typischem Einbenutzersystem entwickelt wurde, bestand neben dem bereits von X11 bekannten Problem der entfernten Darstellung einer grafischen Benutzungsschnittstelle zusätzlich die Aufgabe, den Server mit der notwendigen Mehrbenutzerfähigkeit auszurüsten. Insofern lassen sich also zwei Problemkomplexe identifizieren, die gelöst werden mussten:

- Entwurf einer Erweiterung von Windows für die entfernte Darstellung einer grafischen Benuztungsschnittstelle sowie
- Realisierung einer Mehrbenutzerfähigkeit für Microsoft Windows.

Die für den ersten Teilkomplex notwendige Systemerweiterung und das verwendete Kommunikationsprotokoll werden üblicherweise unter dem Begriff des *Remote Desktop Protocol* zusammengefasst. Das hierauf aufbauende mehrbenutzerfähige Microsoft Windows NT wurde hingegen unter dem Namen Windows NT Terminal Server Edition (TSE) von Microsoft kreiert und als eigenständiges Produkt in die Produktpalette von Windows NT eingefügt. Gemäß des Schwerpunkts dieser Arbeit soll im Weiteren nicht näher auf TSE eingegangen werden, sondern ausschließlich RDP als ein mit X11 vergleichbarer Ansatz betrachtet werden.

RDP hat zum Ziel, Anwendungen, die unter MS Windows auf einem zentralen Rechnerknoten laufen, auf entfernten Arbeitsplätzen verfügbar zu machen. Ein wichtiges Merkmal von RDP ist dabei, dass die Anwendung nicht explizit für den Einsatz in einer Terminal Server Umgebung entworfen sein muss. Quasi jede gültige Windowsanwendung[2] kann via RDP auf einem entfernten Rechner benutzt werden, der über einen RDP Viewer verfügt. Wie der Name bereits impliziert, stand bei RDP zunächst lediglich die Erweiterung des Windows Desktop auf entfernte Maschinen im Vordergrund, während die Plattformunabhängigkeit von X11 nicht gesucht wurde. Inzwischen existieren jedoch auch RDP Browser für andere Betriebssysteme, so dass der Zugriff mittlerweile nicht mehr nur von Windows PCs möglich ist, sondern im Prinzip von einem beliebigen Rechnerarbeitsplatz erfolgen kann, der über eine Netzwerkstrecke mit

[2]Gültig bedeutet in diesem Zusammenhang, dass die Bildschirmausgabe unter Verwendung von Aufrufen der Windows Grafik–API erfolgt und nicht durch direkte Zugriffe auf die Grafikhardware.

dem zentralen Terminal Server verbunden ist, ungeachtet seiner Hardware bzw. seines Betriebssystems.

Das RDP Protokoll basiert im Wesentlichen auf T.120, einer von der ITU standardisierten Familie von generischen Kommunikationsprotokollen, die zur verteilten Nutzung von Anwendungen entworfen wurden. Während T.120 derzeit insbesondere als Basis des weitverbreiteten Kollaborationssystems Netmeeting bekannt ist, stellen *Shared–Editing–* und Videokonferenzsysteme nur einen Ausschnitt des Leistungsspektrums von T.120 dar. Folgende wichtige Merkmale von T.120 lassen sich erkennen [Dat97]:

- Punkt–zu–Punkt sowie Punkt–zu–Multipunkt (Multicast) Transport
- Verlässlicher Transport
- Plattformunabhängigkeit
- Netzwerktransparenz
- Netzwerkunabhängigkeit
- Interoperabilität
- Skalierbarkeit
- Anwendungsunabhängigkeit

T.120 ist dabei lediglich der Sammelbegriff für den gesamten Komplex an Protokollen. So definiert z.B. T.121 das Basiskommunikationsprotokoll und T.122/T.125 das Punkt–zu–Multipunkt Protokoll (Multipoint Communications Services, MCS). T.124 beschreibt eine generische Sitzungssteuerung (Generic Conference Control, GCC), T.126 realisiert den Austausch von Bilddaten (*Still Image Exchange*) usw. Abbildung 4.4 zeigt den prinzipiellen Aufbau der T.120 Protokollfamilie.

Während Netmeeting direkt auf dem ITU Standard aufbaut, wurde dieser für den Einsatz als Basis für RDP massiv optimiert und den speziellen Anforderungen des Anwendungsfalls angepasst. So wurde eine extrem leichtgewichtige *Generic Conference Control* (GCC) realisiert. Auch wurde eine spezifische Multipoint–Kommunikationsschnittstelle (MCS) realisiert, die direkt auf Kernelebene die Daten einer Sitzung aufbereiten kann. Schließlich wurde eine direkte Schnittstelle zur Grafikbibliothek von Windows NT, Win32 GDI, geschaffen, über die eine direkte Interpretation und Kompression von GDI Nachrichten möglich ist. Zu übertragende Grafiken, Bilder oder Symbole aus einer Windows Sitzung werden in verschiedenen Auflösungen aufbereitet, um jederzeit auf Schwankungen der Übertragungskapazität des Netzwerkes reagieren zu können und die Qualität der übertragenen Daten ohne größere Verzögerungen anpassen zu können.

In MS Windows zeichnet das GDI als Schnittstelle zur Anwendungsschicht für die Darstellung von Grafikprimitiven und für deren Kombination zu typischen Komponenten der Windows Oberfläche (Mauszeiger, Texte, Menüs, Dialoge usw.) verantwortlich. GDI bietet hierzu eine standardisierte Menge von Operationen

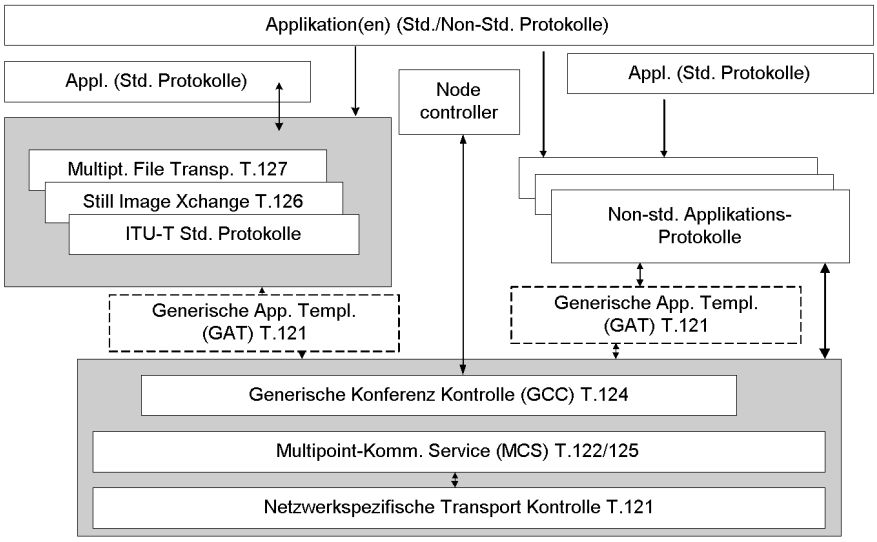


Abbildung 4.4: Schematischer Aufbau des T.120 Multiprotokoll Standards der ITU (nach [Dat97])

über eine gemeinsame Schnittstelle an, welche von der Anwendung nach Belieben genutzt werden können. Korrespondierend zur GDI wird in jedem Windows System ein Grafikkartentreiber installiert, der wiederum eine standardisierte Schnittstelle (sehr einfacher Operationen) für GDI selbst anbietet und auf der anderen Seite die hardwarespezifische Steuerung der Grafikkarte übernimmt. Übergibt die Anwendung also über einen Aufruf an die GDI Schnittstelle das Kommando zur Darstellung eines Dialogfensters an das Betriebssystem, so wird dieses von GDI in eine Vielzahl von Aufrufen an den Grafikkartentreiber umgesetzt, der wiederum die Register der Karte entsprechend programmiert, um die gewünschten Pixel darzustellen.

RDP ermöglicht nun die entfernte Darstellung einer Windows Anwendung, indem der Grafikkartentreiber ausgetauscht wird gegen den so genannten *RDP Display Driver* (RDPDD). Eingehende Aufrufe von GDI werden zunächst in RDP Protokollnachrichten umgesetzt und diese dann anhand von Kompressions– und Cachingalgorithmen derart optimiert, dass nur ein Minimum an Daten an den Klienten übertragen werden muss. Die so entstehenden Zeichenbefehle werden an den *RDP WinStation Driver* (RDPWD) übermittelt und von diesem in Nachrichten zerlegt, die über den *TCP/IP Transport Driver* (TDTCP) an den Klienten übertragen werden. Dort werden sie an den RDP Browser weitergeleitet und von diesem unter erneuter Nutzung etwaiger Optimierungsstrategien auf dem Bildschirm angezeigt. In entgegengesetzter Richtung können Eingaben des Anwenders durch den RDP Browser aufgefangen, in entsprechende RDP Nachrichten umgesetzt und an den Server zurückübermittelt werden, wo sie über den Umweg der genannten Module an die Anwendung übergeben werden, wie dieses auch in einem herkömmlichen Windows System der Fall wäre.

Folgende Algorithmen stellen eine kleine Auswahl der Möglichkeiten dar, die von RDP zur Optimierung des Kommunikationskanals verwendet werden:

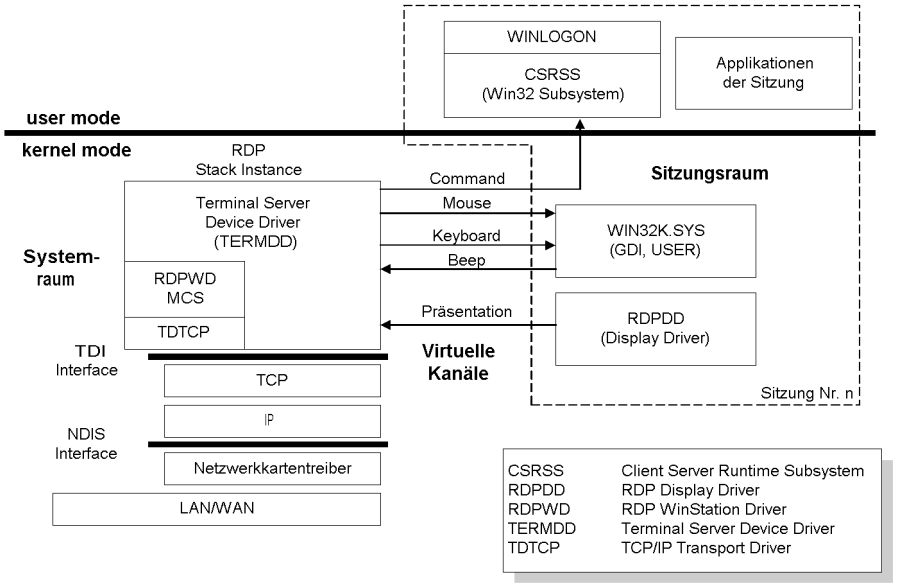


Abbildung 4.5: Funktionale Komponenten von RDP in MS Windows NT4, Terminal Server Edition (nach [CCM99])

- Kommt von GDI der Befehl einen Text auf dem Bildschirm darzustellen, so werden die darzustellenden Zeichen nicht Pixel für Pixel übertragen, sondern diese vom RDP Browser auf dem Klienten nach der ersten Übertragung einer jeden Zeichenrepräsentation zwischengespeichert. Bei jeder weiteren Übertragung wird dann lediglich noch die gewünschte Position des Zeichens übermittelt, während die eigentliche Darstellung des Zeichens als Kopiervorgang aus dem lokalen Speicher des Endgerätes erfolgt.

- Soll in einem Textfenster zeilenweise geblättert werden, so wird nicht der gesamte Fensterinhalt pixelweise neu übertragen, sondern auf dem Klienten jeweils zeilenweise zwischengespeichert. Beim erneuten Aufbau des Fensterinhalts werden dann vom Server lediglich die neuen Positionen der Zeilen übermittelt, die sodann aus dem Cache des RDP Browsers dargestellt werden können.

- Soll ein Rechteck einer bestimmten Farbe und Größe dargestellt werden, so werden lediglich die Koordinaten, Ausmaße und die gewünschte Farbe übertragen, während die eigentliche Darstellung des Rechtecks (also die Berechnung der Positionen hierfür notwendiger Pixel) durch den RDP Browser selbst erfolgt.

- Wird ein Fenster auf dem Bildschirm verschoben, so werden vom RDP Server lediglich die Ausmaße des Fensters, seine ursprüngliche Position und die Koodinaten des verschobenen Fensters übermittelt. Der hieraus resultierende Kopiervorgang von Pixeln wird dann auf dem Endgerät durch den dort residierenden RDP Browser selbst vorgenommen. Ausschließlich diejenigen Inhalte müssen dabei über das Netzwerk übertragen werden, die zuvor durch das Fenster verdeckt waren und nun sichtbar sind – für

sie kann natürlich nicht immer ein Caching auf dem Endgerät erfolgen, sofern sich der Inhalt geändert hat, während er vom verschobenen Fenster verdeckt war.

Wie erkennbar ist, profitieren die Optimierungsalgorithmen dabei von der Tatsache, dass sich aus dem Zusammenspiel zwischen RDP und GDI gewisse semantische Informationen ergeben, die von RDP zwecks Optimierung verwendet werden können. So „weiß“ der RDPDD, welche Pixel gemeinsam ein Zeichen und welche Zeichen eine Zeile formen. Entsprechend kann dann auch das Caching stattfinden. Hierin unterscheidet sich RDP (und auf ähnliche Weise auch ICA) vom im folgenden Abschnitt beschriebenen VNC, wo die Abbildung des Anwendungsbildschirms nicht auf der Ebene der Betriebssystembibliothek erfolgt, sondern auf einer höheren Ebene, auf der lediglich noch analysiert werden kann, welche Pixel sich zwischen zwei Übertragungen zwischen Client und Server geändert haben. Hierin zeigt sich darüber hinaus das generelle Problem einer anwendungstransparenten Integration, die auf semantische Informationen über dargestellte Inhalte verzichten muss und sie daher nicht zur Optimierung des Darstellungsvorgangs heranziehen kann.

Integrationsansätze wie RDP oder auch ICA basieren auf einer relativ komplizieren Erweiterung des Betriebssystemkerns. Sie setzen eine direkte Modifikation des Systems voraus, die mit Anpassungen auf Quelltextebene einhergeht. Insofern können sie nur durch den Hersteller des Betriebssystems bzw. durch dessen Lizenzpartner durchgeführt werden. Aufgrund des entstehenden Umsetzungsaufwands stehen sie nur für wenige (Server–) Betriebssystemumgebungen zur Verfügung, für die sich dieser Aufwand aus der Sicht des Anbieters als lohnend erweist. Auch sind die verwendeten Protokolle und Optimierungsalgorithmen meist vergleichsweise aufwendig. Die verwendeten Algorithmen sind oftmals proprietär und nicht im Detail dokumentiert. Entsprechend existieren die auf der Seite des Endgerätes notwendigen Browser nur für bestimmte Plattformen, was die gewünschte (und konzeptionell mögliche) Plattformunabhängigkeit teilweise erheblich einschränkt. Aus diesem Grunde, und auch um die entfernte Nutzung einfacher Arbeitsplätze möglich zu machen, wurde das Konzept des *Virtual Network Computing* (VNC) erarbeitet, das im nächsten Abschnitt beschrieben wird.

4.2.1.3 Virtual Network Computing

Bei VNC handelt es sich um einen extrem leichtgewichtigen Ansatz zur entfernten Nutzung von Anwendungen in einem verteilten System, der von seinem Abstraktionsniveau auf der Ebene des Betriebssystems angesiedelt werden kann, der jedoch nicht auf direkten Modifikationen des Betriebssystems selbst basiert. VNC wurde im Rahmen der Arbeiten zum Projekt *VideoTile* [Add96] entworfen, in dem am Olivetti Research Laboratory (ORL) ein so genannter Ultra Thin Client entwickelt wurde, ein Gerät also, bei dem ausschließlich die Darstellung von Bildschirminhalten auf dem Endgerät stattfindet, während sämtliche Anwendungsprozesse auf einem Zentralrechner ablaufen[3].

[3] Ähnliche Ansätze wurden u.A. auch in den Projekten *Infopad* [BBB+94, Bro99] sowie *SciencePad* [JWDH95] verfolgt.

Bei VNC residiert im Anwendungsraum des Servers eine Applikation, die jederzeit den Inhalt des Bildschirms erfassen und über das Netzwerk zur Verfügung stellen kann. Auf dem Endgerät läuft ein korrespondierender Browser, der mit der Serverapplikation in Kontakt treten und mit dieser dann Informationen über den Bildschirminhalt des Servers austauschen kann.

Meldet sich ein Klient bei dieser Applikation an, so werden zunächst Parameter wie die Desktopgröße, das Pixelformat oder die zur Verfügung stehende Netzwerkbandbreite zwischen Klient und Server ausgehandelt. Der Server entscheidet sodann auf der Basis dieser Parameter über die Optimierungsstrategie, die beim Austausch von Informationen mit dem Klienten zu wählen ist. Hiernach beginnt er dann, jeweils nach Anforderung durch den Klienten (d.h. *demand–driven*) den Inhalt des Bildschirms zu kopieren, diesen entsprechend der gewählten Optimierungsstrategie zu komprimieren und dann an den Klienten zu übertragen. Der Klient dekomprimiert die empfangenen Daten entsprechend und stellt diese dann auf dem Bildschirm dar. Eventuelle Eingaben durch den Anwender über Tastatur oder Maus werden vom Klienten an die Serveranwendung übertragen und von dieser in die Ereignisverarbeitung des Servers eingefügt. Im Gegensatz zu X11, RDP oder auch ICA ist hierbei zu beachten, dass Ereignisse *uninterpretiert* an den Server zurückgegeben werden. Das bedeutet zum Beispiel, dass Mausklicks nicht einem bestimmten Fenster zugeordnet werden, das durch den VNC Browser dargestellt wird, sondern lediglich als ein Klick an einer bestimmten Position relativ zum mit dem Server vereinbarten Nullpunkt vermeldet werden können. Dieses wird dadurch bedingt, dass VNC kein Wissen über die Struktur des vom Server übernommenen Anwendungsbildschirms erfasst und an den Klienten überträgt. Es werden lediglich unstrukturierte Bilddaten übergeben, die vom Klienten nicht in Fenster oder andere Strukturen zerlegt werden können.

Entsprechend fallen auch die möglichen Optimierungen durch die Serverapplikation weniger vielfältig aus, als dieses etwa bei RDP der Fall ist. Da keine Informationen über die Struktur des erfassten Bildschirminhalts herangezogen werden können, sind ausschließlich Algorithmen möglich, die sich auf die Kompression von Bilddaten beziehen und auf die Zwischenspeicherung von Daten durch den Browser. Möglich sind hierbei z.B. die folgenden in [RSFWH98] beschriebenen Verfahren:

- Übertragung rechteckiger Ausschnitte an Stelle des gesamten Bildschirminhalts bei begrenzten Änderungen in einem bestimmten Bereich des Serverbildschirms
- Übertragung von JPEG Differenzbildern bei über den gesamten Serverbildschirm verteilten, jedoch auf kleinere Bereiche eingrenzbare Änderungen
- Vergleich von zu übertragenen Bildschirminhalten mit zuvor übertragenen Inhalten und Reduktion auf bloße Kopierbefehle, die sodann auf dem Zwischenspeicher des VNC Browsers auf dem Endgerät ausgeführt werden können

Während dieses Vorgehen durch das aufwendige Analyseverfahren in einer vergleichsweise hohen Last auf dem Server resultiert (vgl. [CCM99]), ergibt sich ein

Vorteil gegenüber zuvor beschriebenen Verfahren darin, dass VNC als vollkommen zustandslos anzusehen ist[4]. Kommt es im Zuge der Interaktion zwischen Klient und Server zu einem Verbindungsabbruch, so wird die Anwendungssitzung hiervon nicht beeinträchtigt. Nachdem der Klient (je nach Implementation ohne weitere Intervention durch den Anwender) eine erneute Verbindung mit dem VNC Server aufgebaut hat, wird lediglich der Übertragungszyklus erneut in Gang gesetzt und dann mit der Erfassung und Weiterleitung von Bildschirminhalten fortgefahren. Diese in der Literatur als *Teleporting* bezeichnete Eigenschaft (vgl. [BRH94] bzw. [WRB+96]) von VNC macht es insbesondere in Einsatzkontexten interessant, in denen Kommunikationsunterbrechungen oftmals vorkommen. Reißt beispielsweise bei der Nutzung von VNC über eine drahtlose Netzwerkstrecke kurzzeitig die Verbindung zum Server ab, so wird dieses die entfernte Nutzung einer Anwendung nicht weiter stören, sofern die Unterbrechung nur kurz andauert.

Aufgrund der recht einfachen Struktur von VNC werden auf Seiten des Klienten nur sehr geringe Anforderungen an verfügbare Ressourcen gestellt. Mit einer Größe von lediglich 150 Kilobytes ist z.B. der Klient für Windows extrem klein. Auch diesbezüglich eignet sich VNC also für den Einsatz auf mobilen Plattformen. Schließlich wurde das komplette Protokoll einschließlich einer Referenzimplementation dokumentiert und veröffentlicht, so dass eine Adaption auf weitere Plattformen jederzeit möglich ist.

VNC bietet, ähnlich wie RDP, nur die Lösung zu einem der im Zusammenhang mit der entfernten Nutzung von Anwendungen auftretenden Probleme an. Es ermöglicht die entfernte Darstellung von Anwendungsschnittstellen sowie die Rückübertragung von Interaktionen an die Anwendung. Nicht gelöst ist hingegen die Frage der Mehrbenutzerfähigkeit des zugrundeliegenden Betriebssystems. Während diese für Windows NT durch die Terminal Server Edition beantwortet wird, existieren im Zusammenhang mit VNC keine entsprechenden Lösungen. Ein wirklicher Betrieb von VNC in einer Mehrbenutzerumgebung ist enstprechend nur auf der Basis von Unix sinnvoll, wo die Mehrbenutzerfähigkeit bereits durch das Betriebssystem selbst realisiert ist (Vgl. [RBMH94]). Für Windows und alle anderen Einzelplatzbetriebssysteme ist hingegen nur der Einsatz in Szenarien denkbar, bei denen ausschließlich der entfernte Zugriff auf den Rechner durch *einen* bestimmten Benutzer stattfindet. Dieses kann etwa bei der entfernten Betreuung eines Windows Fileservers durch einen Administrator der Fall sein oder beim Zugriff auf den persönlichen PC durch einen Anwender während einer Geschäftsreise. Da VNC jedoch nicht in das Betriebssystem direkt integriert ist, sondern lediglich als Anwendungsprozess abläuft, der über offene, wohldokumentierte Schnittstellen mit dem Betriebssystem interagiert, lässt sich die Adaption auf eine neue Systemplattform mit nur wenig Aufwand realisieren, während dieses Unterfangen im von RDP oder ICA gewählten Ansatz nur mit Detailwissen über das zu modifizierende System angegangen werden kann. Und selbst in diesem Fall ist der Zugriff auf den Quellcode des Betriebssystems zu verlangen, was nur in seltenen Fällen überhaupt möglich sein dürfte.

Schließlich gilt für VNC, genauso wie für die zuvor beschriebenen Ansätze, dass sich diese nur dann eignen, wenn folgende Anforderungen durch das für den

[4] Die einzige Einschränkung hierzu stellen differentielle Optimierungsverfahren dar, bei denen sich die zu übertragenen Daten aus vorherigen Übertragungen ergeben.

Zugriff zu verwendende Endgerät als erfüllt zu gelten haben:

1. Das Endgerät besitzt eine „offene“ Architektur, welche die Erweiterung durch die im Zusammenhang mit dem gewählten Integrationsansatz notwendige Browsersoftware erlaubt

2. Das Endgerät verfügt über genügende grafische Fähigkeiten, um die Darstellung des Anwendungsbildschirms auf dem Endgerät zu ermöglichen

Beide Punkte werden durch die meisten derzeit auf dem Markt verfügbaren PDAs erfüllt. ICA Browser existieren für eine Vielzahl von Endgeräten inklusive Mobilgeräten mit den Betriebssystemen Windows CE bzw. Symbian EPOC. Ähnliches gilt für RDP, wobei hier die oben beschriebenen (und überwiegend markstrategisch bedingten) Einschränkungen hinsichtlich der Verfügbarkeit auf Nicht–Windows–Plattformen gelten. Die Nutzung von Mobiltelefonen auf der Basis von WAP oder iMode wird hingegen von keinem der beschriebenen Ansätze unterstützt. Während derartige Geräte mittlerweile durchaus genügende Grafikfähigkeiten besitzen, ist in diesem Fall vor allem die mangelnde Offenheit bzw. Flexibilität des verwendeten Systems ausschlaggebender Grund.

Soll von solchen Geräten ein Zugriff auf entfernte Anwendungen erfolgen, so sind betriebssystembasierte Integratsionsstrategien bis auf weiteres ausgeschlossen. An ihre Stelle treten die im folgenden Abschnitt beschriebenen applikationsserverbasierten Verfahren. Hierbei wird nicht versucht, eine getreue Kopie des Serverbildschirms an den Klienten zu übertragen, sondern eine für das betreffende Endgerät passende Aufbereitung auf der Basis der zu übertragenden Informationen zu finden. Die Integration erfolgt hierbei also nicht mehr auf der Basis des Betriebssystems, sondern auf Ebene der Anwendungen.

4.2.1.4 Puppeteer

Der von de Lara et al. in [dLWZ00] für Puppeteer gewählte Ansatz weicht von den bislang geschilderten Varianten der betriebssystembasierten Integration deutlich ab. In diesem Fall wird nicht versucht, direkt in das Betriebssystem einzugreifen, um die Adaption zu ermöglichen, wie dieses bei RDP oder ICA der Fall ist. Auch wird nicht wie bei VNC die Position eines „Beobachters“ eingenommen, der den Zustand der Benutzungsschnittstelle betrachtet und diesen an einen entfernten Browser übergibt. Puppeteer verschiebt gewissermaßen das Schwergewicht weiter vom Betriebssystem hin zur Anwendung und stellt somit eine Zwischenstufe dar, welche noch oberhalb von VNC ansetzt, die jedoch nicht wirklich der Entwurfszeittransformation zuzurechnen ist, da keine Modifikation der eigentlichen Anwendung erfolgt. Auch wird kein generischer Browser verwendet, sondern die Anwendung selbst quasi als eine Art Browser ihrer eigenen Daten verwendet.

Einer Nutzung der Anwendungen in mobilen Kontexten steht oftmals die Größe der verwendeten Dokumente entgegen, die sich nicht über langsame Kommunikationsverbindungen übertragen lassen. Soll beispielsweise eine Präsentationsdatei auf dem Mobilgerät bearbeitet werden, so muss dafür die gesamte Datei auf das Mobilgerät übertragen werden, bevor eine Bearbeitung möglich wird. Angesichts

von Dokumentengrößen von mehreren Megabytes ist dieses nicht praktikabel. Der Kerngedanke bei Puppeteer beruht nun auf der Beobachtung, dass eine zunehmende Menge von Anwendungen sich über externe Schnittstellen steuern lassen. Konkret können Anwendungen wie MS Powerpoint oder MS Internet Explorer mittels COM bzw. OLE extern zum Laden von Dokumenten, zum Navigieren durch diese Dokumente, zur Ausführung von Anwendungsfunktionen usw. veranlasst werden. Darüber hinaus sind mehr und mehr Anwendungen komponentenbasiert: Die von ihnen verarbeiteten Dokumente sind nicht monolithisch, sondern aus einzelnen Objekten aufgebaut, die miteinander in einer für das Anwendungsprogramm sinnvollen Weise verkettet sind. Puppeteer macht sich nun die externe Steuerbarkeit der Anwendungen sowie den komponentenbasierten Aufbau der Dokumente zunutze, um für die Bearbeitung von Dokumenten die Anwendung selbst als Browser verwenden zu können, wobei große Dateien nicht länger komplett auf das Endgerät übertragen werden müssen.

Hierzu werden auf dem Endgerät und im Festnetz zwei Stellvertreter etabliert. Der auf dem Endgerät befindliche Stellvertreter (sog. Puppeteer–Proxy) dient der Steuerung und Überwachung der vom Benutzer zur Bearbeitung des Dokuments verwendeten Anwendung, während der Stellvertreter im Festnetz zur Steuerung einer zweiten Instanz der Anwendung dient, die wiederum zur Verwaltung des eigentlichen Dokuments dient. Aufgrund seiner serverseitigen Anordnung wird dieser Stellvertreter auch als Puppeteer–Server bezeichnet. Sowohl Puppeteer–Proxy als auch Puppeteer–Server kommunizieren mit der jeweiligen Anwendungsinstanz über eine von der Anwendung selbst angebotene Schnittstelle, die als *Document Manipulation Interface* (DMI) bezeichnet wird. Konzeptionell wird von dieser Schnittstelle verlangt, dass hierüber Daten in die Anwendung geladen werden können, dass in den Anwendungsdaten navigiert werden kann und diese auch modifiziert werden können. In der konkreten Implementation wird als DMI zur Steuerung von MS Powerpoint und MS Internet Explorer auf OLE und COM zurückgegriffen. In einem anderen Fall könnte die Schnittstelle jedoch auch auf CORBA, RMI o.ä. basieren, je nachdem, wie die Anwendung extern steuerbar ist.

Proxy und Server realisieren jeweils eine Art Treiber, der die Anwendung steuert. Der Treiber wird in die Dokumentenschnittstelle der Anwendung integriert und registriert sich – soweit vorhanden – auch bei der Ereignisverwaltung der Anwendung. Der Treiber auf der Endgeräteseite muss erkennen können, welche Komponenten gerade vom Anwender betrachtet werden und muss ggf. Speicherungs– und Ladevorgänge anfangen können. Hieraus werden dann Befehle generiert, die an den Treiber auf der Serverseite weitergegeben werden. Dieser muss sodann auf geeignete Weise die Anwendung dazu instruieren die gewünschten Komponenten in den Speicher zu holen und über die DMI der Anwendung die Daten der benötigten Komponente extrahieren. Diese Daten werden an den Endgerätetreiber zurückübertragen und auf dessen Seite über die DMI in den Kontext der klientseitigen Anwendung eingefügt. Im Zuge der Übertragung kann das Treiberpaar je nach Qualität der Netzwerkverbindung eine Anpassung der zu übertragenden Komponenten veranlassen und so die Übertragungsgeschwindigkeit optimieren.

Abbildung 4.6 zeigt die schematische Anordnung der funktionalen Komponenten bei Puppeteer. Alternativ zu der gezeigten Variante, bei der auf der Serverseite ein Puppeteer–Server eine zweite Instanz der Anwendung steuert, ist es auch

denkbar, dass der Puppeteer–Server direkt auf den Dateiserver zugreift. Dieses ist dann möglich, wenn die Dokumentenstruktur der Anwendung bekannt ist und die vom Puppeteer–Proxy eingehenden Anforderungen vom Server direkt in Zugriffe auf das bearbeitete Dokument umgesetzt werden können. Auf diese Weise lässt sich die Geschwindigkeit des Gesamtsystems weiter erhöhen und die Ressourcenlast des Servers verringern. In der gezeigten Variante fungiert die Anwendung gleichsam als Parser, der die Zugriffe auf Komponenten des Dokumentes puffert und in tatsächliche Operationen auf Dateiebene umsetzt.

Schließlich werden Policies dazu eingesetzt, das Verfahren weiter zu verfeinern. So kann anhand einer Policy darüber entschieden werden, inwieweit durch vorgezogenes Laden möglicherweise benötigter Daten (sog. Prefetching) entstehende Pausen in der Datenübertragung genutzt werden sollen oder welche Komponenten zu Beginn einer Sitzung vom Server–Proxy übertragen werden sollen. Im Beispiel der Präsentationssoftware könnte dieses etwa neben der ersten Folie auch ein Überblick über alle Folien sein.

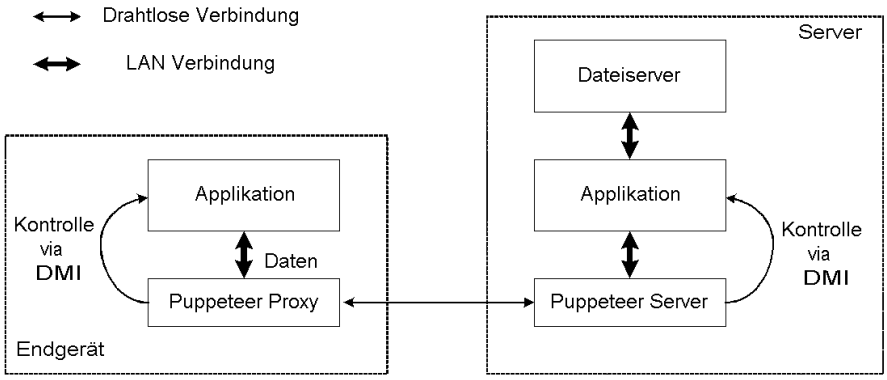


Abbildung 4.6: Schematische Anordnung der funktionalen Komponenten bei Puppeteer (nach [dLWZ00])

Die eigentliche Anwendung wird bei Puppeteer nicht angetastet. Alle Steuerungs– und Überwachungsmechanismen werden in den beiden Stellvertretern zusammengefasst, die ihrerseits über das Document Manipulation Interface die Anwendung kontrollieren. Auf der Serverseite handelt es sich, ähnlich wie bei VNC, um einen Anwendungsprozess, der als eine Art Beobachter über die Anwendung wacht, ihren Zustand ausliest oder diesen ändert. Dabei ist Puppeteer auf eine bestimmte Anwendung beschränkt und überwacht nicht die gesamte Betriebssystemoberfläche. Im Gegensatz zu VNC oder auch RDP wird auf der Endgeräteseite kein in sich geschlossener Browser verwendet. Vielmehr tritt die Anwendung selbst als Browser auf. Vom Puppeteer–Server eintreffende Befehle werden vom Proxy auf die Anwendung übertragen und von dieser angezeigt. Aus der Tatsache, dass die Anwendung selbst letztlich als eine Art Viewer auftritt, ergibt sich auch die endgültige Berechtigung dafür, Puppeteer in eine Reihe mit den zuvor beschriebenen Ansätzen zu stellen. Gewissermaßen wird so ein Thin Client System auf höherer Ebene erzeugt.

De Lara berichtet in [dLWZ00], dass sich mit dem gewählten Ansatz deutliche Verbesserungen in der Bearbeitungsgeschwindigkeit gegenüber einer Übertra-

gung vollständiger Dokumente ergeben. So lässt sich anhand von Messungen nachweisen, dass sich die Latenz bei der Bearbeitung von Powerpoint Dateien über eine Übertragungsstrecke mit einer Kapazität von 384 Kilobit pro Sekunde um über 84 Prozent verringert, für die Darstellung von HTML Dokumenten über eine 56Kbit Leitung immerhin noch um über 76 Prozent. Andererseits setzt das hinter Puppeteer verborgene Konzept voraus, dass die Anwendung auf dem Endgerät verfügbar ist, eine geeignete Bearbeitungsschnittstelle anbietet und das Endgerät neben den notwendigen Ressourcen für die Anwendung selbst zusätzlich auch diejenigen für den benötigten Proxy bereitstellen kann. Somit eignet sich dieser Ansatz also nur dort, wo die Einschränkungen des mobilen Umfelds sich nur auf die (schwache bzw. unzuverlässige) Kommunikationsverbindung beziehen, nicht jedoch auf das mobile Endgerät selbst. Hier ist ein leistungsstarker Laptop vorauszusetzen.

4.2.2 Applikationsserverbasierte Integration

Die applikationsserverbasierte Integration stellt im Prinzip eine Spezialisierung der zuvor beschriebenen betriebssystembasierten Integration dar. Hier wie dort wird ein Browser zur Anzeige der Benutzungsschnittstelle verwendet, wobei bei der applikationsserverbasierten Integration nicht die Betriebssystemebene gewählt wird, um die Trennung der Anwendung von ihrer Präsentationsschicht einzuleiten, sondern auf der Ebene der Anwendung selbst angesetzt wird — konzeptionell also auf einem höheren Abstraktionsniveau.

Applikationsserverbasierte Integration wird insbesondere dort favorisiert, wo mittels einer Vielzahl verschiedener Endgeräte auf eine gemeinsame Datenbasis zugegriffen werden soll. Die Aufbereitung der Daten erfolgt dabei im Anwendungssystem selbst, während der Browser lediglich für die Visualisierung der Daten verwendet wird. Daher eignet sich die applikationsserverbasierte Integration vor allem dort, wo geringe Ressourcen die Datenhaltung und –aufbereitung auf dem Endgerät selbst unmöglich machen, wie es bei mobilen Geräten ja zumeist der Fall ist. Auch ist diese Herangehensweise dort vorteilhaft, wo das Gerät die nachträgliche Erweiterung um einen proprietären Browser nicht erlaubt (sog. „geschlossenes“ Design), aber dennoch, wie beispielsweise beim WAP Telefon, bestimmte Darstellungsfähigkeiten besitzt, welche die Datenverarbeitung in begrenztem Maße ermöglichen.

Applikationsserverbasierte Integration zeichnet sich dadurch aus, dass dem Ansatz eine mindestens dreischichtige Architektur zugrunde liegt. Zwischen Endgerät auf der einen Seite und Datenhaltung auf der anderen Seite existiert eine Zwischenschicht, in der die Daten in eine vom Endgerät darstellbare Form gebracht und in dieser gerätespezifischen Form dann zum Endgerät transportiert werden können. Neben dieser Grundform können durch weitere Aufspaltung der mittleren Schicht beliebige weitere Varianten realisiert werden, in denen die Aufbereitung der Daten für das Endgerät über mehrere Teilschritte erfolgt. Je mehr Zwischenrepräsentationen existieren, desto geringer sind die direkten Abhängigkeiten zwischen der Zielrepräsentation auf der einen Seite und der Datenhaltung auf der anderen Seite.

Üblicherweise wird bei der applikationsserverbasierten Integration ein generisches Anzeigeprogramm verwendet. Generisch bedeutet in diesem Zusammen-

hang, dass es nicht ausschließlich für einen bestimmten Anwendungsfall konzipiert wurde. Vielmehr ist es so universell ausgelegt, dass es zur Anzeige nahezu beliebiger Daten verwendet werden kann, also auch Daten, die bei der Entwicklung des Anzeigeprogramms noch nicht im Einzelnen vorherzusehen waren. Einzige Voraussetzung ist dabei zumeist nur, dass die anzuzeigenden Daten eine bestimmte Struktur aufweisen, die vom Browser interpretiert werden kann. Derart generische Ansätze haben eine besondere Relevanz im Zusammenhang mit geschlossenen Geräten, wo die nachträgliche Erweiterung um einen Browser nur schwer möglich ist, und bei sehr großen oder heterogenen Nutzergruppen, bei denen der Aufwand für die Installation oder gar Entwicklung eines spezifischen Browsers zu groß wäre.

Zur Realisierung generischer Datenrepräsentationen kamen zunächst überwiegend proprietäre Ansätze zum Einsatz (vgl. „Generischer Klient“ [Mer96] oder „DIANA“ [KAC+96]). Trotz ihrer prinzipiellen Eignung für beliebige Anwendungskontexte waren diese zumeist aufgrund ihrer geringen Verbreitung nicht für die Verwendung in geschlossenen Geräten aus dem Bereich der Massenelektronik (Mobiltelefone usw.) brauchbar. Erst nachdem mit HTML eine relativ allgemeine deklarative Beschreibungssprache zu großer Verbreitung gelangte, änderte sich diesbezüglich die Situation. HTML Browser wurden auch in geschlossene Gerätekonzepte einbezogen (z.B. sog. Webphones) und Anwendungssysteme um eine Datenrepräsentation in HTML erweitert, auf die auch mittels solcher Geräte zugegriffen werden konnte. Mit einem an sich generischen Gerät wie besagtem Webphone konnte nunmehr genauso auf Unternehmensdaten im firmeneigenen SAP System zugegriffen werden, wie über das Internet entfernte Anwendungssysteme verwendet werden konnten, etwa bei der Abfrage eines Buchkatalogs oder der hierauf folgenden Bestellung eines bestimmten Titels. Die beiden genannten Beispiele zeigen wichtige Merkmale der applikationsserverbasierten Integration: Auf die Daten eines Anwendungssystems (SAP System, Buchkatalog bzw. –bestellsystem usw.) wird nicht über eine Schnittstelle zugegriffen, die mittels eines plattform– sowie anwendungsspezifischen Zugriffsprogrammes realisiert ist, sondern vielmehr über die in einer generischen Beschreibungssprache realisierte Schnittstelle, die im lokalen Browser lediglich angezeigt wird.

Während HTML in diesem Zusammenhang einen sehr großen Verbreitungsgrad erlangt hat, handelt es sich jedoch auch bei HTML prinzipell um eine spezifische Datenrepräsentation, in der direkt bestimmte Annahmen (z.B. bezüglich der Bildschirmgröße) über die Anzeigeumgebung gemacht und diese in die Kodierung der Schnittstelle einbezogen werden. HTML steht als Beschreibungssprache nicht allein da: Gerade im Bereich des mobilen Zugriffs auf Informationen haben sich andere deklarative Sprachen durchgesetzt, so zum Beispiel WAP ([For99e, KT01]) oder i–Mode [C. 00]. Bei der Aufbereitung der im zugrunde liegenden Anwendungssystem aufbereiteten Daten ergeben sich somit Abhängigkeiten vom verwendeten Browsertyp. Um diese zu umgehen, wird mittlerweile zunehmend auf XML (vgl. etwa [ABK+00]) als tatsächlich generischer Repräsentation übergegangen, die mittels entsprechender Transformationsregeln in eine konkrete Repräsentation überführt werden kann.

Im Bereich der applikationsserverbasierten Integration existiert mittlerweile eine Vielzahl verschiedener Ansätze, die jeweils bestimmte Integrationsaspekte betonen bzw. besonders berücksichtigen. Stellvertretend sollen hier einige Ansätze näher beschrieben werden, nämlich das am T.J. Watson Research Center von

IBM entwickelte System „DataX“ [LBT99] in Abschnitt 4.2.2.1, dessen Erweiterung MDSS in Abschnitt 4.2.2.2 sowie das von Oracle stammende „Portal To Go“[Hal00b, Hal00a] bzw. dessen Nachfolger „OracleMobile“ [Ora00] in Abschnitt 4.2.2.3. In allen Fällen handelt es sich um Ansätze, mit denen von beliebigen Endgeräten auf die in einer zentralen Datenbasis gespeicherten Daten zugegriffen werden können soll, wobei ein generisches Zwischenformat Verwendung findet, um die Anwendungslogik von der Präsentation zu trennen.

4.2.2.1 DataX

Bei DataX handelt es sich um eine Variante der applikationsserverbasierten Integration, die speziell mit dem Ziel entwickelt wurde, einen universellen Zugriffsmechanismus für Datenbanken zu realisieren, der von beliebigen mobilen Geräten verwendet werden können sollte. Besonderes Augenmerk lag dabei auf der Unterstützung ressourcenschwacher Endgeräte. Endgerät und Datenbank sind bei DataX so weit entkoppelt, dass hinsichtlich der verwendeten Datenbank genauso wenige Vorgaben gelten, wie hinsichtlich des verwendeten Endgerätes. Eine Besonderheit von DataX gegenüber anderen Ansätzen dieser Klasse besteht darin, dass bei DataX der Datenzugriff auch dann noch gewährleistet ist, wenn die Kommunikationsverbindung zwischen Datenbank und Endgerät vorübergehend unterbrochen ist.

Erreicht wurden diese Ziele mit einer dreischichtigen Architektur, bei der zwischen Endgerät und Datenbank ein Stellvertreter („Proxy“) etabliert wird, der im Datenaustausch zwischen beiden Partnern vermittelt. Dabei wurden zunächst zwei grundsätzliche Entwurfsentscheidungen getroffen:

1. Die Daten werden auf dem Endgerät in einem generischen, geräteunabhängigen Format verarbeitet[5]
2. Die Präsentationslogik wird von der Anwendungslogik getrennt

Der Entwurf von DataX wurde in vier Teilbereiche zerlegt, die getrennt betrachet wurden und deren Lösung jeweils in Teilkomponenten des Proxy mündete:

Subsetting : Selektive Replikation der Datenbank auf das mobile Endgerät

Targeting : Reduktion der Daten auf der Basis von Benutzervorgaben, Gerätemerkmalen sowie Merkmalen der Kommunikationsverbindung

Synchronisation : Abgleich der (möglicherweise geänderten) Datenkopie auf dem Endgerät mit der Datenbank

Rendering : Darstellung der Daten sowie Zugriff auf die Daten in einer auf die Merkmale des Gerätes angepassten Weise

[5] Da die Präsentation der Daten zwar in einem generischen Format erfolgt, jedoch in einem proprietären (für DataX spezifischen) Browser, ist DataX eher mit den Ansätzen von Merz und Keller et al. zu vergleichen als mit solchen, bei denen die Darstellung in einem Webbrowser oder WAP Browser erfolgt.

Offensichtlich müssen gerade bei einer größeren Datenbasis die Daten zunächst in geeigneter Weise reduziert werden, bevor sie zum mobilen Endgerät übertragen werden können. Diese Aufgabe ist bei DataX im Prozess des *Subsetting* (engl. für „Bilden einer Untermenge “) zusammengefasst. Hierzu werden so genannte Folder definiert, in denen die Kriterien formuliert werden können, nach denen die Reduktion stattfinden soll. Ein Folder beschreibt jeweils eine Untermenge der zugrunde liegenden Datenbasis und Operationen, mittels derer auf diese Untermenge zugegriffen werden darf. Ein Folder besteht hierbei aus vier Teilkomponenten:

1. **Folder Header** : Enthält den Namen des Folders sowie Parameter und deren Defaultwerte, anhand derer die Datenmenge eingeschränkt wird

2. **Data** : SQL–Abfrage mittels derer die tatsächliche Auswahl stattfindet und die auf den zuvor im Header beschriebenen Parametern basiert

3. **Operations** : Operationen, die auf den selektierten Daten erlaubt sind. Festgelegt wird hierbei, auf welche Einträge modifiziert werden dürfen, ob das Löschen von Einträgen erlaubt ist usw.

4. **Constraints** : Integritätsbedingungen, die für die innerhalb des Folders enthaltene Teilmenge der Datenbasis gelten und die durch den Browser sicherzustellen sind

Durch den Administrator können beliebige Folder definiert und auf diese Weise quasi „Sichten“ auf die Datenbank erstellt werden, die ein Anwender für den Zugriff auf die Datenbasis auswählen kann. Im Rahmen der Kapazität des Endgerätes ist es durchaus möglich, mehrere Folder auszuwählen und auf das Endgerät zu übertragen. Es ist Aufgabe des Proxy, beim Datenabgleich dafür zu sorgen, dass jeder der vom Anwender ausgewählten Folder parametrisiert und durch Zugriff auf die Datenbank mit der entsprechenden Untermenge an Daten gefüllt wird, um dann in einem weiteren Schritt auf die Übertragung zum Endgerät vorbereitet werden zu können.

Vielfach ist eine direkte Übertragung der Daten zum Endgerät nicht möglich, sei es, dass die Daten (z.B. im Falle von Audiodaten oder Bilddaten) zum Teil nicht im geeigneten Format vorliegen, um auf dem Endgerät dargestellt zu werden, oder dass die im Folder zusammengefassten Zeilen (z.B. bei Memofeldern) zu groß sind, um auf dem Endgerät verarbeitet werden zu können. Aus diesem Grund wird bei DataX nach der Selektion der gewünschten Teilmenge der Daten mittels *Targeting* eine weitere Anpassung der im Folder gespeicherten Daten durchgeführt, wobei sowohl Merkmale des Endgerätes sowie der Kommunikationsverbindung als auch Präferenzen des Benutzers Eingang finden. Der Anpassungsvorgang wird in zwei wesentliche Verfahren unterteilt, die zur Anwendung kommen:

Filtering : Eine bloße Reduktion der Daten ohne deren Modifikation, beispielsweise die Auslassung einzelner Spalten einer Tabelle bei Übertragung zum Endgerät

Transcoding : Transformation der Daten, entweder gemäß eines Wiedergabetreuemaßes („fidelity“), z.B. bei Umformung von formatierten Texten in einfache ASCII Texte, oder gemäß eines Maßes der Wiedergabeart („modality “), z.B. bei Transformation von Text in gesprochene Sprache

Die Transformation wird dabei durch Regeln gesteuert, die sich entsprechend in Filterregeln sowie Transformationsregeln unterteilen und zudem in drei Klassen weiter aufschlüsseln lassen:

- Allgemeine Regeln (*general rules*)
- Anwendungsspezifische Regeln (*application specific rules*)
- Benutzerspezifische Regeln (*user specific rules*)

Jede Klasse kann eine beliebige Anzahl von Regeln enthalten, die nacheinander auf die im Folder zusammengefassten Daten angewendet werden, wobei zunächst die benutzerspezifischen Regeln, dann die anwendungsspezifischen und schließlich die allgemeinen Regeln verwendet werden.

Regeln bestehen aus Tupeln, in denen jeweils Bedingungen und zugehörige Aktionen zusammengefasst sind. Die Bedingungen beziehen sich dabei jeweils auf:

- Gerätemerkmale
- Merkmale des Kommunikationspfades
- Folder–Namen bzw. Feldnamen
- Datentypen
- Benutzerkennungen

Beispielsweise könnte die Bedingung „Gerät ist vom Typ PalmPilot“ (Gerätemerkmal) zu der Aktion „Transformation der Daten auf max. Bildschirmauflösung von 160 mal 160 Bildpunkten“ führen, die Bedingung „Übertragungsgeschwindigkeit kleiner 64 KBit und Datentyp gleich Videosequenz“ (Kommunikationspfad / Datentyp) zu der Aktion „Übertragung nur des ersten Bildes der Videosequenz“ führen oder die Bedingung „Benutzerkennung gleich ‚mueller–wilken‘ und Folder–Name gleich ‚Kunden‘“ (Benutzerkennung / Folder–Name) zu der Aktion „Weglassen des Feldes ‘Anfahrtsbeschreibung,“ führen usw.

Die so für die Verwendung auf dem Endgerät optimierten Daten des Folders werden schließlich im Zuge der *Synchronisation* vom Proxy an das Endgerät weitergeleitet. Neben einer „echten“ Synchronisation, d.h. der Verknüpfung der auf dem Endgerät befindlichen Daten mit den in der Datenbank vorhandenen Daten, wird dabei auch die einfache Übertragung von Originaldaten aus der Datenbank auf das Endgerät unterstützt, wie es bei Daten vorteilhaft ist, die auf dem Endgerät nicht modifiziert werden können und für die die aufwendige Synchronisation entfallen kann.

Durch die Aufweichung sonst üblicher Konsistenzkriterien wurde dafür gesorgt, dass das Endgerät eine maximale Autonomie bei der Modifikation von Daten

erhält. So muss bei der Reintegration der auf dem Endgerät befindlichen Daten nicht jede einzelne Änderung übertragen werden. Vielmehr wird bei jedem Synchronisationsvorgang ein Abbild des Folders auf dem Proxy zwischengespeichert, der beim nächsten Synchronisationsvorgang dafür verwendet wird, die zwischenzeitlich aufgetretenen Änderungen zu erkennen. Anhand gewisser Präzedenzregeln wird dann im Einzelfall entschieden, wie bei Problemfällen zu verfahren ist. Dabei kann dann der Serverkopie oder der Kopie auf dem Endgerät Vorrang gegeben werden, ein Defaultwert eingesetzt werden oder eine externe Behandlungsroutine angestoßen werden (vgl. [LBT99], Abschnitt 3.3).

Die auf das Endgerät auszugebenden Daten werden schließlich in ein geräteunabhängiges XML Format übertragen und jeweils als vollständiges XML Dokument durch den Proxy weitergeleitet. Jedes Dokument enthält neben den eigentlichen Daten, wie sie aus dem Targeting hervorgegangen sind, ein Datenschema, das den Aufbau der übertragenen Tabelle festlegt, eine Austauschsemantik, welche für jedes Feld festlegt, inwieweit es auf dem Endgerät modifiziert werden darf sowie eine Menge von Integritätsregeln, die für die übertragenen Daten gelten. Die vom Endgerät empfangenen XML Dokumente werden durch den *Renderer* analysiert und dann zur weiteren Sichtung bzw. Bearbeitung durch den Anwender dargestellt. Dabei agiert der Renderer auf Folderebene. Im Folder enthaltene Tabellen können angezeigt oder im Einklang mit den kodierten Zugriffsregeln modifiziert, erweitert oder gelöscht werden. Wurden vom Anwender mehrere Folder für die Verwendung auf dem Endgerät ausgewählt, so kann zwischen diesen beliebig gewechselt werden.

Nachdem mittels Targeting die Repräsentation den spezifischen Merkmalen des Endgerätes angepasst wurde, kann auch die konkrete Darstellung auf dem Endgerät beeinflusst werden. Hierzu kann neben dem eigentlichen XML Dokument auch ein Stylesheet an den Browser übergeben werden, das regelt, wie die im Folder enthaltenen Daten auf dem Endgerät dargestellt werden sollen. Dieses kann beispielsweise bedeuten, dass für bestimmte Einträge ein besonderer Zeichensatz zu verwenden ist oder dass die Anzeige in einer bestimmten Anordnung auf dem Bildschirm erfolgen soll. Auch hierbei kann wieder auf Präferenzen des Anwenders und auf die Gerätemerkmale Rücksicht genommen werden.

4.2.2.2 Mobile Data Synchronization System

In Erweiterung zu dem im vorigen Abschnitt beschriebenen Ansatz wurde das *Mobile Data Synchronization System* (MDSS) [BCG+00] entwickelt. Im Gegensatz zu DataX wird hierbei nicht auf einen proprietären Browser gesetzt, mit dem die in XML kodierten Daten angezeigt bzw. verändert werden können. Vielmehr wird XML lediglich als eine Zwischenrepräsentation eingesetzt, die ausschließlich der Kodierung zu übertragender Daten und der Realisierung eines korrespondierenden Übertragungsprotokolls dient. Zur eigentlichen Verarbeitung (d.h. Eingabe, Speicherung usw.) auf der Seite des Endgerätes bzw. des Diensterbringers können bei diesem Ansatz dem Datentyp angepasste Programme zum Einsatz kommen. Im Falle eines serverbasierten Informationssystems wäre dieses die im Anwendungskontext verwendete Datenbank oder der verwendete Applikationsserver. Im Falle eines PDAs könnten dieses eine Adressenverwaltung für Adressen oder ein Planer für Termindaten sein. Zur Übertragung der anwendungs– bzw.

gerätespezifischen Daten werden so genannte Adapter eingesetzt, die jeweils die Umformung der Daten aus dem vom Anwendungsprogramm verwendeten Datenformat in die XML Repräsentation sowie die umgekehrte Umformung aus der XML Repräsentation in das anwendungsspezifische Datenformat übernehmen. Neben der eigentlichen Umformung obliegt den Adaptern außerdem die Realisierung einer robusten Kommunikationszwischenschicht. Die auf dem Endgerät bzw. Server befindlichen Anwendungen tauschen mit den jeweils lokalen Adaptern über externe Schnittstellen (z.B. JDBC/ODBC oder Java APIs) Daten aus. Die Zwischenspeicherung der Daten sowie Interaktion mit entfernten Adaptern ist dann Sache der beteiligten Adapter selbst.

Die generische Zwischenrepräsentation wird bei MDSS als *Mobile Data Synchronization Protocol* (MDSP) bezeichnet. MDSP zerfällt in drei Teilkomplexe, nämlich die Kodierung von Daten in *Datenelementen*, von Befehlen in *Befehlselementen* sowie von Informationen in *Informationselementen*:

Datenelemente dienen zur Kodierung der eigentlichen generischen Zwischenrepräsentation. MDSP definiert hierbei nur den strukturellen Rahmen der eigentlichen Repräsentation. Die Daten selbst können dann einer beliebigen Notation folgen, so lange diese vom entsprechenden Adapter interpretiert und transformiert werden kann. Eine konkrete Form könnte z.B. die Markupsprache „vCalendarML“ sein, die zur Speicherung von Termindaten definiert wurde.

Befehlselemente definieren Operationen, die zum Austausch von Datenelementen verwendet werden können. Diese Operationen könne Befehle wie „GET“, „UPDATE“, „REMOVE“ usw. kodieren, die vom Adapter in entsprechende Aufrufe an der Schnittstelle des Anwendungsprogramms umgesetzt werden, für das der Adapter geschrieben wurde. MDSP definiert eine Art „In–Band“ Protokoll: Die Befehlselemente werden mit den Datenelementen in einem gemeinsamen XML Dokument übertragen, wobei Befehlselemente jeweils die zugehörigen Datenelemente umschließen.

Informationselemente dienen schließlich der Kodierung von Nachrichten, die sich aus der Verarbeitung von Datenelementen ergeben. Dieses können Empfangsbestätigungen, aus Anfragen resultierende Ergebnisse oder auch Fehlerkodes sein.

Das durch die drei Elementklassen definierte Protokoll ist nicht als endgültig anzusehen. Vielmehr kann es jederzeit durch Anpassungen der jeweiligen Dokumententypdefinitionen (DTDs) modifiziert und auf die Erfordernisse hin erweitert werden, die sich aus der Unterstützung eines bestimmten Anwendungssystems ergeben. Wichtig ist hierbei lediglich, dass die betroffenen Adapter entsprechend erweitert werden müssen.

Auf der Seite der Endgeräte existieren bei MDSS derzeit zwei unterschiedliche Datenmodelle, die zur Zwischenspeicherung der von der Gegenstelle übernommen Daten zum Einsatz kommen. Zur Speicherung unstrukturierter Daten wird auf ein objektbasiertes Modell auf der Basis des *Mobile Network Computer Reference Specification Data Synchronization Framework* (MNCRS DSF) gesetzt, die Speicherung strukturierter, tabellarischer Daten erfolgt mittels eines relationalen Datenmodells. Auf dem Endgerät verfügbare Anwendungen können über

standardisierte Schnittstellen auf diese Zwischenspeicher zugreifen. Im Falle der Verwaltung unstrukturierter Daten über eine Java API, im Falle tabellarischer Daten über JDBC bzw. ODBC.

4.2.2.3 Portal To Go

Ebenfalls ausgehend vom Ziel eines universellen Zugriffs auf eine zentrale Datenbasis bildet das von der Firma Oracle entwickelte System „Portal To Go“ [Hal00b, Hal00a] einen umfassenden Ansatz, dessen Schwerpunkt mehr in der Realisierung von Anwendungssystemen liegt als im bloßen Zugriff auf die in einer Datenbank gespeicherten Tabellen. Portal To Go geht vor allem dadurch über den einfachen Datenzugriff hinaus, dass die Datenbank nicht als einzige Informationsquelle angesehen wird, sondern vielmehr nur als eine mögliche Quelle zu sehen ist. Ähnlich wie bei MDSS kann bei Portal To Go die Anbindung von Quellen mittels Adaptern realisiert werden. Diese kommen in diesem Fall jedoch nicht nur bei der Adaption von Datenbankinhalten zum Einsatz, sondern auch bei der Anbindung von webbasierten Anwendungen, existierenden Applikationsservern oder beliebigen anderen Inhaltsquellen.

Anwendungen werden bei Portal To Go in so genannten „Portalen“ zusammengefasst. Ein Portal bildet einen zentralen Zugriffspunkt, durch den hindurch verschiedene Dienste in Anspruch genommen werden können. Im einfachsten Fall füllt das gesamte Anwendungssystem als ein Dienst das Portal aus. Daneben ist es jedoch auch möglich, dass das Portal sich aus verschiedensten Diensten zusammensetzt. Stellt das eigentliche Anwendungssystem ein zentrales Bestellsystem dar, so könnten im Portal neben diesem Dienst auch der aktuelle Börsenkurs des Unternehmens, Wetterinformationen der verschiedenen Unternehmensstandorte oder andere Dienste erscheinen, je nachdem, welche zusätzlichen Dienste sich als nützlich für die Anwender definieren lassen.

Portal To Go gliedert sich in folgende vier Basiskomponenten:

Request Manager : Der Request Manager realisiert die Schnittstelle zum zugreifenden Endgerät. Er übernimmt die Zugriffskontrolle zum Portal und führt notwendige Vorverarbeitungsschritte (z.B. die Interaktion mit dem WAP Gateway) durch, die im Zusammenhang mit der Geräteanbindung anfallen. Auf der anderen Seite interagiert er mit den verschiedenen Transformern, die dem Portal bekannt sind.

Transformer : Die Transformer steuern die Übertragung von Informationen und Befehlen zwischen dem gerätespezifischen Format, das durch den Request Manager an das Gerät ausgeliefert werden kann und dem generischen Format, das innerhalb des Portals Verwendung findet. Hierbei bedient sich der Transformer eines gerätespezifischen XSL Regelsatzes sowie eines vorgefertigten Transformationsprozessors, der anhand dieses Regelsatzes die Umformung von XML in die jeweils konkrete Beschreibungssprache durchführt. Alternativ kann der Transformer jedoch auch als Java Programm vom Anwendungsentwickler bereitgestellt werden, was hinsichtlich der Transformation größere Flexibilität ermöglicht.

Master Service : Master Services oder einfach „Services“ bilden die Schnittstelle, über welche die Portal To Go Laufzeitumgebung auf die eigentlichen Dienste bzw. die Adapter zugreift, welche den Diensten zugeordnet sind. Die dem Portal bekannten Services werden zu einer einheitlichen Sicht zusammengestellt, wobei dem Anwender im Idealfall keine Trennung in einzelne Services sichtbar wird, sondern das gesamte Portal als ein homogenes Informationssystem wahrgenommen wird.

Adapter : Portal To Go verwendet zur internen Repräsentation von Informationen eine spezielle Beschreibungssprache namens „Portal–To–Go XML“. Da externe Dienste üblicherweise nicht über die Fähigkeit verfügen, Ausgaben in diesem XML Dialekt zu kodieren, muss eine Überführung externer Beschreibungssprachen wie z.B. HTML in die interne Repräsentation stattfinden. Diese Aufgabe fällt bei Portal To Go den Adaptern zu. Adapter werden jeweils für eine bestimmte Datenquelle realisiert. Sie nutzen für die Interaktion mit der Datenquelle entweder ein standardisiertes Protokoll wie ODBC bzw. JDBC oder sprechen die Anwendung über eine spezifische API an. Die so gewonnenen Informationen werden sodann vom Adapter in Portal–To–Go XML umgewandelt und können über den korrespondierenden Service an einen Transformer weitergeleitet werden, der die konkrete Repräsentation erstellt.

Abbildung 4.7 zeigt schematisch das Zusammenspiel der einzelnen Komponenten. Wird mit einem Endgerät eine Verbindung zum Portal To Go Server aufgebaut (1.), so findet über den Request Manager zunächst die Anmeldung beim Portal statt. Hierbei bedient sich der Request Manager des für den Gerätetyp zuständigen Transformers, um eine geeignete Darstellungsform für den Anmeldedialog zu erstellen (1b.). Nunmehr werden alle Master Services aufgefordert, den aktuellen Zustand des Dienstes zu übermitteln, den sie vertreten (2.). Die Master Services leiten die Aufforderung an den jeweiligen Adapter weiter (3.), der in der passenden Kommunikationsform auf den zugrunde liegenden Dienst zugreift und die gewünschten Informationen erfragt (4.). Diese werden in Portal–To–Go XML übertragen und dann an den Master Service zurückübertragen (5.). Die Master Services leiten die nunmehr in XML kodierten Informationen an den für das Endgerät zuständigen Transformator weiter (6.), der die Informationen in die benötigte Zielrepräsentation überträgt und sie an den Request Manager weitergibt (7.). Vom Request Manager gelangen sie zu guter Letzt wieder an das Endgerät zurück (8.). Der so entstehende Zyklus wird ein ums andere Mal durchlaufen, solange das Endgerät mit dem Portal verbunden ist.

OracleMobile Online Studio [Ora00] basiert auf „Portal To Go“. Es verwendet dieselben Komponenten wie Portal To Go, wobei diese hinsichtlich mobiler Einsatzszenarien erweitert wurde. So wird zur Definition von Anwendungen die Beschreibungssprache MobileXML verwendet, die auf dem Sprachstamm von Portal–To–Go XML aufbaut. Zusätzlich zu den dort definierten Merkmalen enthält MobilXML die Möglichkeit, vorgefertigte Funktionsmodule einzubinden, die speziell auf die Verwendungsprofile mobiler Geräte zugeschnitten sind, (z.B. Routenplanung, Nachrichtendienste, Kurznachrichtendienste usw.) und die von Oracle in Form zusätzlicher Komponenten zur Verfügung gestellt werden. Dabei sind diese Komponenten so gestaltet, dass sie problemlos in eigene Anwendungen

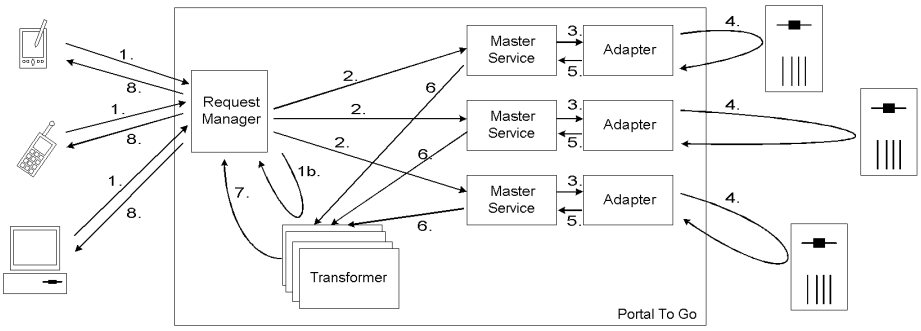


Abbildung 4.7: Schematisches Zusammenspiel der Teilkomponenten in Portal To Go

übernommen werden können und sich in ihrem Aussehen der Anwendung einerseits und dem verwendeten Gerät andererseits anpassen. Außerdem ermöglicht das Online Studio zusätzlich zu Portal To Go die Einbindung von Skripten, die eine serverseitige Personalisierung bzw. Parametrisierung der Anwendung ermöglichen.

Portal To Go bzw. OracleMobile Online Studio stellen einen sehr umfangreichen Ansatz zur Integration beliebiger Endgeräte in bestehende Informationssysteme dar. Das Zusammenspiel der generischen Zwischenrepräsentationen Portal–To–Go XML sowie MobileXML und den gerätespezifischen Transformern ermöglicht es, Dienste zu definieren, auf die flexibel über verschiedenste Zugriffskanäle zugegriffen werden kann. Dabei können auch im Nachhinein noch Kanäle erschlossen werden, indem weitere Transformer hinzugefügt werden. Die Zusammenstellung von einzelnen Diensten zu umfassenden Portalen bietet die Möglichkeit, Informationen aus verschiedenen Quellen zusammenzubringen, ohne dass für den Anwender eines mobilen Endgeräts ein zu großer Navigationsaufwand zwischen den einzelnen Teildiensten entsteht. Im optimalen Fall ist der Aufbau des Portals aus einzelnen Dienstangeboten für ihn nicht sichtbar.

4.2.3 Entwurfszeit–Transformation

Eine besondere Klasse der bislang beschriebenen Herangehensweisen stellen diejenigen Ansätze dar, bei denen die Adaption an verschiedene Endgeräte bereits in die Entwurfsphase der Anwendung mit einbezogen wird. Bei dieser Herangehensweise, die als Entwurfszeit–Transformation bezeichnet werden soll, wird die Präsentationslogik mittels wohldefinierter Beschreibungssprachen abstrakt beschrieben und in externen Dateien abgelegt oder mittels spezieller Programmierbibliotheken das Anwendungsprogramm selbst dazu in die Lage versetzt, solche Beschreibungen zur Laufzeit zu generieren. Während eines Programmlaufs wird die abstrakte Beschreibung dann auf eine für den Ansatz charakteristische Weise in eine konkrete Repräsentation umgesetzt. Ereignisse, die durch die Nutzung der konkreten Schnittstelle erzeugt werden, können dann von der Applikationslogik aufgefangen werden, um sie der weiteren Verarbeitung zuzuführen. Die tatsächliche Repräsentation ist dabei zumeist nicht in der abstrakten Schnittstellenbeschreibung festgeschrieben, sondern wird durch eine Stilbeschreibung

bewirkt, die, je nach gewünschter Ausprägung, der Schnittstelle hinzugefügt wird. Abb. 4.8 zeigt schematisch, wie dieses vonstatten gehen kann.

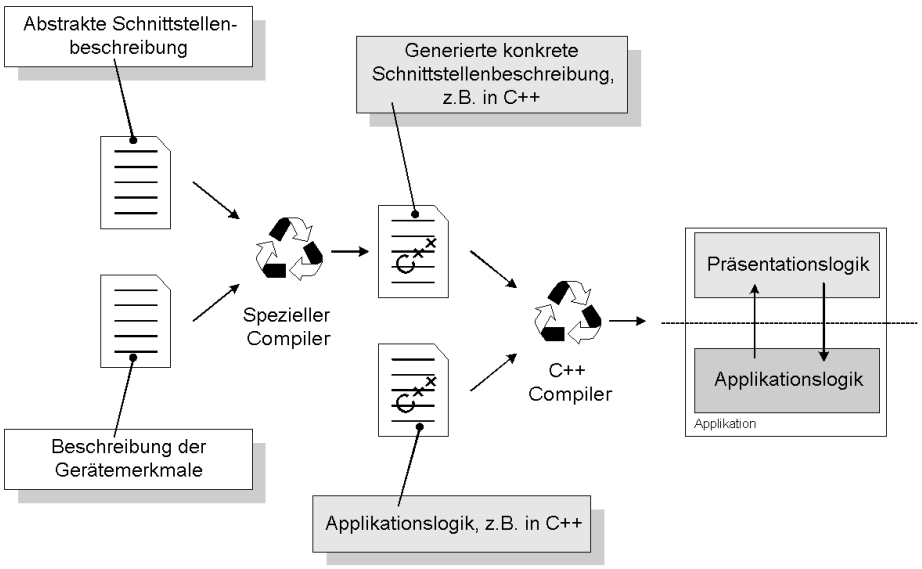


Abbildung 4.8: Entwurfszeit–Transformation als Ansatz zur Entkopplung von Applikationslogik und Präsentationslogik

Wie aus der Abbildung ersichtlich ist, wird zunächst eine abstrakte Beschreibung der Benutzungsschnittstelle entworfen. Diese wird dann gemeinsam mit einer Beschreibung der gewünschten Zielrepräsentation durch einen Compiler zu einem Quelltext in der Programmiersprache zusammengefügt, die zur Realisierung des Anwendungssystems verwendet wird. Das so entstandene Anwendungsmodul wird gemeinsam mit der eigentlichen Anwendungslogik in das endgültige Anwendungsprogramm übersetzt, das dann die gewünschten Ausgaben auf dem Endgerät erzeugen kann. Die Anwendungslogik ist bei diesem Vorgehen intern klar von der Präsentationslogik getrennt, so dass bei Bedarf eine andere Stilbeschreibung an die Stelle der zuvor verwendeten treten kann, um die Darstellung auf einem anderen Gerätetyp zu ermöglichen. Hierbei kann im Idealfall auf die Modifikation des Anwendungscodes verzichtet werden, so dass lediglich das neu erstellte Anwendungsmodul mit der in ihm enthaltenen geänderten Präsentationslogik gemeinsam mit dem ursprünglichen Anwendungsquelltext übersetzt wird. Auf diese Weise kann dann schnell ein auf die geänderten Bedürfnisse angepasstes Anwendungsprogramm erstellt werden.

Wie sich im weiteren Verlauf des Abschnitts zeigen wird, existieren diverse Abweichungen zu diesem Grundschema. So kann es sein, dass die Stilbeschreibung nicht bereits zur Entwurfszeit einbezogen wird, sondern hier lediglich festgelegt wird, wie die Anwendung auf bestimmte Ereignisse in der Benutzungsschnittstelle reagiert. Die konkrete Darstellung wird bei diesem Vorgehen dann zur Laufzeit geregelt, indem die Anwendung die entsprechende Beschreibung erst zu diesem Zeitpunkt aus einer externen Quelle einliest und zur Transformation der abstrakten Beschreibung in eine konkrete Repräsentation verwendet. Auch kann das Ergebnis des Übersetzungsprozesses eine Anwendung sein, in der die

Präsentationslogik nicht (wie in der Abbildung angedeutet) direkt mit der Anwendungslogik verbunden ist, sondern ein eigenständiges Modul darstellt, das lediglich über ein Netzwerk mit der Anwendungslogik kommuniziert. Schließlich kann es sein, dass die Präsentation zur Laufzeit wieder in eine konkrete Beschreibung umgesetzt wird, die dann in einem Laufzeitsystem (Browser, Renderer, Engine o.ä.) angezeigt wird. Diese Variante wurde in allgemeiner Form ja bereits in Abschnitt 4.2.2 angerissen.

Der Vorteil des Einsetzens zur Entwurfszeit liegt darin, dass eine Berücksichtigung der Anwendungssemantik möglich ist und eine optimale Abstimmung auf den Anwendungskontext bzw. auf mögliche Zielrepräsentationen erzielt werden kann. Insbesondere können hierbei bereits bei der internen Aufbereitung von Daten durch die Anwendungslogik Alternativen berücksichtigt werden, die bei der Darstellung unter verschiedenen Bedingungen zum Tragen kommen können. So kann eventuell eine Grobklassifikation in zu erwartende Zielrepräsentationen mit fehlenden, geringen oder normalen Darstellungsfähigkeiten erfolgen und im Anwendungscode dann die Erzeugung einer allgemeinen Beschreibung der Benutzungsschnittstelle für ebendiese Zielklassen vorgesehen werden. Abhängig von der tatsächlich geforderten Zielrepräsentation kann dann zunächst entschieden werden, welche der Klassen passend ist und die Vorbereitung der Darstellung demgemäß erfolgen. Die eigentliche Umsetzung in eine bestimmte Darstellungsform wird dann wieder anhand von Regelwerken durchgeführt, welche die allgemeine Darstellung in eine konkrete Repräsentation überführen.

Während das Prinzip der Entwurfszeit–Transformation bereits seit längerem bekannt ist, hat es für die Nutzung in mehrschichtigen Integrationssystemen durch die weite Verbreitung von XML und dessen regelgesteuerter Umsetzung mittels XSL stark an Bedeutung gewonnen. Im Folgenden sollen einige Ansätze vorgestellt werden, bei denen Anwendungssysteme bereits zur Entwurfszeit auf die Ausgabe allgemeiner Informationen über ihre Benutzungsschnittstelle vorbereitet sind.

4.2.3.1 User Interface Markup Language

Die User Interface Markup Language (UIML) wurde mit dem Ziel entwickelt, ein Verfahren zu schaffen, im Entwurf von Zugriffsmechanismen auf das Internet die Benutzungsschnittstelle von der Anwendungslogik zu trennen [APB+99]. Dabei sollten die Vorteile deklarativer Entwurfssprachen (Einfachheit und Verständlichkeit, schnelle Prototypentwicklung usw.) wie HTML aufgenommen und hinsichtlich der besonderen Anforderungen an die Nutzung von verschiedensten Endgeräten erweitert werden. Der zu realisierende Ansatz sollte unter anderem die Trennung zwischen Präsentation und Applikationslogik betonen. Er sollte flexibel erweiterbar sein, um auch zukünftige Geräte unterstützen zu können. Darüber hinaus sollte die Gruppierung von Gemeinsamkeiten über verschiedene Schnittstellen und deren Wiederverwendung ermöglicht werden. Schließlich sollte er auch von Nichtprogrammierern genutzt werden können, um so den Entwurf von Benutzungsschnittstellen durch diesbezügliche Spezialisten zu ermöglichen.

Die Definition einer Benutzungsschnittstelle zerfällt bei UIML in fünf Teilbereiche: „Beschreibung“, „Struktur“, „Daten“, „Stil“ und „Ereignisse“. Gemeinsam

```
<?xml version="1.0" standalone="no"?> <uiml version="2.0">
  <interface name="SomeName" class="MyClass">
    <description>...</description>
    <structure>...</structure>
    <data>...</data>
    <style>...</style>
    <events>...</events>
  </interface>

  <logic>...</logic>
</uiml>
```

Abbildung 4.9: Grundgerüst einer UIML Definition (nach [APB+99])

werden diese Aspekte einer Schnittstelle in einer Schnittstellendefinition (gemäß XML Syntax als `<interface>` bezeichnet) zusammengefasst.

In der *Beschreibung* werden alle Elemente zusammengefasst, welche die Schnittstelle ausmachen, also z.B. die Menüleiste, die einzelnen Menüpunkte, die Textfelder usw. Die Beschreibung bezieht sich dabei jedoch auf deren bloße Existenz, während inhaltliche oder gar semantische Aspekte nicht enthalten sind. Die *Struktur* bildet die Basis für die Realisierung der oben angedeuteten Klassifikation. Es werden diejenigen Elemente zusammengefasst, die logisch zueinander gehören und die entsprechend gemeinsam einbezogen oder ausgeschlossen werden können, wenn dieses für ein bestimmtes Gerät notwendig ist. Werden in der Beschreibungssektion z.B. Menüleisten und Menüpunkte definiert, so werden diese in der Struktursektion zu Menüs kombiniert. Alle Aspekte, die aus Sicht der Anwendung als zusammenhängende Bausteine aufgefasst werden müssen, werden in der *Datensektion* zusammengefasst. Die Datensektion definiert alle anwendungsspezifischen Elemente, ohne dabei plattformspezifische Vorgaben zu machen. So werden beispielsweise in der Datensektion den zuvor definierten Menüs konkrete Beschriftungen zugeordnet, wobei für verschiedene Lokalisationen natürlich unterschiedliche Datensektionen in der jeweiligen Landessprache möglich wären. Als *Stil* werden hingegen diejenigen Anteile definiert, die spezifisch für ein bestimmtes Gerät oder Ausgabemedium sind. Jeder Eintrag in dieser Sektion regelt die Darstellung eines bestimmten Elements der Benutzungsschnittstelle für eines der unterstützten Geräte. Somit können für eine Benutzungsschnittstelle auch im Prinzip beliebig viele Stildefinitionen existieren. Beispielsweise könnte das zuvor definierte Menü auf einem Arbeitsplatzrechner in gewohnter Form, also als vertikale Liste, aus der am oberen Bildschirmrand befindlichen Menüleiste auftauchen, während es für ein Mobiltelefon in die dort übliche verschachtelte Optionsliste integriert wird. In der Sektion der *Ereignisse* werden schließlich alle Ereignisse zusammengefasst und definiert, die im Zusammenhang mit der Benutzungsschnittstelle auftreten können und die als Nachrichten mit der zugrunde liegenden Anwendungslogik ausgetauscht werden müssen. Idealerweise wird jedes Element der Benutzungsschnittstelle so durch eines oder mehrere Ereignisse eingebunden und mit dem Anwendungskern verdrahtet. Aber auch Elemente der Schnittstelle untereinander können durch Ereignisse verbunden werden.

Während die Ereignissektion die Ereignisse definiert, die von der Benutzungs-

```
  <-- Definition eines Fensters mit einfachem Dateimenü -->

  <interface name="SomeName" class="MyClass">
    <description>
      <element name="Main" class="Main"/>
      <element name="File" class="ActionGroup"/>
      <element name="NewAction" class="ActionItem"/>
      <element name="CloseAction" class="ActionItem"/>
      <element name="QuitAction" class="ActionItem"/>
    </description>
    <structure>
      <element name="Main">
        <element class="Bar">
          <element name="File">
            <element name="NewAction"/>
            <element name="CloseAction"/>
            <element class="Separator"/>
            <element name="QuitAction"/>
          </element>
        </element>
      </element>
    </structure>
    <data>
      <content name="Main">Example</content>
      <content name="File">File</content>
      <content name="NewAction">New</content>
      <content name="CloseAction">Close</content>
      <content name="QuitAction">Quit</content>
    </data>
    <style>
      <attribute class="Main" type="rendering"
                 value="java.awt.Frame"/>
      <attribute class="Main" type="size" value="100,80"/>
      <attribute class="ActionItem" type="rendering"
                 value="jawa.awt.MenuItem"/>
      <attribute class="Separator" type="rendering"
                 value="wrapper.MenuSeparator"/>
      <attribute class="ActionGroup" type="rendering"
                 value="java.awt.Menu"/>
      <attribute class="Bar" type="rendering"
                 value="java.awt.MenuBar"/>
    </style>
    <events>
      <event name="SelectQuit" class="ActionSelect"
             source="QuitAction" trigger="Select">
        <action target="Main" method="exit"/>
      </event>
    </events>
  </interface>
</uiml>
```

Abbildung 4.10: Beispielhafte Schnittstellendefinition (nach [APB+99])

schnittstelle ausgehen können, werden hier keine *konkreten* Vorgaben hinsichtlich der Verdrahtung mit der Applikationslogik gemacht. Nach dem Verständnis der UIML–Entwickler ist dieses nicht Teil der eigentlichen Benutzungsschnittstelle und sollte demgemäß auch nicht als ein Teil hiervon definiert werden. Entsprechend wurde der Schnittstellenbeschreibung in der `<interface>` Sektion eine Beschreibung der Integrationslogik in der so genannten `<logic>` Sektion beigeordnet. In diesem Beschreibungsteil werden die eigentlichen Aufrufe kodiert, die durch das Eintreten eines Ereignisses ausgelöst werden. Diese Aufrufe sind dann spezifisch für die Programmiersprache, in der die Anwendungslogik programmiert ist. Je nachdem sind also Definitionen in C++, Visual Basic, Java usw. in der Logiksektion möglich. Durch die Trennung von Schnittstellenbeschreibung und Integrationslogik ist es auch möglich, dieselbe Schnittstelle durch Austausch der Integrationslogik mit Anwendungskernen in unterschiedlichen Programmiersprachen zu verbinden.

Insgesamt muss der deklarative Charakter von UIML betont werden. Es wird lediglich definiert, *was* geschehen soll, nicht jedoch, *wie* dieses zu erreichen ist[6]. Diese Aufgabe obliegt dann einer für jede Plattform spezifischen Softwarekomponente, dem so genannten Renderer, der die fünf Teilbereiche der Benutzungsschnittstelle und die Definition der Anbindungslogik zusammennimmt und in einer für das jeweilige Endgerät passenden Weise aufbereitet, wobei er sich der in den einzelnen Sektionen definierten Alternativen bedienen kann.

Hinsichtlich der Realisierung des Renderers existieren mehrere Möglichkeiten. Zunächst kann er in der von HTML bekannten Weise als Browser bzw. Interpreter realisiert sein. Er lädt hierbei die Schnittstellen– und Logikdefinition ein und setzt sie sodann in eine interaktive Anwendungsschnittstelle um. Diese Umsetzung kann auch in einem zwischen Endgerät und Anwendungskern positionierten Dienst erfolgen und durch eine Umsetzung beispielsweise nach HTML oder WML die Nutzung auch von Geräten ermöglichen, für die kein UIML Browser existiert. Eine Alternative besteht darin, die Definitionen durch einen Compiler in Quellcode der Entwurfssprache zu übersetzen, mit der die Anwendungslogik realisiert ist. Dieser Quellcode kann dann gemeinsam mit der Anwendungslogik kompiliert und so zu einem eigenständigen Programm zusammengefasst werden (in [Bin00] und [BS01] wird auf Aspekte des Rendering genauer eingegangen).

UIML stellt einen sehr generischen und daher flexiblen Ansatz zur Definition von Benutzungsschnittstellen und zu deren Ankopplung an eine lokale oder entfernte Anwendungslogik dar. Um UIML für beliebig existierende bzw. kommende Schnittstellentypen einsetzen zu können, wurde der Sprachkern auf ein Minimum reduziert, während ein Großteil sowohl des Inhalts als auch der Struktur der Schnittstelle durch die fünf Teilbereiche formuliert wird. Wie in Abb. 4.2.3.1 deutlich wird, schafft diese Flexibilität aber bereits für einfache Beispiele eine enorme Komplexität. Nicht zuletzt bedingt durch den deklarativen Sprachcharakter werden Schnittstellendefinitionen schnell groß und daher unübersichtlich sowie schwer handhabbar, was dem Anspruch an die einfache Nutzbarkeit zuwiderläuft. Aus diesem Grunde existieren für UIML domänenspezifische Anpassungen, in denen gewisse Annahmen über die zu unterstützenden Schnittstellen in

[6] Jedenfalls hinsichtlich der Aufbereitung auf dem Endgerät. In der Logiksektion muss anhand konkreter Aufrufe der zugrunde liegenden Anwendungslogik natürlich eine gewisse Aussage darüber gemacht werden, wie die Anbindung an den Anwendungskern vonstatten gehen soll.

```
<interface name="DSUIML Beispiel" class="JavaAWT">
  <window name="Main" content="UIML Example">
    <menubar name="Selections">
      <menu name="FileSelection" caption="File">
        <menuitem name="itemF0" caption="New" shortcut="Ctrl-N"/>
        <menuitem name="itemF3" caption="Close"/>
        <menuseparator/>
        <menuitem name="itemF4" caption="Quit"/>
      </menu>
    </menubar>
  </window>
</interface>
```

Abbildung 4.11: Schnittstellendefinition in UIML/GUI (nach [APB+99])

die Definition bestimmter *Tags* einfließen, die erst mittels XSL in die eigentliche UIML Definition übersetzt werden. Eine solche Domäne ist z.B. die Beschreibung grafischer Schnittstellen, die in UIML/GUI definiert ist. Während domänenspezifische Anpassungen eine Einschränkung der Flexibilität von UIML darstellen, führen sie andererseits zu einer deutlichen Vereinfachung, wie aus Abb. 11 ersichtlich ist.

4.2.3.2 Universal Lightweight Client

Während die anderen in diesem Abschnitt beschriebenen Verfahren zur Trennung von Präsentation und Applikationslogik auf der Beschreibung der Benutzungsschnittstelle in XML und deren Darstellung in einem speziellen Browser bzw. dessen Transformation in eine gängige Beschreibungssprache basieren, handelt es sich beim Konzept der *Universal Lightweight Clients* (ULC) um eine vergleichsweise programmiernahe Herangehensweise. Diese setzt nicht auf die Definition einer speziellen Beschreibungssprache, sondern beruht auf einer Modifikation der verwendeten GUI Bibliothek und einer hieraus resultierenden Zerlegung der Benutzungsschnittstelle in einen clientseitigen und einen serverseitigen Anteil. Dabei sollen die aus der zentralisierten Struktur typischer Terminalanwendungen erwachsenden Vorteile mit denjenigen moderner grafischer und auf der Nutzung von Browsern basierenden Benutzungsschnittstellen im Internetumfeld kombiniert werden. Das primäre Ziel von ULC besteht auch weniger in der Realisierung plattformunabhängiger Benutzungsschnittstellen (obwohl diese durchaus Resultat sein können), sondern vielmehr in der Schaffung einer möglichst leichtgewichtigen und kommunikationsminimalen Trennung zwischen Präsentationslogik und Anwendungslogik.

Kerngedanke von ULC ist der Übergang von einer klassischen Client/Server Architektur zu einem dreischichtigen Design, bei dem die Anwendungslogik in einer mittleren Applikationsschicht zusammengefasst wird. Wie bei den anderen beschriebenen Ansätzen wird auf der Granularitätsstufe von GUI Widgets angesetzt und jedes Widget als Kombination zweier „Halbobjekte“[7] realisiert,

[7] vgl. *Half Object Pattern* nach Meszaros [Mes94] sowie *Proxy Pattern* nach Gam-

von denen eines innerhalb der Applikationslogik angeordnet ist und das andere sich in einem speziellen Browser auf dem Endgerät befindet. Entsprechend besteht ULC aus einer in Java geschriebenen *Universal Interface Engine*, die als Applet oder eigenständige Anwendung auf den Client geladen wird und dort zur clientseitigen Darstellung von Elementen der Benutzungsschnittstelle sowie zur Entgegennahme von Daten und deren Weiterleitung an die Applikationslogik eingesetzt wird. Auf der Seite der Applikationslogik steht die *ULC Application*, die das Geschäftswissen der Anwendung repräsentiert und eine abstrakte Version der Benutzungsschnittstelle, bestehend aus speziellen GUI Widget Objekten, ohne konkrete Darstellungskomponente realisiert. Aufgrund der fehlenden Darstellungskomponente werden diese Widgets auch als *faceless widgets* bezeichnet. Jedem Faceless Widget auf Seite der Applikationslogik steht ein konkretes Widget in der UI Engine gegenüber. Die beiden Komponenten kommunizieren über das Netzwerk in einem speziellen Protokoll. Die ULC Applikation kann ihrerseits mit entfernten Diensten kommunizieren, welche die der Anwendung zugrunde liegenden Daten oder allgemeine, gemeinsam genutzte Businessobjekte bereitstellen.

Die Anwendungslogik wird in der für die verwendete Programmiersprache üblichen Weise realisiert, wobei die durch die zur Realisierung von Faceless Widgets angepasste GUI Bibliothek in ihrer API weitestgehend mit der ursprünglichen Version übereinstimmt. Erweiterungen der API betreffen vor allem Aspekte der optimierten Kommunikation zwischen Präsentation und Applikationslogik, d.h. die Abhängigkeiten zwischen einzelnen GUI Elementen, das Zwischenspeichern von Daten im GUI und die asynchrone Datenübernahme aus dem GUI in die Applikationslogik. Auf der Seite des Client residieren zu keinem Zeitpunkt applikationsspezifische Programmteile, sondern allenfalls Routinen wie z.B. zur Plausibilitätsüberprüfung einzelner Datenfelder. Das kompilierte Anwendungsprogramm baut beim Start eine Verbindung zur UI Engine auf und kommuniziert während des Anwendungslaufs mit dieser zur Darstellung der Benutzungsschnittstelle.

Besonderer Wert wurde bei ULC darauf gelegt, dass das System auch in Umgebungen mit mäßiger Kommunikationsbandbreite bzw. hoher Latenz zum Einsatz kommen kann. Die UI Engine kann die Übernahme von Teilen der Benutzungsschnittstelle auf den Zeitpunkt verzögern, wo diese tatsächlich zur Anzeige gebracht werden, etwa bei Auswahllisten, von denen nur ein Teil dargestellt werden kann. Auch können statische Anteile des GUI wie z.B. Bilder oder bestimmte Dialogboxen asynchron und nur einmalig in die Engine übertragen werden und sodann bei Bedarf aus dem lokalen Cache heraus angezeigt werden. Darüber hinaus kommunizieren alle Halbobjekte asynchron mit ihrem jeweiligen Gegenüber. Wird beispielsweise in einer Liste geblättert, so kommt es daher nicht zu Verzögerungen, wenn einzelne Datenelemente noch nicht übertragen werden konnten. Vielmehr werden in solch einem Fall zunächst Stellvertreterobjekte angezeigt, die dynamisch gegen die tatsächlichen Daten ausgetauscht werden, sofern diese über das Netzwerk nachgeladen werden konnten. Schließlich wurde die UI Engine zustandslos realisiert: Alle Informationen der Benutzungsschnittstelle werden zu jedem Zeitpunkt durch die ULC Applikation gepuffert, so dass im Falle eines Kommunikationsabbruchs nach dem Neustart der Engine die Daten aus der

ma et al. [GHJV95].

Applikationsschicht übernommen werden können und die auf dem Client angeordneten Halbobjekte mit den Daten ihrer korrespondierenden Faceless Widgets initialisiert werden können. Dieses Vorgehen schafft eine zusätzliche Robustheit und ist insbesondere beim Einsatz als Applet in einem Browser unumgänglich.

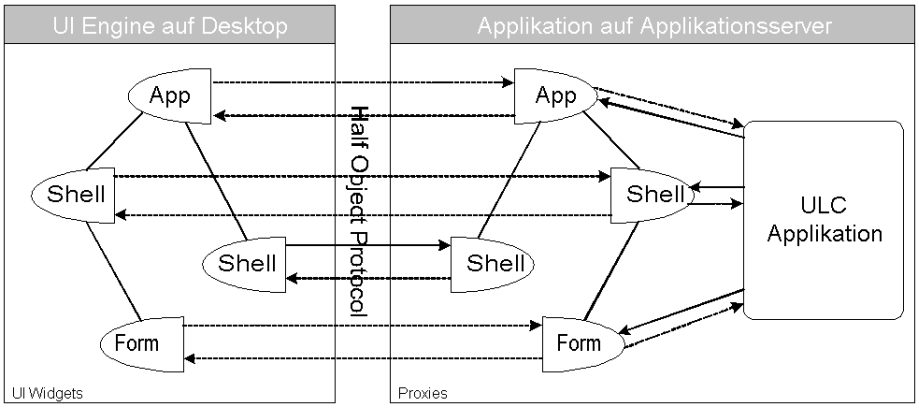


Abbildung 4.12: Schematischer Aufbau einer Anwendung in ULC (vgl. [MSGW98])

Der schematische Aufbau einer ULC Anwendung ist in Abb. 4.12 dargestellt. Jedem Widget in der UI Engine steht ein Widget in der Applikation gegenüber, das durch die ULC Applikation gesteuert und von dieser mit Informationen aus der zugrunde liegenden Datenschicht versorgt wird.

Um einen Einsatz von ULC auch dort zu ermöglichen, wo eine direkte Modifikation der zugrunde liegenden Businesslogik nicht möglich ist und daher der nachträgliche Austausch von GUI Objekten durch spezielle ULC Halbobjekte ausgeschlossen ist, bedient sich ULC einer in [MSGW98] als *Screen–Scraping* bezeichneten Technik: Hierbei wird die Bildschirmausgabe der eigentlichen Anwendung durch eine spezielle Bibliothek auf Serverseite überwacht und an die Applikationslogik weitergeleitet. Auf dem entgegengesetzten Weg werden dann Kommandos in der Applikationslogik wie Tastaturbefehle zusammengesetzt und durch die Bibliothek als solche an die entfernte Anwendung übermittelt. Es ist dann Aufgabe der Applikationslogik, die entgegengenommenen Bildschirminhalte zu analysieren, diese in Halbobjekte umzusetzen und so der Präsentationsschicht zur Verfügung zu stellen bzw. die richtigen Tastaturkommandos aus den eingehenden Ereignissen abzuleiten.

4.2.3.3 eXtensible User interface Language

Die *Extensible User interface Language* (XUL) wurde zur Definition von Benutzungsschnittstellen im Rahmen des Mozilla Projekts entworfen, in welchem ein Webbrowser auf der Basis des Netscape Communicators entwickelt wird [Pro99]. Wichtiges Entwurfskriterium im Mozilla Projekt war die möglichst plattformübergreifende Nutzbarkeit des Browsers und die Möglichkeit zur schnellen Anpassung an neue Plattformen. Dieses sollte unter anderem durch eine Trennung von Anwendungslogik und Präsentationsschicht erreicht werden, die

konzeptionell im *Cross Platform Front End* (XPFE) adressiert wurde. Das XPFE enthält neben XUL als der eigentlichen Sprache zur Definition von Schnittstellen des Weiteren Komponenten zur Anzeige von in XUL formulierten Schnittstellenbeschreibungen (Gecko Layout Engine). In einer zu UIML vergleichbaren Weise tritt der Gecko bei XPFE als Renderer auf, der die Kopplung der Schnittstelle mit der zugrunde liegenden Anwendungslogik übernimmt.

In der XUL Terminologie wird derjenige Teil der Benutzungsschnittstelle, der mittels XUL definiert wird, als *Chrome* bezeichnet. Unter diesem Begriff werden alle Teile eines Anwendungsfensters zusammengefasst, die nicht mit der Darstellung anwendungsspezifischer Inhalte zusammenhängen. Im Falle eines Webbrowsers wären dieses die Menüzeile, Statuszeilen, Dialogfenster, Auswahlfelder usw., während die angezeigte Webseite anwendungsspezifisch und somit nicht Teil des Chromes ist.

Die eigentlichen Inhalte werden nicht als Chrome bezeichnet und in XUL definiert. Ihre Aufbereitung und Darstellung obliegt der alleinigen Verantwortung der Anwendung selbst. Die mittels XUL beschriebene Schnittstelle wird in Widgets aufgespalten. Widgets stellen in sich geschlossene Elemente der Schnittstelle dar, die jeweils einen Teil des Anwendungsfensters ausfüllen und neben einer bestimmten Optik auch ein definiertes Verhalten besitzen. Widgets können miteinander kombiniert werden und so wiederum zu größeren Widgets zusammengefasst werden. XUL Beschreibungen bestehen genau aus Definitionen von solchen Widgets und ihres spezifischen Verhaltens.

XUL basiert auf der Verwendung von XML als Beschreibungssprache und dessen Kombination mit *Cascading Style Sheets* (CSS). Ersteres dient primär zur Beschreibung der Struktur der Schnittstelle, während Letzteres deren Aussehen definiert. XUL besitzt eine enge Verwandtschaft mit HTML4. So können in einer XUL Beschreibung bestimmte Bereiche des Aufbaus der Schnittstelle in HTML formuliert werden, und die Verdrahtung der definierten Widgets mit der zugrunde liegenden Anwendungslogik erfolgt über JavaScript. Wie von HTML bekannt wird JavaScript im Übrigen zur Definition aller Aspekte der Schnittstelle herangezogen, die eine Interaktion der Widgets untereinander betrifft, d.h. bei denen keine direkte Auswirkung auf den Anwendungskern existiert.

Das in Abbildung 4.13 dargestellte Beispiel eines XUL Dokuments zeigt den typischen Aufbau einer Schnittstellenbeschreibung in XUL. Die optische Aufbereitung der Schnittstelle wird über ein externes Stylesheet (xul.css) geregelt, das in der Präambel eingebunden wird. Daraufhin erfolgt die Definition eines verschachtelten Widgets, das ein Fenster mit zugehöriger Menüzeile enthält. Ereignisse werden über JavaScript Operationen realisiert, die Aufrufe der externen Anwendungslogik enthalten. Der Fensterinhalt wird nicht in XUL bzw. als Teil des XUL Dokuments definiert. Er wird mit herkömmlichem HTML realisiert und von der Schnittstelle über eine externe Referenz angebunden.

4.2.3.4 XML Formularbeschreibungssprachen

Den speziellen Bereich des elektronischen Formularwesens decken Ansätze wie derjenige der *Extensible Forms Definition Language* (XFDL) bzw. der *XML Forms Architecture* (XFA) ab. Hier wird anhand plattformunabhängiger Schnittstellendefinitionen versucht, Formulare zur durch Computer unterstützten Da-

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/xul.css"
                 type="text/css"?>
<!DOCTYPE window>
<window id="main-window"
        xmlns:html="http://www.w3.org/1999/xhtml"
        xmlns="http://www.mozilla.org/keymasters/
                      gatekeeper/there.is.only.xul">

  <menubar>
    <menu name="File">
      <menuitem name="Open" onclick="dump('File:Open!\n');"/>
      <menuitem name="Close" onclick="dump('File:Close!\n');"/>
      <menuitem name="Quit" onclick="dump('File:Quit!\n');"/>
    </menu>
  </menubar>

  <html:iframe id="content-frm" src="contentfrm.html" flex="100%"/>
</window>
```

Abbildung 4.13: Eine einfache Benutzungsschnittstelle in XUL (nach [Pro99])

tenerfassung und –verarbeitung zu realisieren und in eine Laufzeitumgebung einzubetten. Insofern handelt es sich bei den hier realisierten Benutzungsschnittstellen eher um Eingabemasken als um GUIs im üblichen Sinne.

Ähnlich wie die zuvor beschriebenen Ansätze gilt für XFA und XFDL, dass hier zunächst mit einer speziellen Definitionssprache auf Basis von XML eine Benutzungsschnittstelle erstellt wird. Diese kann zunächst aus einer Anzahl von typischen Elementen eines GUI bestehen (obwohl die Sprache im Vergleich diesbezüglich eher beschränkt ist). Mit ihrer Hilfe wird die visuelle Komponente des Formulars beschrieben, die Eingabefelder, Hilfstexte, Schaltflächen usw. Hinzu kommt dann die begrenzte Anwendungslogik, die der Schnittstelle selbst zugerechnet wird (z.B. Plausibilitätsprüfungen, Berechnung bestimmter Feldinhalte et cetera). Da Formularbeschreibungssprachen primär zur Datenerfassung dienen, ist die Definition der Anbindung an eine korrespondierende Anwendungslogik hingegen vergleichsweise eingeschränkt und weitaus weniger flexibel als bei den zuvor beschriebenen Ansätzen. Dennoch ist es auch hier möglich, Aufrufe entfernter Anwendungskomponenten in die Schnittstellendefinition zu integrieren und so eine entsprechende Ankopplung zu schaffen.

Die Visualisierung der so definierten Eingabemasken erfolgt zumeist in speziellen Formularbetrachtern, die auch als *forms processors* bezeichnet werden. Wie im Worldwide Web werden Formulardefinitionen von einem entfernten Server geladen und im lokalen Prozessor dargestellt. Eingebettete Programmroutinen werden bei Bedarf durch den Prozessor ausgeführt bzw. an die entfernte Anwendung weitergeleitet. Auf dieselbe Weise werden auch schließlich die Eingaben in die Maske an den Server zurückübertragen. Formulardefinitionen können oftmals in einzelne Segmente aufgeteilt werden, zwischen denen wie zwischen Seiten eines Formulars gewechselt werden kann, ohne dass hierfür ein entfernter Zugriff

```
<?xml version="1.0"?> <XFDL version="4.1.0">
  <page sid="Pythagorean_Theorem">
    <label>Pythagorean Theorem Form</label>
    <field sid="A">
      <label>Enter A:</label>
      <value>3</value>
    </field>
    <field sid="B">
      <label>Enter B:</label>
      <value>4</value>
    </field>
    <field sid="C">
      <label>Hypothenuse length C:</label>
      <value content="compute">
        <cval>5</cval>
        <compute>sqrt(A.value^"2"+B.value^"2")</compute>
      </value>
    </field>
  </page>
```

Abbildung 4.14: Formularbeschreibung mit eingebundener Berechnungsfunktion in XFDL (nach [BBG98])

notwendig wird.

4.2.4 Bewertung existierender Ansätze

Im vorigen Abschnitt wurden verschiedene Integrationsansätze auf Anwendungsebene vorgestellt. Während sich diese in ihrer Herangehensweise und ihrem Aufbau zum Teil sehr stark voneinander unterscheiden, weisen sie doch bestimmte Gemeinsamkeiten auf:

- Sie setzen auf der Anwendungsebene an, befassen sich also nicht mit Kommunikationsprotokollen auf niederer Ebene und setzen existierende Kommunikationsverbindungen voraus.
- Sie trennen Anwendungslogik von Präsentationslogik und schaffen so eine Abstraktionsschicht, die quasi als Sollbruchstelle fungiert und die Relokation der Darstellungskomponente einer Anwendung auf ein entferntes Gerät erleichtern.
- Sie definieren eine Beschreibungssprache, ein Austauschformat oder ein anders geartetes abstraktes Format, mit dem Benutzungsschnittstellen beschrieben und die Beschreibungen ausgetauscht werden können.
- Sie erreichen Plattformunabhängigkeit, indem sie die beschriebene Benutzungsschnittstelle in einem Laufzeitsystem, einem Browser zum Beispiel, zur Darstellung bringen und die auf dem Endgerät stattfindende Interaktion mit der dargestellten Schnittstelle über den Kommunikationspfad an das eigentliche Anwendungssystem zurückmelden.

Dennoch hat sich eben gezeigt, dass es sehr verschiedene Wege gibt, diese gemeinsamen Merkmale zu erreichen. Wie lassen sich die beschriebenen Ansätze bewerten? Dieser Abschnitt soll die Aussagen der vorigen Abschnitte nochmals verdichten, bevor dann in der folgenden Zusammenfassung versucht wird, Lehren zu ziehen und Schlussfolgerungen für den Entwurf eines eigenen Ansatzes abzuleiten.

Auf niedrigster Abstraktionsebene standen die beschriebenen Ansätze aus dem Bereich der betriebssystembasierten Integration. Das Ansetzen auf dieser systemnahen Stufe bietet den Vorteil, dass Anwendungen nicht auf die entfernte Nutzung hin angepasst werden müssen. Ohne dass sie explizite Schnittstellen für den entfernten Zugriff besäßen, wird ihre Benutzungsschnittstelle ausgelesen, in ein abstraktes Format überführt, an ein entferntes Gerät übertragen und dort in einem Browser angezeigt. Die beschriebenen Systeme X11, RDP, VNC und Puppeteer zeigten, dass hierbei auch innerhalb der Betriebssystemschicht auf verschiedenen Ebenen angesetzt werden kann, wobei der Grad variiert, zu dem das Betriebssystem selbst modifiziert werden muss. In der Notwendigkeit hierzu liegt ein wesentlicher Nachteil dieses Vorgehens. Nur durch aufwendige Modifikationen lässt sich erreichen, dass eine solche Anpassung des Betriebssystems gelingt. Wie im Falle von X11, RDP oder auch ICA muss die Anpassung sogar auf Quelltextebene des Betriebssystems erfolgen.

Nachdem diese Änderungen jedoch erfolgt sind, bietet die betriebssystembasierte Integration den Vorteil einer sehr originalgetreuen Darstellung auf dem entfernten Gerät. Wie RDP, ICA und VNC zeigen, ist es möglich, von einem PDA genauso in gewohnter Weise auf die Textverarbeitung zuzugreifen[8], wie es von einem Laptop oder einem leistungsstarken PC aus der Fall ist. Im Vergleich der verschiedenen Ansätze zeigt sich, dass das Wissen um die Semantik der Schnittstelle bei der optimierten Übertragung und Darstellung von Vorteil ist. Während bei X11, RDP und ICA die Identifikation der einzelnen Elemente der Benutzungsschnittstelle verwendet werden kann, um jeweils passende Kompressions– und Übertragungsverfahren einsetzen zu können, besitzt VNC mit seinem geringen Wissen über die Struktur der Schnittstelle nur die Möglichkeit zur vergleichsweise eingeschränkten Optimierung auf Pixelebene, d.h. indem die gesamte Benutzungsschnittstelle wie ein nicht weiter analysiertes Bild aufgefasst wird.

Ansätze aus dem Bereich der betriebssystembasierten Integration zeichnen sich dadurch aus, dass die entfernte Darstellung der Benutzungsschnittstelle in einem speziellen Browser geschieht. Dieser ist hochgradig auf den jeweiligen Ansatz optimiert und auch nur in diesem Zusammenhang nutzbar. Ein ICA Browser kann also nicht zur Darstellung von WML Seiten verwendet werden, ein VNC Viewer nicht zum Zugriff auf einen RDP Server. Die Browser stellen jeweils vergleichsweise komplexe Anwendungsprogramme dar, die bestimmte Anforderungen hinsichtlich verfügbarem Speicher bzw. verfügbarer Rechenleistung an ihre Laufzeitumgebung stellen. Insbesondere jedoch sind sie soweit als proprietär anzusehen, als sie nicht in geschlossenen Endgeräten wie z.B. Mobiltelefonen verfügbar sind. Wichtige Grundvoraussetzung für die Nutzung der betriebssystembasierten Integration im Zugriff auf entfernte Anwendungssysteme ist also zunächst,

[8] Natürlich mit der Einschränkung einer eventuell fehlenden Tastatur, eines vergleichsweise kleinen Sichtbereiches usw.

dass es sich bei dem verwendeten Endgerät um ein *offenes* System handelt, dass dieses recht leistungsfähig ist und über genügende grafische Fähigkeiten zur Darstellung der übertragenen Benutzungsschnittstelle verfügt. Hierin liegt nämlich ein wichtiger Nachteil der betriebssystembasierten Integration: Eigentlich alle Ansätze aus dieser Gruppe basieren auf der detailgetreuen Übertragung *grafischer* Strukturen der Benutzungsschnittstelle. Die Realisierung entsprechender Browser auf einfachen Mobiltelefonen, PDAs et cetera ist insofern oft nicht möglich, die Umsetzung in gesprochene Sprache sogar gänzlich ausgeschlossen.

Ein weiteres Problem der betriebssystembasierten Ansätze liegt darin, dass eine optimierte Übertragung nur dann möglich ist, wenn Wissen über die Semantik der Benutzungsschnittstelle (d.h. die grafische Betriebssystemoberfläche) vorliegt. Ist dieses wie bei X11, ICA oder RDP der Fall, so kann auf die Übertragung bestimmter Anteile der Schnittstelle verzichtet werden, die als wiederkehrend bekannt sind (z.B. systemeigene Anteile wie Laufleisten, Systemknöpfe usw.). In anderen Fällen können Anteile zwischengespeichert und aus dem lokalen Speicher heraus wiederverwendet werden, wie in dem Fall, wenn zeilenweise in einem Textdokument geblättert wird. Können einzelne Zeilen unterschieden werden, so kann der erneute Textaufbau durch verschieben einzelner Zeilen auf dem Bildschirm erfolgen, ohne dass der gesamte Bildschirminhalt pixelweise neu übertragen werden müsste. Steht ein solches Wissen nicht zur Verfügung, so kann hingegen nur auf der Basis des Bildschirminhalts optimiert werden, was nur in begrenztem Maße möglich ist, wie der Ansatz von VNC zeigt.

Aus der Sicht des Anwendungsentwicklers stellt die betriebssystembasierte Integration eine sehr attraktive Möglichkeit zur Integration mobiler Geräte dar, da eine explizite Anpassung des Anwendungssystems nicht notwendig ist. Ist weiterhin gewährleistet, dass eine Kommunikationsverbindung während der Nutzung der Anwendung jederzeit vorhanden ist und die verwendeten Endgeräte die zuvor genannten Bedingungen hinsichtlich Leistungsfähigkeit und Offenheit erfüllen, so bringt diese Form der Integration mit die besten Ergebnisse bezüglich Darstellungstreue über verschiedene Gerätetypen.

In der akkuraten Reproduktion der Benutzungsschnittstelle liegt gerade in mobilen Kontexten ein wichtiges Problem versteckt, da die Benutzungsschnittstelle einer Anwendung oftmals hinsichtlich des vom Anwendungsentwickler vorgesehenen Nutzungskontextes optimiert ist. So könnte die Anordnung der Bedienelemente eine bestimmte Bildschirmgröße voraussetzen, die Schnittstelle nur bei Verwendung einer Maus effizient bedienbar sein oder der farbliche Kontrast so gewählt sein, dass die Darstellung auf einem einfachen LCD unbefriedigende Ergebnisse bringt. Auch ist die Dateneingabe auf Mobilgeräten oft sehr mühsam. Eine Benutzungsschnittstelle für mobile Geräte sollte daher die Erfassung von Informationen auf ein Minimum begrenzen und nach Möglichkeit durch Auswahl aus einer Menge von Alternativen realisieren. Auch insofern stößt die detailgetreue Darstellung betriebssystembasierter Integrationsansätze an ihre Grenzen, sofern das verwendete Endgerät in seiner Architektur und Bedienung zu stark vom herkömmlichen Personalcomputer abweicht.

Schließlich stellt der Browser einen festen Rahmen für die Darstellung der Benutzungsschnittstelle dar, welcher der Anwendung kaum Möglichkeiten zur Interaktion mit den Ressourcen des verwendeten Endgerätes lässt. Besondere Merkmale des Gerätes wie die etwaige Stiftbenutzung oder Schrifterkennung, die Nutzbar-

keit von Daten über die verschiedenen auf dem Gerät vorhandenen Anwendungen usw. können nur sehr schwer einbezogen werden, lokale Datenspeicher nur selten erkannt und genutzt werden usw. Die ohnehin recht begrenzten Möglichkeiten des mobilen Gerätes können somit letztlich nicht voll ausgeschöpft werden. Die betriebssystembasierte Integration ist insgesamt also nur in eng begrenzten Anwendungsszenarien und für ganz bestimmte Endgeräte sinnvoll.

Ähnlich wie betriebssystembasierte Ansätze bieten auch applikationsserverbasierte Integrationsansätze nur bedingte Ansatzpunkte für eine feingranulare Anpassung eines Anwendungssystems an die verwendeten Endgeräte. Da in diesem Fall zum einen auf den festgefügten Rahmen einer Browserumgebung gesetzt wird und grundsätzlich die gesamte Anwendungslogik im Zentralsystem verbleibt, können möglicherweise nicht alle Eigenschaften des Gerätes ausgenutzt werden. Ein weiterer Nachteil der applikationsserverbasierten Integration ist zudem, dass sie zumindest in ihrer Grundform für die gesamte Nutzungsdauer des Anwendungssystems eine Kommunikationsverbindung zwischen Endgerät und Applikationsserver voraussetzt. Dieses führt gerade in mobilen Umgebungen zu hohen Kosten oder ist überhaupt nicht möglich, wenn die notwendige Kommunikationsinfrastruktur am gegenwärtigen Aufenthaltsort nicht verfügbar ist. Es wurden verschiedene Ansätze beschrieben, die zum Teil der Grundform eines Applikationsserversystems entsprachen (z.B. Portal To Go) oder diese auf die eine oder andere Art abwandeln, indem sie den Applikationsserver nur zum Datenaustausch verwenden, die eigentliche Bearbeitung der Daten auf dem Endgerät jedoch in einem lokalen Anwendungsprogramm stattfindet (z.B. DataX).

Bei DataX und MDSS handelt es sich um deutlich „datenbankzentrierten“Ansätze. Es wird in beiden Projekten vornehmlich darauf Wert gelegt, die Erfassung von Daten bzw. deren Darstellung zu ermöglichen; der Schwerpunkt liegt also auf typischen Funktionen eines Zugriffswerkzeugs zu einer Datenbank. Während im Kontext dieses Anwendungsfeldes eine maximale Flexibilität erreicht werden kann, eignet sich der Ansatz doch nur für eine geringe Anzahl von Anwendungen. Hinzu kommt, dass durch die im Design verankerte Notwendigkeit einer speziellen Browserkomponente auf dem Endgerät eine nur bedingte Tauglichkeit des Ansatzes für geschlossene Geräte begründet liegt.

MDSS unterscheidet sich also vor allem dadurch von DataX, dass bezüglich der Verarbeitung von Daten auf dem Endgerät sowie dem Diensterbringer eine größere Flexibilität herrscht. Daten können auf dem Endgerät nicht nur in einer bestimmten Anwendung erfasst bzw. modifiziert und auf dem Server nicht nur in einer bestimmten Datenbank abgespeichert und verarbeitet werden. Über die Verwendung von Adaptern ergibt sich insbesondere auf der Seite des Endgerätes der Vorteil, dass jeweils native Anwendungen zum Einsatz kommen und so die Eigenschaften des Gerätes besser ausgenutzt werden können. Unterstützen die auf dem Gerät bereits vorhandenen Anwendungen zudem standardisierte Schnittstellen wie ODBC, kann darüber hinaus auf proprietäre Klienten verzichtet werden und stattdessen auf bereits vorhandene Software zurückgegriffen werden. Leider ist dieses gerade bei mobilen Endgeräten selten der Fall. Insofern kann MDSS erst dann einen signifikanten Vorteil gegenüber DataX bringen, wenn der Adapter selbst auf den Datenspeicher der unterstützten Anwendung zugreifen und somit dynamisch und ohne deren Einwirkung einen Datenabgleich möglich machen kann. Die Ausführungen von Butrico et al. in [BCG+00] lassen zwar den Schluss zu, dass eine solche Erweiterung von MDSS geplant ist, derzeit ist der

Ansatz jedoch wie DataX auf speziell zu realisierende Anwendungen angewiesen.

Das in Portal To Go gewählte Vorgehen entspricht dagegen dem klassischen Ansatz eines Applikationsservers. Darstellung, Verarbeitung und Speicherung von Daten werden auf drei oder mehr Schichten verteilt, wobei Portal To Go selbst hauptsächlich die abstrakte Zwischenschicht realisiert, die Einbindung von Endgeräten kontrolliert und die Transformation generischer Beschreibungen von Informationen in konkrete Repräsentationen steuert. Die eigentliche Bereitstellung der Daten findet durch externe Anwendungen statt, die über Adapter in den Applikationsserver eingebunden werden.

Portal To Go zeichnet sich durch ein hohes Maß an Flexibilität aus. Der definierte XML Dialekt bietet eine genügende Ausdrucksmächtigkeit, um die mit seiner Hilfe kodierten Informationen auf eine sehr breite Palette von Endgeräten zu übertragen. Daneben wird durch das im OracleMobile Online Studio hinzugefügte Konzept der Funktionsmodule die Erweiterung des Applikationsservers um weitere externe Komponenten formalisiert und weitestgehend vereinfacht. Die Aufspaltung einer Informationsquelle in Master Service, Adapter und externe Anwendung erlaubt die Erschließung prinzipiell beliebiger Quellen. Sofern ein Adapter für die jeweilige Anwendung entworfen werden kann, der die Überführung in die interne Repräsentation Portal–To–Go XML bzw. MobileXML übernimmt, ist so eine Brücke zwischen existierenden Anwendungen (sog. „Legacy Applications“) und den Endgeräten möglich, für die ein Transformer existiert. Hierbei liegt letztlich jedoch die Schwierigkeit von Portal To Go und OracleMobile: Für jede externe Anwendung muss ein solcher Adapter realisiert werden, was im Zweifelsfalle eine recht komplexe Aufgabe sein kann. Andererseits hängt es von der Qualität des jeweiligen Adapters ab, wie gut eine Datenquelle für die Nutzung in Portal To Go erschlossen werden kann und inwieweit diese zu einem homogenen Portal beitragen kann. Dort, wo die Anwendung keine expliziten Schnittstellen bereitstellt, ist der Entwurf von Adaptern ohne weiterführende Konzepte sogar kaum möglich. Um eine wirklich optimale Unterstützung einer bestimmten Anwendung zu erhalten, ist es daher ratsam, bei der Erschließung der Anwendungsschnittstelle von einer großen und bei Anwendungsentwurf nicht endgültig bekannten Menge von Endgeräten auf Ansätze nach dem Prinzip der Entwurfszeit–Transformation zurückzugreifen.

Einen recht einfachen Ansatz stellen in dieser Gruppe die vorgestellten Formularbeschreibungssprachen wie XFDL oder XFA dar. Diese Sprachen wurden mit einem ganz bestimmten Anwendungsgebiet im Sinn entworfen, nämlich der Umsetzung zuvor nicht elektronischer Datenerfassung im Formularwesen in EDV basierte Lösungen. Die Optimierung auf einen bestimmten Kontext führt zunächst dazu, dass Sprache wie XFDL oder XFA sich sehr gut für die Definition von Eingabemasken eignen. Ein bestimmtes Segment möglicher Benutzungsschnittstellen grafischer Anwendungen lässt sich damit sehr effizient und gut beschreiben und in eine Abstraktionsschicht umsetzen. In diesem Segment bieten Formularbeschreibungssprachen durchaus Vorteile hinsichtlich Effizienz, Plattformunabhängigkeit oder Ausgereiftheit des jeweiligen Produkts. Leider bieten Formularbeschreibungssprachen nur einen sehr eingeschränkten Satz an Beschreibungselementen. Sie lassen sich somit nicht zur Beschreibung beliebiger grafischer Schnittstellen heranziehen, ohne dass dafür die jeweilige Beschreibungssprache umfassend erweitert werden müsste. Während dieses zwar prinzipiell möglich wäre, hat sich gezeigt, dass die betrachteten Ansätze in ihrem

Aufbau vergleichsweise inflexibel bezüglich derartiger Modifikationen sind. Sie beruhen zumeist auf einem vorgefertigten Formularbetrachter, der nicht in der gewünschten Weise angepasst werden und hinsichtlich der Unterstützung weiterer GUI Elemente erweitert werden kann. Darüber hinaus existieren die notwendigen Formularbetrachter nur für eine sehr geringe Zahl von Plattformen. Während dieses zwar für den eigentlichen Einsatzzweck durchaus akzeptabel sein mag, stellt eine solche Einschränkung für den im Kontext dieser Arbeit geplanten Einsatzzweck ein weitgehend prohibitives Merkmal dar. Schließlich erwiesen sich die untersuchten Formularbetrachter als zum Teil sehr anspruchsvoll hinsichtlich bereitzustellender Ressourcen, so dass deren Einsatz in einem mobilen Umfeld kaum realistisch sein dürfte. Aber auch die Herauslösung der verwendeten Beschreibungssprache, deren Erweiterung und schließlich deren Nutzung in einem eigenen Betrachter muss als nicht sinnvoll erachtet werden: Der hierfür zu veranschlagende Aufwand stünde in keinem Verhältnis zu dem zu erwartenden Gewinn.

Die User Interface Markup Language stellt hingegen einen sehr interessanten Ansatz dar. Sie wurde speziell mit dem Ziel entwickelt, eine Abstraktionsschicht zwischen grafischer Benutzungsschnittstelle und Anwendungslogik zu schaffen und die Präsentationsschicht auf einem beliebigen entfernten Gerät darstellen und nutzen zu können. Wie bei den oben genannten Formularbeschreibungssprachen erfolgt die Definition der Schnittstelle in einer allgemeinen Notation auf Basis von XML, die dann in einem Browser auf einem entfernten Rechnerknoten dargestellt werden kann. Zusätzlich kann auch die Übersetzung in eine herkömmliche Programmiersprache erfolgen und die Schnittstelle so gemeinsam mit der Applikationslogik zu einem monolithischen Anwendungsprogramm verknüpft werden.

Beim Entwurf von UIML wurde besonderer Wert darauf gelegt, die Schnittstelle in einer Form beschreibbar zu machen, die eine Optimierung auf die besonderen Kriterien des verwendeten Endgerätes erlaubt. Logische Zusammenhänge zwischen Elementen des GUI können definiert werden und so Module identifizierbar gemacht werden, welche als Ganzes dargestellt oder unterdrückt werden können, ganz nach Maßgabe für das jeweilige Gerät. Außerdem können Alternativen formuliert werden, so dass nicht nur die Reduktion der Schnittstelle durch komplettes Weglassen ganzer Teile möglich ist, sondern vielmehr auch die Form der Darstellung wechseln kann. Auf diese Weise kann die Ausdrucksmächtigkeit eines jeden Endgerätes optimal unterstützt werden und so eine sehr gute Anpassung der Anwendung an das Endgerät erfolgen. Letztlich besitzt UIML somit sogar die Möglichkeit zur Abbildung von Benutzungsschnittstellen auf nicht grafische Repräsentationen. Neben der Umsetzung in klassische GUIs kann also auch die „Darstellung“einer Schnittstelle über ein einfaches Telefon oder eine intelligente Mailbox erfolgen.

Eine weitere Stärke von UIML liegt in der Möglichkeit, die Umsetzung der abstrakten Schnittstellenbeschreibung in eine konkrete Repräsentation nicht durch einen Browser auf dem Endgerät durchführen zu lassen, sondern diese Aufgabe einem Applikationsserver zuzuordnen. Dieser könnte dann beispielsweise mit einem Webserver gekoppelt werden und sodann eine Umsetzung der Schnittstellenbeschreibung in herkömmliches HTML vornehmen, das dann durch den Webserver an das zugreifende Endgerät weitergeleitet werden kann. Auf diese Weise können auch diejenigen Geräte UIML verwenden, deren geschlossenes De-

sign oder deren Ressourcenknappheit die Verwendung eines besonderen UIML Browsers unmöglich macht.

Während sich UIML also zunächst sehr gut für den in dieser Arbeit betrachteten Anwendungskontext zu eignen scheint, liegt ein Problem darin, dass die Flexibilität des Ansatzes zu einer vergleichsweise hohen Komplexität geführt hat. Die Erstellung der Schnittstellenbeschreibung mit ihren verschiedenen Strukturelementen, den Alternativen und Ereignisdefinitionen erfordert ein hohes Maß an Aufwand und lässt sich kaum automatisieren. Zwar lassen sich durch den Einsatz von domänenspezifischen Varianten gewisse Vereinfachungen erreichen, wobei hierdurch jedoch auch die Flexibilität des Ansatzes deutlich eingeschränkt wird. Schließlich muss die Abstraktionsschicht zwischen Präsentation und Applikationslogik direkt auf Ebene des Quellcodes in die Anwendung eingefügt werden. Die Nutzung von UIML zur Realisierung einer Abstraktionsschicht bei Anwendungen, die nicht im Quellcode zur Verfügung stehen, ist somit ausgeschlossen.

Weniger generisch und auf den Einsatz zur Beschreibung von grafischen Benutzungsschnittstellen beschränkt ist der Ansatz von XUL, der Extensible User interface Language. XUL entstammt einem konkreten Anwendungsfall und wurde demgemäß diesbezüglich optimiert. Wie UIML ermöglicht auch XUL die Trennung der Präsentationslogik von der eigentlichen Applikationslogik, wobei in diesem Fall nicht die Übertragung der Schnittstelle auf ein entferntes Endgerät und die Nutzung der Anwendung über eine Netzwerkverbindung im Vordergrund stand. Vielmehr sollte der Entwicklungsprozess alternativer Schnittstellen vereinfacht und die Anpassung an Geräte mit spezifischen Fähigkeiten ermöglicht werden, ohne dass hierfür Änderungen an der Applikationslogik erforderlich sein würden.

Wie UIML basiert auch XUL auf einer XML Notation. Dabei steht die definierte Markup Sprache weitaus weniger isoliert dar, sondern bedient sich selbst weiteren Beschreibungssprachen, die im Umfeld des Projektes anzutreffen sind, in dessen Rahmen XUL entwickelt wurde. So werden alle Aspekte der tatsächlichen Darstellung durch Cascading Stylesheets abgedeckt, die Beschreibung aller Inhalte findet mittels HTML4 statt, und die Verdrahtung der Präsentation mit einer korrespondierenden Anwendungslogik erfolgt über in die Schnittstellenbeschreibung integrierte Aufrufe in JavaScript. Während sich auf diese Weise eine gute (und im Projektrahmen ausgiebig erprobte) Effizienz und vergleichsweise flache Lernkurve bei der Erstellung von Schnittstellen erzielen lässt, stellt diese feste Einbettung in den Projektkontext eine Einengung der flexiblen Nutzung des Ansatzes außerhalb der Beschreibung von Schnittstellen für Webbrowser dar. Obwohl sich durch den Einsatz von JavaScript zumindest noch die räumliche Entkopplung der Präsentationsschicht und die entfernte Steuerung der Anwendungslogik durch entsprechende Aufrufe realisieren ließe, fehlen XUL doch wichtige Merkmale, die seinen Einsatz im Zusammenhang mit der Nutzbarmachung von Anwendungsschnittstellen auf beliebigen Mobilgeräten möglich machen. So muss auf dem Endgerät in jedem Fall mit der *Gecko Engine* eine spezifische Laufzeitumgebung existieren. Die Nutzung für geschlossene Endgeräte ist somit nicht möglich. Auch müssen weite Teile der Anwendungslogik in einer XUL Definition selbst realisiert werden, wodurch das eigentliche Abstraktionsziel stark verwässert und die Komplexität der Schnittstellendefinition erhöht wird. Schließlich bedingt auch XUL die explizite Anpassung der Anwendungs-

logik an die Nutzung mit einer entkoppelten Präsentationsschicht. Somit gilt wie für UIML, dass eine Nutzung nur dann möglich ist, wenn der Quellcode der Anwendung vorliegt und entsprechend modifiziert werden kann. Während XUL also durchaus zur Beschreibung von grafischen Benutzungsschnittstellen einsetzbar ist, stellt es sich insgesamt als zu eingeschränkt heraus, um als Basis eines generischen Integrationsansatzes zu dienen, insbesondere, da die Einbeziehung sehr konkreter Anteile (JavaScript usw.) die diesbezügliche Erweiterung und Anpassung als nicht sinnvoll erachten lässt.

Universal Lightweight Clients (ULC) stellen insofern einen deutlichen Unterschied zu den zuvor beschriebenen Ansätzen dar, als in diesem Fall keine explizite Notation auf XML Basis definiert wird, die zur Beschreibung von Benutzungsschnittstellen verwendet wird. Vielmehr wird dem traditionellen Ansatz des Entwurfs von grafischen Benutzungsschnittstellen mittels objektorientierter Programmiersprachen gefolgt, nachdem durch fortgesetzte Aufrufe einer GUI Bibliothek die interne Repräsentation der Schnittstelle sukzessiv zur Laufzeit konstruiert wird. Bemerkenswert ist hierbei zunächst nur, dass die so erstellten GUI Elemente (sog. faceless widgets) keine direkte Ausgabe auf dem Bildschirm erzeugen, sondern ihrerseits lediglich Aufrufe an eine entfernte Laufzeitumgebung erzeugen können, in der dann die eigentliche Visualisierung stattfindet. Auf diese Weise wird die Trennung zwischen Präsentationslogik und Applikationslogik auf einem sehr niedrigen Abstraktionsniveau erreicht.

Wichtiges Entwurfsziel bei ULC war vor allem die effiziente Kommunikation zwischen der Applikationslogik und der über das Netzwerk angekoppelten Präsentationslogik. Mittels tiefgreifender Optimierungsverfahren wurde die zügige Nutzung der entfernten Anwendung auch unter widrigen Bedingungen hinsichtlich Bandbreite und Latenz der zwischen Endgerät und Anwendungssystem liegenden Netzwerkstrecke ermöglicht. Durch besondere Cachingstrategien, sinnvolles Verzögern oder Vorziehen von anfallenden Datenübertragungen sowie die Entkopplung der Kommunikation zwischen den Faceless Widgets und ihren konkreten Repräsentanten auf dem Endgerät wurde es darüber hinaus möglich, eine Anwendung auch dann noch über eine ansprechende grafische Benutzungsschnittstelle zu verwenden, wenn dieses mit herkömmlichen Verfahren nicht länger sinnvoll möglich wäre.

Dabei war von ganz besonderem Interesse im Zusammenhang dieser Arbeit die Nutzung eines als *Screen–Scraping* bezeichneten Verfahrens, mit dessen Hilfe existierende Terminalanwendungen zur Laufzeit analysiert und deren Ausgabe in die Applikationslogik von ULC zur weiteren Verarbeitung und grafischen Aufbereitung in Faceless Widgets übertragen werden kann. Auf entgegengesetztem Weg können die vom Anwender ausgelösten Ereignisse in der Anwendungslogik in Tastendrucke umgesetzt werden, die dann wiederum an die Terminalanwendung weitergeleitet und dieser wie eine tatsächliche Bedienung über die Tastatur vorgespielt werden können. Im Gegensatz zu den anderen beschriebenen Ansätzen erlauben es die Universal Lightweight Clients daher auch, in sich geschlossene Anwendungen anzubinden, deren Quelltext nicht zur Verfügung steht und die daher für eine direkte Modifikation ausgeschlossen sind. Dieses gilt allerdings nur für die zugrunde liegenden Anwendungs– bzw. Businessobjekte. Die ULC Anwendung muss weiterhin unter Verwendung einer herkömmlichen Programmiersprache erstellt und manuell in ein lauffähiges Programm übersetzt

werden. Ein vollständiger Verzicht auf das Eingreifen auf Ebene der Programmiersprache ist auch bei ULC nicht möglich.

Die Entkopplung bei ULC direkt auf der Basis des Quelltextes wirkt sich positiv auf den zu erzielenden Grad an Unterstützung der einzubindenden Businessobjekte aus. So kann während einer Übergangszeit die mittels Screen–Scraping erfragte Information in der geeigneten Weise durch eine Zwischenschicht von Faceless Widgets umgesetzt werden, bevor das geschlossene Businessobjekt schließlich ganz durch eine auf Faceless Widgets beruhende Anwendung abgelöst wird.

Die ökonomische Verwendung von Netzwerkressourcen sorgt schließlich für eine gute Skalierbarkeit des Ansatzes. Indem die zur Optimierung und zur Extraktion von Informationen aus bestehenden Anwendungen verwendeten Strategien durch eine Applikationsschicht zwischen Präsentationslogik und Datenhaltung erfolgt, werden hohe Anforderungen an die Rechenkapazität des Endgerätes vermieden. Die Nutzung wird daher auch auf schwach angebundenen und weniger gut ausgestatteten Geräten, z.B. kleineren Mobilsystemen möglich.

Andererseits macht die Betrachtung des Ansatzes sehr deutlich, dass hierbei eine räumliche Trennung zwischen Präsentation und Anwendungslogik im Vordergrund stand, nicht jedoch die Nutzung der Anwendung von *beliebigen* Endgeräten. So ist man beim Einsatz von ULC zunächst einmal auf einen speziellen Browser angewiesen, in dem die konkrete Repräsentation der Faceless Widgets dargestellt wird und die auf Clientseite angesiedelten Optimierungsstrategien realisiert werden. Während somit die Nutzung von geschlossenen Geräten wie herkömmlichen Mobiltelefonen ausgeschlossen ist, stellt dieses sicherlich keine besondere Einschränkung im Vergleich zu den anderen vorgestellten Ansätzen dar: Auch diese beruhen zumeist auf der Verwendung eines speziellen Laufzeitsystems zur Anzeige der entkoppelten Benutzungsschnittstelle der Anwendung. Im Gegensatz zu anderen Ansätzen wie z.B. UIML besteht bei ULC jedoch nicht die Möglichkeit zur dynamischen Anpassung der Schnittstelle an die besonderen Anforderungen eines bestimmten Geräts. Es kann in dem Sinne keine abstrakte Beschreibung der Schnittstelle erfolgen, die dann abhängig vom Endgerät konkretisiert wird. Hierfür muss jeweils eine eigene ULC Applikation entworfen werden, durch die dann eine Umsetzung in der für das Endgerät optimalen Weise erfolgen kann. Mit jedem weiteren Endgerät, das unterstützt werden soll, muss dann auch wiederum eine neue Anwendung entwickelt werden.

Auch ist bei ULC zunächst nur die Nutzung von GUI Elementen der im Zusammenhang mit Java verwendeten SWING bzw. AWT Bibliotheken möglich. Der notwendige Aufwand für die Verwendung von ULC in Mobiltelefonen, einfachen PDAs usw. ist damit als äußerst hoch einzuschätzen. Der Einsatz von ULC in einem heterogenen Umfeld erscheint damit als nicht sinnvoll. Schließlich wird trotz Screen–Scraping keine dynamische Anbindung möglich, ohne dass hierfür ein aufwendiger Entwicklungsprozess neuer Software notwendig werden würde. Jede Anwendung, die in ULC entfernt ausgeführt werden soll, muss unter Verwendung von Faceless Widgets in der Programmiersprache Java neu realisiert werden. Auch hierin teilt ULC also einen Nachteil aller Verfahren, die sich durch Entwurfszeit–Transformation auszeichnen.

Welche Eigenschaften soll die zu realisierende Integrationsumgebung aufweisen? Welche Merkmale lassen sich aus der Beschreibung und Bewertung der in diesem Kapitel angeführten Beispiele als wünschenswert bzw. notwendig ableiten?

Idealerweise sollte der in dieser Arbeit verfolgte Ansatz natürlich möglichst viele Vorteile der beschriebenen Konzepte in sich vereinen. So sollte der Ansatz insbesondere:

- durch die Trennung zwischen Präsentationslogik und Applikationslogik die Nutzung der Anwendung auf sehr ressourcenschwachen Zugriffsgeräten ermöglichen,
- in seinem Entwurf möglichst keine expliziten Vorgaben hinsichtlich verwendbarer Kommunikationskanäle formulieren,
- die Nutzung von Anwendungen auch auf solchen Geräten ermöglichen, auf denen die Nutzung eines speziellen Browsers durch ein geschlossenes Gerätedesign unmöglich ist.
- Er sollte des Weiteren den entfernten Zugriff auf Anwendungen entfernt ermöglichen, die nicht explizit hierfür realisiert wurden, wobei keine grundlegenden Modifikationen des zugrunde liegenden Betriebssystems notwendig werden sollte, d.h.
- die Adaption an Endgeräte bzw. Repräsentationen sollte möglich sein, ohne dass hierfür die Anwendung bzw. das Integrationssystem grundsätzlich modifiziert werden müssen.
- Schließlich sollte ein generisches Zwischenformat zur Beschreibung von Benutzungsschnittstellen verwenden werden, das hinsichtlich der Übertragung in gewünschte Zielrepräsentationen möglichst wenige Einschränkungen besitzt.

Für das verwendete Zwischenformat sollte hierbei gelten, dass es:

- möglichst effizient transformierbar ist und gängigen Zielrepräsentationen mobiler Endgeräte so weit entspricht, dass existierende Werkzeuge zur Transformation zum Einsatz kommen können,
- die Unterstützung spezifischer Merkmale des Gerätes möglich macht und dass hierbei insbesondere die Darstellungsfähigkeiten sowie Ein– und Ausgabefähigkeiten des Gerätes jeweils zu berücksichtigen sein sollten und schließlich, dass der Ansatz
- keine Festlegung auf eine bestimmte Zielrepräsentation enthalten sollte, sondern diese vielmehr jederzeit abgeändert werden können.

Für den letzten Punkt soll insbesondere gelten, dass der Wechsel der Zielrepräsentation nicht nur ohne Modifikationen des Anwendungssystems oder der Integrationsumgebung möglich sein soll, sondern dass dieser vielmehr auch während der Laufzeit der Anwendung noch durchführbar sein sollte, wobei der Anwendungszustand erhalten bleiben sollte. Konkret soll es also möglich sein, je nach Bedarf das Endgerät wählen zu können, von dem der Zugriff auf die Anwendung erfolgt, ohne dass hierfür ein Neustart der Anwendung nötig werden würde.

In den folgenden Abschnitten sollen die Konzepte beschrieben werden, die zur Erfüllung der genannten Forderungen entwickelt wurden und die die Grundlage der realisierten Integrationsumgebung bilden.

4.3 Einbindung von Dienstnehmern — Integration durch Abstraktion

Wie sich aus den Ausführungen des vorigen Abschnitts schließen lässt, stellt die Trennung von Applikationslogik und Präsentationslogik einen wichtigen Bestandteil eines jeden Integrationsansatzes dar, der die Unterstützung ressourcenschwacher Endgeräte zum Ziel hat. Es zeigte sich, dass es eine ganze Reihe verschiedener Wege gibt, diese Trennung zu erreichen. Allen gemeinsam ist jedoch das Prinzip, den Anwendungszustand in einer bestimmten Repräsentation zu kodieren, ihn in dieser Repräsentation zum Endgerät zu übertragen und dort zur Anzeige zu bringen.

Auch das in dieser Arbeit entworfene Integrationskonzept beruht auf diesem Prinzip. Hierbei greift der Ansatz auf einem sehr hohen Abstraktionsniveau an. Es wird nicht (wie bei RDP oder ICA) in die Systembibliotheken des Betriebssystems eingegriffen. Auch wird nicht der Anwendungscode selbst modifiziert, wie dieses bei der applikationsserverbasierten Integration (z.B. DataX) und der Entwurfszeit–Transformation (wie etwa bei UIML oder XUL) der Fall ist. Stattdessen wird ähnlich wie bei VNC die Rolle eines Betrachters eingenommen und die grafische Schnittstelle der Anwendung zur Laufzeit analysiert. Während VNC jedoch keine Informationen über die Struktur der Schnittstelle sammelt, sondern lediglich die grafische Repräsentation der gesamten grafischen Arbeitsfläche komprimiert und überträgt, geht der hier vorgestellte Ansatz weiter: Es wird eine tiefgehende Analyse der Benutzungsschnittstelle einer bestimmten Anwendung durchgeführt, welche die Elemente der Schnittstelle, also Schaltknöpfe, Textfelder, Tabellen usw. in ihrem Typ und in ihrer Beziehung untereinander erkennt. Die so gewonnen Informationen werden in eine allgemeine Repräsentation in XML übertragen und an den Transformationskern der Integrationsumgebung übergeben. Hier kann die allgemeine Beschreibung dann anhand zuvor definierter Transformationsregeln in eine konkrete Repräsentation zurück überführt und schließlich auf einem Endgerät angezeigt werden. Der gesamte Prozess von Analyse bzw. Generierung der allgemeinen Beschreibung über deren Transformation bis hin zur Übertragung an ein Endgerät erfolgt dabei in Echtzeit und ohne Beeinflussung der Anwendung selbst.

Das dieser Arbeit zugrunde liegende Konzept beruht also darauf, dass die eigentlich konkrete grafische Schnittstellenrepräsentation einer Anwendung zur Laufzeit in eine abstrakte Zwischenrepräsentation überführt und diese nach einer Transformation in die für das verwendete Endgerät geeignete Repräsentation wieder konkretisiert wird. Der Ansatz soll daher als „Integration durch Abstraktion“ bezeichnet werden. *Abstrakt* bedeutet hierbei, dass die Zwischenrepräsentation (weitestgehend) auf Bezüge zur ursprünglichen Darstellungsform einerseits und auf Bezüge zu möglichen Darstellungsformen auf dem Endgerät andererseits verzichtet. Das gewählte Zwischenformat ist daher als neutral zu bezeichnen. Für dieses Format existiert demgemäß auch keine direkte Möglichkeit zur Darstellung. Erst durch die Verknüpfung mit einem bestimmten Satz von Transformationsregeln und der Übersetzung anhand dieses Satzes ergibt sich wieder eine konkrete Repräsentation, die zur Darstellung in einem geeigneten Programm bestimmt ist. Durch die sich somit ergebende Entkopplung zwischen Ursprung und Ziel wird es möglich, auch im Nachhinein noch durch Hinzufügen

weiterer Transformationsregelsätze die Darstellung auf zuvor nicht unterstützten Endgeräten zu ermöglichen, ohne dass hierfür die Analyse der grafischen Benutzungsschnittstelle bzw. die Generierung der Zwischenrepräsentation angepasst werden müsste.

Was sind die logischen Bestandteile eines solchen Konzeptes? Im Wesentlichen lassen sich vier funktionale Komponenten identifizieren:

Analysekomponente : Der Analysekomponente fällt die Aufgabe zu, die grafische Schnittstelle der Anwendung zu untersuchen und deren Strukturen zu erfassen. Darüber hinaus obliegt es ihr, die gesammelten Informationen in eine allgemeine Beschreibung zu überführen, die zu einem bestimmten Zeitpunkt den Zustand der Schnittstelle darstellt.

Transformationskomponente : Die Transformationskomponente ist für die Überführung der allgemeinen Beschreibung in ein konkretes Format zuständig. Hierbei bedient sie sich eines zuvor bestimmten Regelsatzes, der für jedes in der allgemeinen Beschreibung mögliche Element (Knopf, Liste, Textfeld usw.) eine Übersetzungsregel in das konkrete Format enthält.

Ferndarstellungskomponente : Die aus der Transformation herrührende konkrete Repräsentation der Benutzungsschnittstelle muss an das Endgerät übertragen und dort dargestellt werden. Die hierfür zuständige Infrastruktur (z.B. bestehend aus Webserver, WAP Gateway, aber auch Sitzungsverwaltung o.ä.) kann logisch zu einer Komponente zusammengefasst werden, der Ferndarstellungskomponente.

Fernsteuerungskomponente : Aus der Darstellung der Schnittstelle auf dem Endgerät resultieren letztlich Befehle, die wieder an die Anwendung übertragen werden müssen. Wird auf dem Endgerät ein dort angezeigter Knopf gedrückt oder ein Textfeld gefüllt, so muss diese Veränderung der Schnittstelle auch an der Anwendung nachgeführt werden. Es muss also in die Ereignissteuerung der Anwendung eingegriffen werden. Diese Aufgabe obliegt der Fernsteuerungskomponente.

Die entstehende Abstraktionsschicht fußt letztlich auf zwei Bausteinen: Auf der einen Seite wird durch die Analysekomponente eine Abstraktion zwischen der Anwendung und der Integrationsumgebung realisiert. Indem lediglich die von der Analysekomponente generierten Repräsentationen an die Integrationsumgebung weitergegeben werden, ist die tatsächliche Funktionalität der Anwendung für die weitere Bearbeitung unerheblich. Auf der anderen Seite steht die Ferndarstellungskomponente. Sie enthält das notwendige Wissen über die Mechanismen, die zur Übertragung der konkreten Beschreibung zum Endgerät notwendig sind. Die Integrationsumgebung übergibt lediglich die aus der Transformation hervorgegangene konkrete Beschreibung an die Ferndarstellungskomponente. Die Bereitstellung auf einem Webserver, die Interaktion mit dem WAP Gateway usw. ist dann alleinige Sache dieser Komponente.

Die einzelnen Komponenten lassen sich zusätzlich durch einen Vergleich mit dem aus dem Entwurf grafischer Benutzungsschnittstellen bekannten Konzept der *Model–View–Controller* (MVC) Architekturen nach Wirfs–Brock (vgl. etwa [WBWW90] bzw. [You95]) einordnen. Nach diesem Konzept wird die grafische

Verarbeitung von Daten in drei funktionale Komponenten aufgegliedert, denen jeweils eine bestimmte Aufgabe zukommt:

Model : Das Modell (engl. Model) stellt die der Anwendung zugrunde liegenden Daten dar. Dieses können die Adressen einer Adressenverwaltung oder die Zahlen einer Tabellenkalkulation sein.

View : Die Sicht (engl. View) stellt eine bestimmte Sicht auf die Daten dar, wobei die Zahl der Sichten prinzipiell nicht beschränkt ist. Sichten auf eine Adressenkartei könnten beispielsweise die Darstellung in einer Eingabemaske oder in einer Liste sein, Sichten auf eine Tabelle könnten die tabellarische Darstellung oder die Darstellung in einer Kuchengrafik sein.

Controller : Mittels Steuerungen (engl. Controller) werden die im Modell zusammengefassten Daten manipuliert. Controller in der Adressenkartei könnte beispielsweise der Rollbalken sein, mit dem in der Kartei geblättert wird, in der Tabellenkalkulation könnte ein Controller die Darstellung der Zahlen in einer Kuchengrafik auslösen.

Das MVC–Konzept wurde in der Literatur ausführlich beschrieben; es soll daher an dieser Stelle nicht näher auf dessen Hintergründe eingegangen werden.

Wie lässt sich nun die Konstellation von Analysekomponente, Transformationskomponente, Ferndarstellungskomponente sowie Fernsteuerungskomponente anhand des MVC–Konzepts einordnen? Die Analysekomponente befasst sich einzig mit der Präsentationsschicht der Anwendung, also mit deren *View*. Sie versucht, Informationen über den Aufbau der grafischen Benutzungsschnittstelle zu erfassen und diese für die weitere Verarbeitung verfügbar zu machen. Sowohl die Transformations– als auch die Ferndarstellungskomponente sind in der Folge damit betraut, die aus der Analyse der Sicht auf die Anwendung hervorgegangenen Informationen weiter zu verarbeiten und schließlich entfernt darzustellen. Auch sie können also der View zugeordnet werden. Die Fernsteuerungskomponente ist hingegen damit betraut, Befehle, die vom Anwender erteilt wurden, an die Anwendung weiterzugeben. Sie löst in der Anwendung Funktionen aus, die Einfluss auf die von der Anwendung gespeicherten Daten haben. Ihre Aufgabe ist entsprechend genau dem *Controller* Konzept zuzuordnen. Das Modell entspricht schließlich der analysierten Applikationslogik. Es steht extern zur Abstraktionsschicht und wird durch diese nicht selbst realisiert, sondern im Einklang mit dem MVC–Konzept nur indirekt über Controller und Views betrachtet und gesteuert.

In der tatsächlichen Implementation können mehrere der funktionalen Bestandteile zu zusammenhängenden Softwaremodulen integriert werden. Insbesondere für die Fernsteuerungskomponente und die Analysekomponente ist dieses sinnvoll, da beide direkt auf vergleichbar niedrigem Niveau mit der Anwendung interagieren. Im Zuge der weiteren Beschreibung in diesem Kapitel soll jedoch zunächst auf eine derartige Gruppierung verzichtet werden. Auf die tatsächliche Umsetzung wird in den folgenden Kapiteln noch näher eingegangen.

Neben eben skizzierten Kernkomponenten umfasst die Integrationsumgebung eine Reihe weiterer Module, welche die Nutzung der Anwendung von einem Mobilgerät aus ermöglichen. Diese regeln die Benutzerverwaltung und Zugriffssteuerung, die Geräteklassifikation und Anwendungskoordination usw. Auf die

Konzepte hinter all diesen Komponenten wird nunmehr der Reihe nach eingegangen werden, wobei mit der Analysekomponente begonnen werden soll.

Im vorigen Abschnitt wurden drei Klassen eingeführt, anhand derer Integrationsansätze auf Anwendungsebene unterteilt werden können. Vor dem Hintergrund dieser Klassifikation wurden sodann Projekte aus verschiedenen Bereichen der Integration vorgestellt und grob in ihrer Herangehensweise und ihren Eigenheiten umrissen. Zum Abschluss wurden aus den vorgestellten Projekten wünschenswerte Eigenschaften abgeleitet, die eine zu realisierende Integrationsumgebung aufweisen sollte. An diesem Punkt setzt nun dieser Abschnitt an. Es wird beschrieben, wie das Prinzip der Abstraktion, das allen zuvor beschriebenen Ansätzen zugrunde lag, in der Integrationsumgebung aufgegriffen und verwirklicht wird, die das Ergebnis dieser Arbeit darstellt.

4.3.1 Integration durch Abstraktion

Nahezu jedes Anwendungsprogramm lässt sich in zwei oder drei funktionale Komponenten zerlegen. Neben der Datenhaltung existiert üblicherweise eine Verarbeitung dieser Daten und schließlich die Interaktion mit dem Benutzer der Anwendung. Es wurde anhand diverser Beispiele bereits gezeigt, dass eine sehr mächtige und flexible Integrationsstrategie darin liegt, die oft monolithische Verflechtung zwischen Applikationslogik und Präsentationslogik aufzubrechen und zu einer klaren Schnittstelle zwischen beiden Funktionen zu kommen. Der Austausch zwischen beiden Komponenten findet dann über wohldefinierte Protokolle statt, indem z.B. Nachrichten über geänderte Daten an die Präsentationsschicht bzw. Nachrichten über Benutzereingaben an die Applikationsschicht gesendet werden. Durch die Entkopplung ist eine Relokation der Präsentationsschicht auf einen entfernten Rechnerknoten relativ leicht zu bewerkstelligen. Nachrichten werden nicht mehr durch lokale Prozeduraufrufe übermittelt, sondern durch entfernte Prozeduraufrufe oder anderweitige Netzwerkprotokolle (RPC, RMI, HTTP, WAP usw.) ausgetauscht.

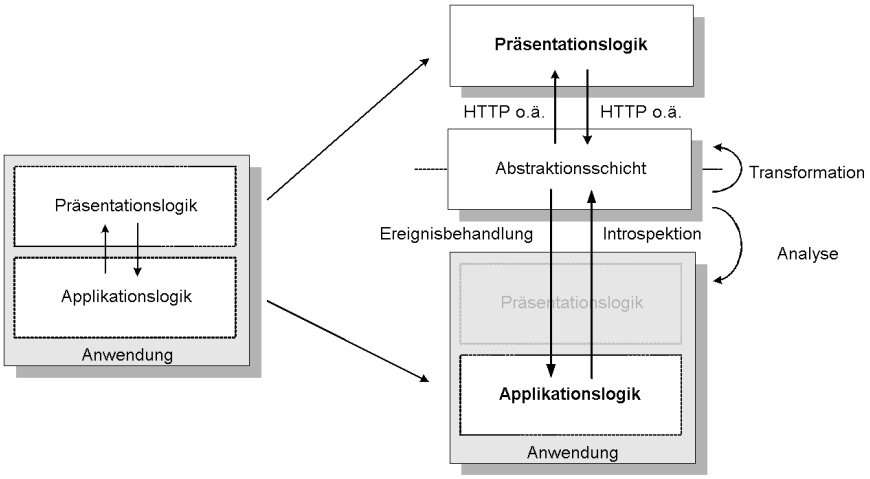


Abbildung 4.15: Schematische Darstellung des Abstraktionskonzepts

In Abbildung 4.15 wird das Prinzip verdeutlicht: Die im Gesamtkomplex der Anwendung verbundenen Funktionen der Applikationslogik sowie der Präsentationslogik werden durch die Einführung einer Abstraktionsschicht (logisch) auseinander gebrochen und die Präsentationsschicht der eigentlichen Anwendung im Kontext der Integrationsumgebung dabei kurzgeschlossen. Die Interaktion zwischen Applikationslogik und Abstraktionsschicht erfolgt über eine der Anwendung beigeordnete Analysekomponente. Innerhalb der etablierten Abstraktionsschicht erfolgt eine Transformation in das gewünschte Zielformat, und die Interaktion zwischen Abstraktionsschicht und Präsentationsschicht erfolgt schließlich mittels den von der Ferndarstellungskomponente realisierten Diensten. Die Darstellung der übertragenen Schnittstelle selbst erfolgt durch ein entsprechendes Anzeigeprogramm auf dem Endgerät.

4.3.1.1 Introspektion als Schlüsselmechanismus

Weiter oben wurden diverse Ansätze vorgestellt, die dem Entkopplungsprinzip folgen. Dabei wurde der Versuch unternommen, deutlich zu machen, dass eine wichtige Frage darin liegt, wie die Informationen aus der Applikationslogik extrahiert und an die Präsentationsschicht übermittelt werden können. Im Wesentlichen konnten dabei folgende Vorgehensweisen identifiziert werden:

- Benutzung von Systembibliotheken des Betriebssystems zur Erzeugung eines Zwischenformats
- Überwachung der Bildschirmausgabe und deren Übertragung auf einen entfernten Knoten
- Explizite Erstellung der abstrakten Schnittstellenbeschreibung und deren Speicherung in externen Dateien
- Benutzung spezieller Programmierbibliotheken im Zuge der Anwendungsentwicklung

Während die ersten beiden Verfahren sich dadurch auszeichnen, dass sie ohne explizite Berücksichtigung durch den Anwendungsprogrammierer zum Einsatz kommen können, haben die beiden letzten Verfahren den Vorteil, sehr gut mit der Anwendung abgestimmt werden zu können. Außerdem können in diesem Zusammenhang deklarative Sprachen auf Basis von XML zum Einsatz kommen, die sich über den hierfür extra vorgesehenen Weg der Stylesheets in eine Vielzahl existierender Beschreibungssprachen wie WML, HTML usw. übertragen lassen. Auf diese Weise kann dann eine große Menge von Endgeräten unterstützt werden, die nicht explizit auf den Einsatz in einem Integrationskontext vorbereitet sind.

Wie zum Ende des letzten Abschnitts formuliert wurde, liegt ein wichtiges Ziel dieser Arbeit darin, die Vorteile beider Verfahren miteinander zu verknüpfen. Es soll also untersucht werden, ob deklarative Sprachen zur Schnittstellenbeschreibung herangezogen werden können und gleichzeitig eine für das Anwendungssystem transparente Methode gefunden werden kann, die Informationen aus der Applikationslogik zu extrahieren, um sie in eine Beschreibung in einer deklarativen Sprache umzusetzen. Die Introspektion stellt das wesentliche Werkzeug hierzu dar.

Dem Wortsinne nach handelt es sich bei der Introspektion um eine „Selbstbetrachtung“ (vgl. [Bro86b]). Im Zusammenhang mit den Programmiersprachen Java bzw. C++ wurde der Introspektionsbegriff geprägt als die Möglichkeit, anhand von wenigen festgelegten Operationen die Aufrufschnittstelle von Objekten zu erkunden und so zur Laufzeit einer Anwendung dynamisch deren Funktionalität erschließen zu können (vgl. z.B. [Chi95] bzw. [Gol98]). In der vorliegenden Arbeit wurde das Konzept des *Application Shadow* entworfen, der beide Begrifflichkeiten in sich vereint, um das oben genannte Ziel zu erreichen: Der Application Shadow wird als Ausführungsumgebung der eigentlichen Applikation gestartet und in der Folge genutzt, um den jeweiligen Zustand der Benutzungsschnittstelle der Anwendung auszulesen und weiterzugeben. Es findet also eine Introspektion im allgemeinen Wortsinne statt, indem der Shadow „sich selbst“ (bzw. die in seiner eigenen Laufzeitumgebung gestartete Applikation) betrachtet. Gleichzeitig ist das Vorgehen jedoch auch mit der von Java bekannten Introspektion zu vergleichen, da über die durch *Reflection* oder ähnliche Mechanismen (siehe Kapitel 6) bereitgestellten Aufrufschnittstellen die Struktur der grafischen Benutzungsschnittstelle der Anwendung untersucht wird.

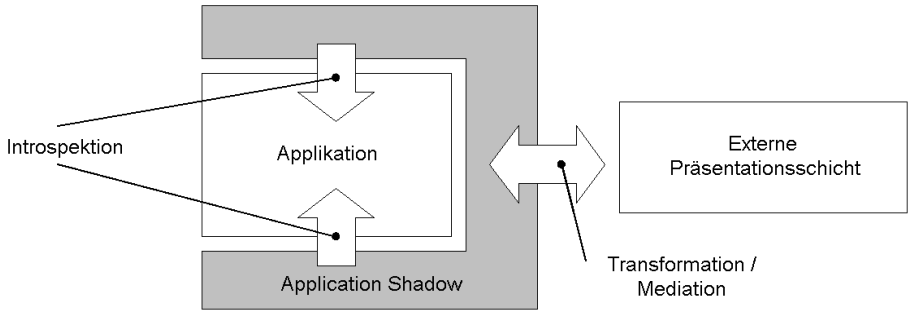


Abbildung 4.16: Schematische Darstellung der Introspektion

Abbildung 4.16 zeigt schematisch, wie die Entkopplung von Anwendungslogik und Präsentationslogik durch den Mechanismus der Introspektion zustandekommt. Der Application Shadow als Analysekomponente legt sich wie eine Klammer um die Anwendung. Über allgemein verfügbare Systemschnittstellen greift er auf die Anwendung zu und liest die Struktur ihrer Benutzungsschnittstelle aus. Die so gewonnenen Informationen werden in ein geeignetes Format transformiert und können dann an eine entfernte Komponente übergeben werden, welche die Beschreibung zu einer externen Präsentationsschicht aufbereitet[9].

Wie gezeigt wird die Introspektionskomponente direkt an die Anwendung gekoppelt. Ihre Aufgabe besteht darin, die Benutzungsschnittstelle zu analysieren mit dem Ziel, eine externe Repräsentation davon zu schaffen, die transformiert, über ein Netzwerk zum mobilen Gerät übertragen und dort dargestellt werden kann.

[9]Dem Application Shadow als Teil der entwickelten Integrationsumgebung kommt hierzu noch eine zweite Aufgabe zu: die Steuerung der Anwendung durch auf dem Endgerät entgegengenommene Befehle. Wie bereits angemerkt, soll diese Fernsteuerung als ein eigenständiger Teil des Gesamtkonzeptes an gesonderter Stelle in einem eigenen Unterabschnitt beschrieben werden.

Im Einzelnen lassen sich der Introspektionskomponente folgende Aufgaben zuordnen:

- Analyse der Benutzungsschnittstelle der Anwendung,
- Generierung einer internen Repräsentation der analysierten Schnittstelle in einem Objektgeflecht, aus dem die Beziehung der Schnittstellenelemente untereinander hervorgeht sowie
- Umformung der Repräsentation in eine abstrakte, externalisierte Form, die als Grundlage der weiteren Verarbeitung durch den Transformator dienen kann.

Bevor auf die drei Punkte im Detail eingegangen wird, soll zunächst kurz erörtert werden, warum bei der Erstellung der konkreten Zielrepräsentation zwischen Analyse und Darstellung der Zwischenschritt einer generischen Schnittstellenrepräsentation steht.

Ein intuitiver Ansatz wäre es, die in der Schnittstelle gefundenen Strukturen direkt in die gewünschte Zielrepräsentation, also z.B. die *Wireless Markup Language* umzusetzen. Für verschiedene Zielrepräsentationen ließen sich dabei jeweils spezifische Analysekomponenten bereitstellen, die den zeitaufwendigen Zwischenschritt überflüssig machen würden. Während dieses den Vorteil einer recht einfachen und somit effizienten Umsetzung bieten würde, stellt sich als Nachteil dar, dass bei diesem Vorgehen für jede Zielrepräsentation eine spezielle Introspektionskomponente realisiert werden müsste. Da die Introspektion jedoch in der Laufzeitumgebung der Komponente stattfindet, ließe sich in diesem Fall das Ziel nicht erreichen, dynamisch und zur Laufzeit der Anwendung eine Änderung der Zielrepräsentation durchführen zu können. Vielmehr müsste die Anwendung gestoppt und gemeinsam mit einer neuen Introspektionskomponente erneut gestartet werden. Um diese Problematik zu umgehen, ist es also notwendig, eine allgemeine Repräsentation zu nutzen, die entsprechend der jeweils gewünschten Zielrepräsentation in eine konkrete Form transformiert werden kann. Im Übrigen würde sich bei Hinzunahme von zusätzlichen Ausgangs– bzw. Zielrepräsentationen die Notwendigkeit ergeben, eine zunehmende Menge von Analysekomponenten für alle entstehenden Permutationen bereitzustellen, was als nicht sinnvoll erachtet werden muss.

Zunächst soll nunmehr auf die eigentliche Analyse der grafischen Benutzungsschnittstelle der Anwendung im Zuge der Introspektion eingegangen werden. Hierzu ist es sinnvoll zu verdeutlichen, in welcher Weise grafische Schnittstellen üblicherweise aufgebaut sind. Unabhängig von Parametern wie Betriebssystem, Programmiersprache oder Rechensystem werden Schnittstellen in modernen grafischen Systemen anhand von Objekthierarchien aufgebaut. Jedes GUI Element wie z.B. ein Listenfeld, eine Texteingabe oder ein Auswahlknopf wird durch eine entsprechende Objektklasse ausgedrückt, die den Zustand des Elements kodiert und dessen Anzeige auf dem Bildschirm veranlasst[10]. Jedem auf dem Bildschirm sichtbaren Element liegt eine konkrete Instanz der Objektklasse im Speicher des Rechners zugrunde.

[10] Auch hier kommt das bereits beschriebene MVC Konzept zum Tragen. Das Objekt stellt gewissermaßen das zugrunde liegende Modell des GUI Elements dar.

Gewisse Objektklassen können als Container aufgefasst werden. Sie sind in der Lage, Verweise auf andere Objekte zu enthalten. Für die auf dem Bildschirm sichtbare Schnittstelle bedeutet dieses, dass das Container–Element dem enthaltenen Element als Basis dient. Container–Elemente können sowohl auf dem Bildschirm sichtbar sein (wie z.B. die Auswahlliste, die eine Reihe von auswählbaren Einträgen enthält), sie können darüber hinaus jedoch auch für den Anwender unsichtbar sein. Diese Elemente, die oft auch als Layout–Container bezeichnet werden, dienen vornehmlich der Strukturierung der grafischen Schnittstelle. Sie gruppieren einzelne GUI Elemente und steuern deren Anordnung in Relation zueinander bzw. in Relation zu anderen Layout–Containern und GUI Elementen. Oftmals werden Layout–Container zur Kodierung semantischer Informationen der grafischen Benutzungsschnittstelle verwendet, indem sie die Zusammengehörigkeit von Elementen festlegen oder deren Unabhängigkeit voneinander unterstreichen. Für bestimmte sichtbare Container–Elemente gilt schließlich, dass sie auch die Rolle eines Layout–Containers übernehmen können, beispielsweise indem sie einen Rahmen um die enthaltenen GUI–Elemente zeichnen.

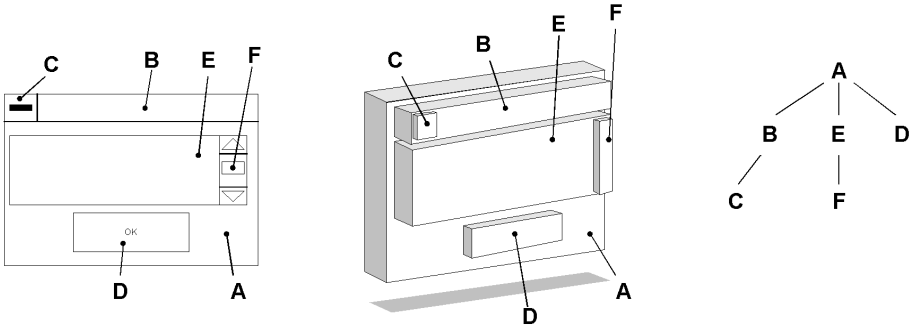


Abbildung 4.17: Verschiedene Darstellungsweisen einer Benutzungsschnittstelle

Üblicherweise werden grafische Schnittstellen aufgebaut, indem die benötigten Objektklassen instantiiert werden und dabei jeweils Referenzen auf ein enthaltenes Element bei der Initialisierung eines Container–Elements übergeben werden. Auf diese Weise entsteht dann im Speicher eine Baumstruktur aus Objekten. In Abbildung 4.17 ist das Prinzip verdeutlicht. Dargestellt ist eine sehr einfache grafische Schnittstelle, eine Dialogbox. Diese setzt sich aus folgenden GUI–Komponenten zusammen.

- Ein Fenster (A), das als Basis für alle enthaltenen GUI Elemente dient. Das Fenster stellt den zugrunde liegenden Container der Dialogbox dar.
- Die Titelzeile (B) der Dialogbox. Diese enthält im vorliegenden Fall nichts weiter als einen Knopf (C), hinter dem sich ein System–Menü verbirgt.
- Ein Auswahlknopf (D), mit dem die Dialogbox wieder geschlossen werden kann.
- Eine Liste (E). Diese ist im vorliegenden Fall nicht mit Einträgen gefüllt.

- Ein Rollbalken (F) zum Blättern in der Liste[11].

Wie sich aus der Abbildung (Mitte) ergibt, kann das auf dem Bildschirm sichtbare Fenster mit den darin enthaltenen GUI Elementen als eine dreidimensionale Schichtung von Objekten aufgefasst werden. Jedes Element kann dabei als Plattform für weitere Elemente dienen. Im Beispiel stellt das Dialogfenster die Basis für alle anderen Elemente dar. Hierauf sind die Titelzeile, die Liste und der Auswahlknopf angeordnet, die in der beschriebenen Weise wiederum den System–Knopf und den Rollbalken enthalten. Drückt man die Beziehung zwischen den Objekten schließlich durch einen Baum aus, so erhält man die rechte Darstellung aus Abbildung 4.17.

Anhand dieser schematischen Darstellung lässt sich nun das Vorgehen der Introspektionskomponente bei der Analyse der grafischen Benutzungsschnittstelle veranschaulichen. Ist es bei gewissen, im ersten Teil dieses Kapitels beschriebenen Ansätzen zur Entkopplung zwischen Präsentationsschicht und Applikationslogik wie z.B. VNC so, dass (bildlich gesprochen) auf der zweidimensionalen Sicht der Schnittstelle aufgesetzt und lediglich das Abbild selbst an einen Zielknoten übertragen wird, dringt das hier entworfene Konzept zur Analyse in die Objekthierarchie ein. Mit den vom Betriebssystem bzw. dem GUI System zur Verfügung gestellten Introspektionsmechanismen wird zunächst das Basisobjekt ermittelt. Hiervon ausgehend wird dann jeweils festgestellt, ob das gerade behandelte Objekt ein Containerobjekt ist und ob es ggf. Verweise auf weitere Objekte enthält. Es werden die Attribute des Objektes ermittelt (z.B. Identität, Typ, Größe, Belegung, ggf. auch Farbe, Form usw.) und diese dann in das interne Zwischenformat umgeformt. Handelt es sich um einen unsichtbaren Layout–Container, so wird dieses in der generierten Zwischenrepräsentation vermerkt, bevor zu den enthaltenen GUI–Elementen übergegangen wird. Sukzessive kann so die Schnittstelle analysiert und schrittweise in eine interne Repräsentation überführt werden.

Bestimmte Aspekte sind bei diesem Übergang von der konkreten Form der grafischen Schnittstelle in die intern verwendete Repräsentation unbedingt zu beachten:

1. Es ist essentiell für die weitere Verarbeitung, dass jederzeit eindeutig von der internen Repräsentation eines GUI Elements auf das ursprüngliche Objekt in der grafischen Benutzungsschnittstelle rückgeschlossen werden kann. Hierzu muss im Zuge der Analyse eine eindeutige Identität festgestellt werden, soweit das GUI System dieses zulässt, oder andernfalls beigeordnet werden und diese während der weiteren Verarbeitung verwaltet werden. Nur so kann die spätere Zuordnung von Ereignissen zu konkreten Elementen der Schnittstelle erfolgen.

2. Üblicherweise ist in einer grafischen Schnittstelle nicht nur die bloße Existenz von GUI Elementen relevant, sondern auch deren Anordnung relativ zueinander bzw. zu ihrem Container. Über die Anordnung werden oftmals semantische Informationen übermittelt, so dass die Anordnung der

[11] Tatsächlich würde der Rollbalken selbst wieder in eine Reihe weiterer Objekte zerfallen. Der Einfachheit halber soll in dieser Darstellung jedoch auf die weitere Aufspaltung verzichtet werden.

Elemente möglichst zu erhalten ist. Layout–Container müssen also erkannt und in einer geeigneten Form in die Zwischenrepräsentation integriert werden.

3. Die Analyse erfolgt nicht einmalig, sondern muss fortlaufend wiederholt werden. Entsprechend ist ein Verfahren notwendig, das die Analyse effizient durchführt. Anzustreben ist hierbei ein Vorgehen, bei dem die Analyse durch von Controllern ausgelöste Änderungen in der Schnittstelle selbst angestoßen wird (sog. „Push–Prinzip“) .

Es muss betont werden, dass die Objekthierarchie der grafischen Benutzungsschnittstelle durch die Introspektion nur analysiert, *nicht* jedoch beeinflusst wird. Es werden weder Objekte eingefügt noch Objekte aus der Hierarchie entfernt. Dieses gilt im Übrigen auch für die Fernsteuerungskomponente: Auch sie greift nicht direkt auf die Objekte zu, sondern bedient sich externer Schnittstellen – hierzu jedoch später mehr.

Wesentlich für das entwickelte Konzept ist an dieser Stelle nur, dass vom Betriebssystem bzw. dem GUI System ein Weg zur Verfügung gestellt werden muss, auf die Objekte zuzugreifen, die hinter der grafischen Schnittstelle stehen. Es muss möglich sein, das Basisobjekt festzustellen und von diesem ausgehend durch die Hierarchie der GUI Elemente zu navigieren, um die Zwischenrepräsentation zu erstellen. Wie dieses im Einzelnen geschieht ist Sache der Analysekomponente selbst. Diese muss lediglich in der Lage sein, als Ergebnis der Analyse eine Repräsentation der Schnittstelle in einem zuvor festgelegten Zwischenformat zu erstellen. Verschiedene GUI Systeme (z.B. MOTIF [You90, You95], Qt [Dal99], Win32/MFC [Feu97] usw.) können durch jeweils auf das zugrunde liegende Objektsystem abgestimmte Analysekomponenten unterstützt werden. Bezogen auf die Integrationsumgebung selbst wirkt die Analysekomponente also als eine Abstraktionsschicht gegenüber dem konkreten GUI System. Betrachtet man nochmals Abbildung 4.17, so stellt die Analysekomponente den Schritt zwischen der mittleren und der rechten Skizze dar: Es findet eine Umformung der konkreten Schnittstelle in einer für das jeweilige GUI System spezifischen Repräsentation (Mitte) in ein generisches Zwischenformat (rechts) statt, wobei die Analyse einer für verschiedene GUI Systeme implementierten Dialogbox im Idealfall zu einer identischen Zwischenrepräsentation führen sollte.

Die Zwischenrepräsentation stellt gewissermaßen den Ausgangspunkt für die eigentliche Verarbeitung der zuvor gesammelten Informationen über die grafische Benutzungsschnittstelle innerhalb der Integrationsinfrastruktur dar. Entsprechend sind an sie folgende Forderungen zu knüpfen:

Abbildung hierarchischer Strukturen : Wie bereits erläutert, lassen sich grafische Benutzungsschnittstellen, wie sie in den gängigsten GUI Systemen definiert werden, als eine hierarchische Anordnung von Objekten verstehen, die im Speicher des Rechners durch Referenzen miteinander verkettet sind. Als wichtigstes Merkmal der zu entwerfenden Zwischenrepräsentation ist also die Möglichkeit zu werten, eine solche hierarchische Struktur auszudrücken.

Effizienz : Die Zwischenrepräsentation muss sich sehr effizient konstruieren und sodann weiter verarbeiten lassen. Beim Übergang von Zwischenre-

präsentation zu Zielrepräsentation sollte möglichst keine zusätzliche Umformung in ein vom Übersetzer verwendbares Eingabeformat notwendig werden. Die Zwischenrepräsentation muss also direkt vom Übersetzer interpretierbar sein. Die grafische Schnittstelle muss während der Laufzeit der Anwendung viele Male abgetastet, in das abstrakte und danach in ein konkretes Format umgesetzt werden. Eine hohe Effizienz ist daher unbedingt erforderlich.

Plattformunabhängigkeit : Die Zwischenrepräsentation sollte keine technische Abhängigkeit von bestimmten Plattformen oder Programmiersprachen besitzen. Insbesondere sollten der Aufbau und die weitere Verarbeitung von Schnittstellenrepräsentationen in verschiedenen Programmiersprachen oder auf unterschiedlichen Hardwareplattformen erfolgen können. Diese Forderung entspringt einerseits dem Abstraktionsgedanken, nach dem die Unterstützung beliebiger GUI Systeme durch die Integrationsumgebung möglich sein sollte. Andererseits ist nur durch Plattformunabhängigkeit die Verteilung von Verarbeitungsfunktionen auf verschiedene Rechnerknoten bzw. die Umsetzung einzelner Integrationsschritte in der jeweils optimalen Programmiersprache möglich, was nicht zuletzt zugunsten der geforderten Effizienz wünschenswert ist.

Verteilung : Da aufgrund der zu fordernden Effizienz auf die Skalierbarkeit der zu entwerfenden Konzepte geachtet werden muss, ist für die Zwischenrepräsentation sicherzustellen, dass sie die Verteilung auf verschiedene Rechnerknoten erlaubt. Insbesondere ist also die schnelle Externalisierung (bzw. Serialisierung) sowie Rekonstruktion zuvor externalisierter (bzw. serialisierter) Daten zu fordern.

Erweiterbarkeit : Die Zwischenrepräsentation sollte hinsichtlich ihrer Ausdrucksmächtigkeit so gewählt werden, dass die nachträgliche Erweiterung zur Unterstützung zusätzlicher GUI Systeme jederzeit möglich ist. Insbesondere sollte der Entwurf sicherstellen, dass eine solche Erweiterung möglich ist, ohne dass hierbei der grundsätzliche Aufbau der Zwischenrepräsentation betroffen wird.

Alle genannten Forderungen sprechen für den Einsatz von XML als Grundlage der abstrakten Zwischendarstellung und für die Verwendung von DOM (*Document Object Model*) als Basis für deren interne Verarbeitung. XML wurde explizit für die Verarbeitung hierarchischer Dokumente entwickelt. In XML ausgedrückte Informationen sind plattformunabhängig. Sie besitzen mit DOM eine effizient nutzbare *interne* Repräsentation und lassen sich aus ihrer DOM Repräsentation ebenso effizient in XML Dokumente externalisieren, die als einfache Texte zwischen Rechnern ausgetauscht werden können. Auch existiert mit XSL ein umfassendes Werkzeug für XML zur Transformation von Dokumenten anhand formal definierbarer Regeln (vgl. [ABK+00]). Über die *Document Type Definition* (DTD) lässt sich die Struktur einer mittels XML definierten Zwischenrepräsentation schließlich genau spezifizieren und durch Anpassung der DTD zu späterem Zeitpunkt auch kontrolliert erweitern bzw. modifizieren.

Entsprechend wurde für die interne Darstellung der Benutzungsschnittstelle eine DTD und eine Sprache (die unter Abschnitt 4.3.3 näher beschriebene *Java Swing*

Markup Language) definiert, in welcher das Aussehen der Schnittstelle in einem plattformneutralen Format ausgedrückt werden kann. Indem die Analysekomponente durch die hierarchisch angeordneten Elemente der Benutzungsschnittstelle navigiert, entsteht eine interne Repräsentation, die den Definitionen der verwendeten DTD genügt. Gewissermaßen wird so in einem ersten Transformationsschritt mittels Introspektion die für das GUI System spezifische Objektrepräsentation der Schnittstelle in eine für die Integrationsumgebung spezifische Objektrepräsentation im DOM Format umgewandelt. Am Ende dieses ersten Schritts steht also ein Objektgeflecht im DOM Format.

Welche Forderungen lassen sich nun auf konzeptioneller Ebene an den Entwurf der als Zwischenrepräsentation genutzten Sprache knüpfen? Bereits ohne auf die Implementation im Einzelnen einzugehen, lassen sich wesentliche Merkmale formulieren, welche die Sprache aufweisen muss:

Plattformneutralität : Die zu entwerfende Integrationsumgebung soll eine Unterstützung möglichst beliebiger Ausgangsplattformen bieten. Entsprechend darf die abstrakte Zwischenrepräsentation im Idealfall keine Festlegung auf eine konkrete Programmiersprache oder ein bestimmtes GUI System enthalten. Dieses gilt ebenso auch für die von der Integrationsumgebung zu unterstützenden Zielplattformen, gegenüber denen die zu entwerfende Repräsentation neutral sein sollte.

Ausdrucksstärke : Die Zwischenrepräsentation soll alle Aspekte der grafischen Benutzungsschnittstelle abbilden können. Insbesondere angesichts der geforderten Unterstützung für beliebige Ausgangsplattformen muss auf eine möglichst große Ausdrucksstärke der zu konzipierenden Sprache geachtet werden.

Flexibilität : Da sich nicht ausschließen lässt, dass im Falle des Hinzufügens weiterer Ausgangsplattformen die Zwischenrepräsentation nicht alle Aspekte der neuen Ausgangssprache darzustellen vermag, muss die zu entwerfende Sprache über genügende Flexibilität verfügen, um diese auch nachträglich noch erweitern zu können. Insbesondere muss darauf geachtet werden, dass Erweiterungen möglich sind, ohne hierbei grundsätzliche Änderungen am bestehenden Integrationskonzept notwendig werden zu lassen.

Von besonderer Wichtigkeit ist der erste Punkt. Nur wenn es gelingt, die für eine bestimmte Ausgangs– oder Zielplattform spezifischen Aspekte aus der Sprachdefinition herauszuhalten, ist die nachträgliche Erweiterung hinsichtlich einer Unterstützung weiterer Plattformen realisierbar. Es ist genau dieser Aspekt, welcher der Zwischenrepräsentation das Attribut des „abstrakten“ bzw. „generischen“ hinzufügt.

Weiterhin ist zu beachten, dass die Zwischenrepräsentation entsprechend der Gesamtkonzeption ausschließlich zur Kodierung des Aussehens der grafischen Applikationsschnittstelle dienen soll, nicht jedoch zur Beschreibung funktionaler Aspekte. So soll im Falle einer grafischen Schnittstelle beschrieben werden können, dass diese ein Textfeld sowie einen Auswahlknopf enthält, nicht jedoch, dass der Auswahlknopf zum Löschen der im Textfeld befindlichen Zeichen dient.

Ebenso werden keine semantischen Informationen in der Zwischenrepräsentation gespeichert, da es mit Hilfe mechanischer Verfahren kaum möglich ist, diese aus der Schnittstelle zu extrahieren. So kann zwar erkannt werden, dass ein Dialogfenster über zwei Auswahlknöpfe verfügt, es ist jedoch nicht zu erkennen, welcher Knopf der Bestätigung des Dialoges dient bzw. welcher dessen Abbruch. Entsprechende Informationen werden daher auch nicht in der Zwischenrepräsentation erfasst.

Insgesamt dient die Zwischenrepräsentation also der Speicherung von drei Aspekten der zuvor analysierten grafischen Benutzungsschnittstelle:

- GUI–Elemente
- Container– bzw. Layout–Elemente
- Meta–Informationen

Die zuerst genannte Gruppe enthält alle sichtbaren Elemente der Schnittstelle. Dieses können Knöpfe, Listen, Textfelder, Bearbeitungsfelder usw. sein. Wichtig ist hierbei, dass bei der Konzeption der Sprache zwar konkrete GUI Systeme herangezogen werden können, dass aber so weit als möglich auf die Beschränkung auf Elemente eines bestimmten GUI Systems verzichtet werden und stattdessen danach gestrebt werden muss, eine möglichst umfassende Übermenge zu berücksichtigen.

Die zweitgenannte Gruppe umfasst die bereits beschriebenen Struktur gebenden bzw. nicht sichtbaren Elemente. Dieses sind beispielsweise die sichtbaren Rahmen, die bestimmte GUI–Elemente einfassen oder GUI–Elemente, die ihrerseits als Basis für weitere GUI–Elemente dienen. Schließlich müssen alle Elemente erfasst werden, die nicht direkt sichtbar sind, sondern lediglich zur Anordnung enthaltener Elemente Verwendung finden. Soweit diese im Analyseprozess erfasst werden können, bildet ihre Speicherung letztlich die einzige Möglichkeit zur begrenzten Übermittlung gewisser semantischer Informationen. Ihnen fällt im Zuge der Introspektion also eine durchaus wichtige Rolle zu.

Während das Augenmerk bei der Konzeption der Zwischenrepräsentation im Wesentlichen auf der Speicherung von Merkmalen der zuvor analysierten Benutzungsschnittstelle liegt, dient die Sprache darüber hinaus auch dem Austausch von Informationen *über* die untersuchte Schnittstelle. Diese als „Meta–Information“ bezeichneten Informationen umfassen etwa die eindeutigen Kennungen, die erkannten GUI–Elementen beigeordnet wurden, die laufende Nummer des Analysevorgangs, anhand derer die verzögerte Transformation gesteuert werden kann usw. Während die Speicherung von Meta–Informationen im Umfang der Zwischenrepräsentation einen vergleichsweise kleinen Raum einnimmt, ist bei ihrer Konzeption gleichwohl darauf zu achten, dass die Behandlung von Meta–Informationen möglich ist.

4.3.2 Entfernte Steuerung von Anwendungen

Die Kopplung zwischen zu integrierender Anwendung und Integrationsumgebung erfolgt bidirektional. Durch die Analyse der Anwendungsschnittstelle entsteht die abstrakte Beschreibung des GUIs, die zum Kern der Integrationsumgebung übertragen, dort in weiteren Schritten transformiert und schließlich an

mögliche Endgeräte ausgesendet wird. In entgegengesetzter Richtung werden die als Resultat der Bedienung der Schnittstelle auf dem Endgerät erzeugten Befehle durch die Integrationsumgebung zur Applikation zurückgeschleust. Sie müssen hier in geeigneter Form an die eigentliche Benutzungsschnittstelle übergeben werden, um sie so zur Beeinflussung der Anwendung einsetzen zu können. Die hierfür notwendige Steuerungslogik wird konzeptionell der Fernsteuerungskomponente zugeschlagen. Ihrer Beschreibung dient dieser Abschnitt.

Ähnlich wie für die Zwischenrepräsentation lassen sich auch für die Fernsteuerung der Anwendung bestimmte Merkmale fordern, welche diesen aufgrund gemeinsamer Optimierungsziele ähneln. Im Einzelnen soll für das Konzept Folgendes gelten:

Plattformneutralität : Das Fernsteuerungskonzept soll nicht spezifisch für eine bestimmte Ausgangs– oder Zielplattform sein. Sind im konkreten Fall für die Unterstützung einer bestimmten Ausgangs– oder Zielplattform plattformspezifische Realisierungsschritte notwendig, so sollen sich diese auf die Fernsteuerungskomponente selbst isolieren lassen. Der Wechsel der Ausgangs– oder Zielplattform soll also nicht den Wechsel der jeweils anderen Plattform nach sich ziehen.

Flexibilität : Der Fernsteuerungsmechanismus ist so zu konzipieren, dass auch im Nachhinein noch weitere Ausgangsplattformen hinzugenommen werden können, ohne dass der Mechanismus hierfür grundsätzlich abgeändert werden müsste.

Realisierungstransparenz : Die Fernsteuerungskomponente soll ohne explizite Modifikation der Anwendung auskommen. Insbesondere sollen keine Änderungen des Anwendungsquelltextes erforderlich sein.

Nutzungstransparenz : Die Anwendung soll die entfernte Steuerung nicht als solche wahrnehmen. Die entfernte Nutzung soll sich insbesondere nicht von der direkten Nutzung über Tastatur und Maus unterscheiden, die an den Rechner angeschlossen sind, auf dem die Anwendung läuft.

Generizität : Der Fernsteuerungsmechanismus soll nicht spezifisch für bestimmte Anwendungen sein und auf deren Unterstützung hin optimiert werden müssen. Vielmehr soll der Mechanismus auf einer Ebene ansetzen, auf welcher für alle Anwendungen eines bestimmten GUI Systems dieselbe Fernsteuerungskomponente zum Einsatz kommen kann.

Daneben ist als besonders wichtig zu erachten, dass der Fernsteuerungsmechanismus die Anwendung nicht vollständig isolieren darf. Neben einer entfernten Nutzung soll auch weiterhin der unmittelbare Zugriff über Tastatur und Bildschirm des Rechners möglich sein, auf dem die Anwendung läuft. Hierzu muss folglich die Fernsteuerung während eines Anwendungslaufs jederzeit auszuschalten und zu späterem Zeitpunkt ebenso wieder einzuschalten sein. Schließlich soll zur Laufzeit der Anwendung ein Wechsel des Endgerätes möglich sein, von dem auf die Anwendung zugegriffen wird, ohne dass der Wechsel einen Neustart zwecks Anpassung der Fernsteuerungskomponente auf das neue Zielmedium notwendig machen würde.

Trotzdem muss die entfernte Steuerung der Anwendung genügende Effizienz aufweisen. Die Nutzung vom entfernten Endgerät aus soll sich nicht grundsätzlich von der lokalen Nutzung unterscheiden. Abgesehen von der unvermeidbaren Übertragungsverzögerung (und natürlich abgesehen von der geänderten Charakteristik der auf das Endgerät transponierten Schnittstelle) ist also ein ähnliches „Nutzungsgefühl“ für den Anwender der Applikation zu erzielen.

Wie lassen sich die Forderungen erfüllen? Zur Klärung dieser Frage soll zunächst näher auf mögliche Ziele und Formen der Anwendungssteuerung eingegangen werden.

Im Wesentlichen sind folgende Motivationen für die Fernsteuerung von Anwendungssystemen denkbar:

- Nutzung der Anwendung von einem entfernten Rechnerknoten aus („Fernsteuerung“ im eigentlichen Wortsinne).
- Autonome Wiederholung von Anwendungsfunktionen.

In die erste Gruppe fallen alle in vorigen Abschnitten genannten Ansätze wie VNC, RDP, ICA usw. In jedem dieser Fälle wird versucht, eine Anwendung von einem entfernten Rechner zu steuern und die Resultate der Steuerungseingriffe auf dem Bildschirm jenes Rechners anzuzeigen. Die Wiederholung von Anwendungsfunktionen wird hingegen vornehmlich zur Arbeitserleichterung (bspw. bei der Aufzeichnung von Makros in einer Textverarbeitung) sowie zur Funktionsanalyse und dem Stresstest von Programmen im Rahmen der Anwendungsentwicklung verwendet. Hierbei wird das Nutzungsverhalten des Anwenders simuliert, indem die Verwendung von Maus und Tastatur in bestimmten Sequenzen aufgezeichnet wird, die dann quasi zufällig aneinander gereiht und immer wieder abgespielt werden. Die tatsächlich ausgelösten Tastendrucke, Mausverschiebungen, Eingaben usw. werden protokolliert und können so zu späterem Zeitpunkt nachvollzogen werden.

Die Anwendungssteuerung durch den Anwender erfolgt bei normalen Rechnersystemen über die Eingabegeräte, also zumeist Tastatur sowie Maus. Mittels dieser Geräte werden Befehle ausgelöst oder Eingaben gemacht, welche dann die Anwendungsfunktion beeinflussen. Das Ziel einer Anwendungsfernsteuerung liegt entsprechend darin, die Eingaben zu simulieren, die von diesen Eingabegeräten erzeugt werden können. Diese müssen auf dem entfernten Rechnerknoten erfasst, über den Kommunikationskanal an den fernzusteuernden Rechner übergeben und dort an geeigneter Stelle in den Eingabestrom der Anwendung eingefügt werden.

Um zu klären, an welcher Stelle die Modifikation des Eingabestroms erfolgen kann, soll die Eingabeverarbeitung moderner GUI Systeme grob skizziert werden. Um den Anforderungen des zugrunde liegenden Mehrprozesssystems gerecht zu werden, erfolgt die Verarbeitung von Eingaben in solchen Umgebungen nicht direkt durch die Anwendung selbst. Vielmehr wird die Verarbeitung an zentraler Stelle innerhalb des GUI Systems durchgeführt und dann für eine Verteilung an jeweils aktive Anwendungen gesorgt. Von grundlegender Bedeutung für diese Zentralisierung war die Einführung von Ereignissen (Events), auf denen die Verarbeitung von Ein– und Ausgaben in GUI Systemen basiert. Jede Änderung in einem GUI System wird in Form eines Ereignis kodiert. Auf

niedrigem Abstraktionsniveau kann ein Ereignis das Niederdrücken einer bestimmten Taste oder die Bewegung der Maus darstellen. Auf höherem Niveau kann das Aktivieren eines Fensters ein Ereignis sein oder das Auswählen des Systemmenüs. Ereignisse werden in GUI Systemen zumeist als Datenstrukturen realisiert, die eine eindeutige Ereigniskennung enthalten (z.B. für „Taste gedrückt“) sowie Nutzdaten, die dem Ereignis zuzuordnen sind. Dieses können der Ursprung und der Empfänger des Ereignisses sein oder auch Informationen über das Ereignis selbst (z.B. „gedrückte Taste war ein ‘a‘ “). Ereignisse auf niedrigem Abstraktionsniveau entstehen, indem das GUI System bestimmte Unterbrechungsroutinen im Betriebssystem anmeldet und so über die Änderung bestimmter Systemzustände informiert wird. Nach dem Eintreten eines solchen Ereignis kann dann die entsprechende Datenstruktur erstellt werden. Ereignisse auf höherem Niveau können zudem auch von Anwendungen selbst über die vom GUI System hierfür angebotenen Schnittstellen erzeugt werden.

Der Austausch von Ereignissen zwischen GUI System und Anwendung sowie zwischen Anwendungen untereinander wird als Ereignisbehandlung bezeichnet. Die Ereignisbehandlung erfolgt in so genannten Warteschlangen, in welche die Ereignisse in der Reihenfolge ihres Eintreffens eingereiht werden. Das GUI System entnimmt der Warteschlange jeweils ein Ereignis, prüft, ob dieses einer bestimmten Anwendung zuzuordnen oder an die derzeit aktive Anwendung auszuliefern ist und überträgt die Ereignisdaten sodann an den ermittelten Empfänger. Der Empfänger kann dann die Nutzdaten des Ereignis auswerten und entsprechend darauf reagieren.

In GUI Systemen – und hierbei insbesondere in objektorientierten Varianten solcher Systeme – ist die Reaktion auf das Eintreffen von Ereignissen für GUI Elemente relativ stark reglementiert. Jedes Element besitzt vorgegebene Ereignisse, die im Entwurf des GUI Systems vorgesehen wurden und auf die das GUI Element reagieren kann. Die Ereignisverwaltung beschränkt sich in diesem Fall nicht auf die bloße Auslieferung eines Ereignisses an die gerade aktive Anwendung, sondern ruft direkt am betroffenen GUI Element die Behandlungsroutine für das Ereignis auf. So kann der Anwendungsentwickler im Falle eines Auswahlknopfes eine Routine schreiben, die gestartet wird, wenn der Knopf gedrückt wurde. Im Falle eines Rollbalkens kann er Routinen für das Verschieben des Reglers sowie für das Drücken auf einen der Pfeile an den Enden des Rollbalkens bereitstellen usw. Im objektorientierten System existieren für die verschiedenen GUI Elemente jeweils Standardroutinen, die nur abgeändert werden, wenn eine anwendungsspezifische Behandlung gewünscht ist. Bei der Instantiierung eines GUI Elements erfährt das GUI System von der Existenz der Behandlungsroutinen und kann diese dann nach Eintreffen des Ereignisses gegebenenfalls aufrufen.

Für die Fernsteuerung von Anwendungen ergeben sich somit mehrere Ebenen, auf denen der Eingriff in die Verwaltung von Eingaben vonstatten gehen kann:

- Beeinflussung von Systemunterbrechungen, z.B. durch Auslösung bzw. Vortäuschung von Tastatur–Interrupts.
- Generierung von Ereignissen auf Ebene des GUI Systems und deren Parametrisierung mit den gewünschten Nutzdaten.
- Generierung von Ereignissen auf Ebene der Anwendung, d.h. Aufruf der

Behandlungsroutinen mit Kodierung der gewünschten Nutzdaten in den Aufrufparametern der Anwendung.

Der erste Fall stellt einen stark plattformspezifischen Eingriff in das Rechensystem dar, der sich nur in wenigen Fällen rechtfertigen lässt. So ist ein solches Vorgehen sinnvoll, wenn im Rahmen von Systemtests das GUI System selbst oder gar das zugrunde liegende Betriebssystem gesteuert werden soll und die Generierung von Ereignissen eine unzulässige Beeinflussung des zu testenden Systems darstellen würde. Für die Mehrzahl der Fälle ist dieses Vorgehen jedoch aufgrund seiner Komplexität nicht anzustreben.

Die Generierung von Ereignissen über die vom GUI System angebotenen Schnittstellen stellt hingegen ein recht weit verbreitetes Vorgehen zur Fernsteuerung dar. Hierbei werden die über das Netzwerk vom entfernten Rechnerknoten eintreffenden Befehle von einem speziellen Programm empfangen, das diese in Ereignisse umsetzt, die dann in der für das GUI System üblichen Weise verarbeitet werden. Meist übernimmt dieses Programm zugleich die Aufgabe, Änderungen im Bildschirmaufbau zu erfassen und diese an den entfernten Rechnerknoten zurückzuübermitteln. Es ist dieser Ansatz, der beispielsweise von VNC verfolgt wird und der auch die konzeptionelle Grundlage für RDP oder ICA bildet. Er bietet den Vorteil, aus der Sicht der Anwendung vollständig transparent zu sein. Die Anwendung kann nicht erkennen, ob ein Ereignis durch das Eintreten eines Systemereignisses ausgelöst wurde oder ob es über die Aufrufschnittstelle des GUI Systems von einem anderen Anwendungsprogramm quasi synthetisiert wurde. Insbesondere bei der Fernsteuerung des gesamten GUI Systems (wie es bei VNC, RDP oder ICA der Fall ist) stellt die Beeinflussung der Ereigniswarteschlange den besten Ansatz dar.

Im Kontext dieser Arbeit erfolgt die Fernsteuerung nicht auf der Ebene des GUI Systems, sondern auf der nächsthöheren Ebene. Es wird die grafische Benutzungsschnittstelle einer *bestimmten* Anwendung analysiert, diese in ein abstraktes Zwischenformat übertragen und an entfernter Stelle wieder visualisiert. Demgemäß muss auch nicht das gesamte GUI System bei der Steuerung berücksichtigt werden, sondern lediglich die einzelne dargestellte Anwendung.

Während ein Eingreifen in die Ereignisbehandlung auf Systemebene auch in diesem Fall ein mögliches Vorgehen darstellt, setzt es genau genommen auf einer zu niedrigen Ebene an. Es ist als sinnvoller zu erachten, falls möglich, in die Ereignisbehandlung der Anwendung selbst einzugreifen. Wie oben geschildert wurde, müssen hierfür die Behandlungsroutinen der GUI Elemente betrachtet werden. Es gilt, diese in der gewünschten Weise aufzurufen und dabei die richtigen Parameterwerte zu übergeben. Der wesentliche Vorteil bei diesem Vorgehen besteht darin, dass das GUI System von der Modifikation weitgehend unbeeinflusst bleibt; eine spezielle Programmierschnittstelle des GUI Systems muss nicht vorausgesetzt werden. Außerdem vereinfacht sich in diesem Fall die Behandlung von Ereignissen, die vom entfernten Endgerät über die Nutzung bestimmter GUI Elemente eintreffen. So müssen diese nicht erst in eine Menge von Ereignissen umgesetzt werden, welche den Mauszeiger neu positionieren und dann das Drücken des Mausknopfes vortäuschen, wie dieses notwendig wäre, wenn auf der Ereignisverarbeitung des GUI Systems selbst aufgesetzt werden würde. Vielmehr kann das betroffene GUI Element direkt angesprochen werden.

Während dieses beispielsweise für VNC kein Problem darstellt, da auf dem entfernten System auch die Bewegung des Mauszeigers als Ereignis erfasst wird und ohnehin bereits entsprechende Ereignisse übertragen werden, geht das hier entwickelte Konzept nicht vom Vorhandensein eines Maus–ähnlichen Zeigegerätes auf dem Endgerät aus (beispielsweise bei der Verwendung einer natürlichsprachlichen Geräteschnittstelle). Entsprechende Ereignisse müssten also vollständig durch die Fernsteuerungskomponente synthetisiert werden, was zu einer stark erhöhten Komplexität führen würde.

Andererseits muss natürlich sichergestellt werden, dass die Fernsteuerungskomponente in der Lage ist, die Behandlungsroutinen der einzelnen GUI Elemente zu erkennen und diese in der Folge direkt zu nutzen. Die Fernsteuerungskomponente kann hierbei auf die Informationen zurückgreifen, die bereits im Zuge der Schnittstellenanalyse erfasst werden. So kennt die Integrationsumgebung bereits die Identitäten aller GUI Elemente der Anwendung und kann auf diese somit direkt zugreifen. Über denselben Mechanismus der Introspektion, welcher es zuvor erlaubte, Informationen über die GUI Elemente zu erfragen, können nunmehr die Behandlungsroutinen gesteuert werden. Wird auf dem entfernten Gerät ein bestimmtes GUI Element ausgewählt, so wird dieses Ereignis von der Integrationsumgebung aufgefangen und als ein Befehl an die Fernsteuerungskomponente weitergeleitet. Diese wählt anhand der zuvor von der Analysekomponente gesammelten Informationen das betroffene GUI Element der eigentlichen Anwendung aus und ruft schließlich dessen Behandlungsroutine auf.

Die Integrationsumgebung ist nicht ausschließlich auf die unmittelbare Nutzung von Behandlungsroutinen festgelegt. So weit dieses notwendig ist, kann auch auf der Ebene der niedrigeren Ereigniswarteschlange des GUI Systems angesetzt werden. Dieses ist der Fall bei Ereignissen, die nicht einzelne Elemente der Schnittstelle, sondern die Anwendung als Ganzes betreffen. So existieren in grafischen Systemen oftmals Befehlsmenüs, die der Anwendung beigeordnet, von dieser jedoch nicht selbst verwaltet werden (sog. Systemmenüs). In Systemmenüs sind z.B. Befehle zur Beendigung der Anwendung oder zur Änderung ihrer Fenstergröße untergebracht. Da das entsprechende Menü nicht als Teil der grafischen Schnittstelle der Anwendung angesehen werden kann, entgeht es folglich der Analyse durch die Analysekomponente. Um dennoch die entsprechenden Behandlungsroutinen aufrufen zu können, kann verfahren werden, indem das entsprechende Ereignis direkt von der Fernsteuerungskomponente generiert wird und dann auf dem normalen Weg durch das GUI System an die Anwendung gelangt.

Generell stellt die Generierung von Ereignissen über die Programmierschnittstelle des GUI Systems im Konzept der hier entworfenen Fernsteuerungskomponente eine Ausnahme dar, die im Zusammenhang mit der realisierten Integrationsumgebung nur dann Verwendung finden soll, wenn, wie im Falle der Systemmenüs, eine direkte Steuerung der Behandlungsroutinen der GUI Schnittstelle nicht möglich ist. Letztlich wirkt die Fernsteuerungskomponente jedoch als eine Art Abstraktionsschicht zwischen der Integrationsumgebung auf der einen Seite und dem GUI System auf der anderen Seite. Aus der Sicht der Integrationsumgebung ist es daher unerheblich, auf welche Weise die Anwendung ferngesteuert wird. Es ist Sache der Fernsteuerungskomponente, die an sie übermittelten Befehle zur Auslösung bestimmter Ereignisse umzusetzen. Während die direkte Steuerung von Behandlungsroutinen zwar bevorzugt wird, wäre ein vollständiges Ausweis-

chen auf die Nutzung der Warteschlange des GUI Systems durchaus möglich. Dieses könnte beispielsweise dann notwendig und daher sinnvoll sein, wenn eine direkte Identifikation der GUI Elemente nicht möglich ist oder das GUI System keinen direkten Zugriff auf die von den GUI Elementen implementierten Behandlungsroutinen zulässt. Zur Unterstützung solcher GUI Systeme könnte die Fernsteuerungskomponente dann entsprechend angepasst werden.

Die direkte Steuerung der Behandlungsroutinen der in der grafischen Benutzungsschnittstelle der Anwendung enthaltenen GUI Elemente entspricht in idealer Weise den zuvor formulierten Bedingungen an die Fernsteuerung im Rahmen des gewünschten Konzepts:

- Es ist plattformneutral, da nicht auf der Ereignisverwaltung des GUI Systems aufgesetzt wird und daher die Synthetisierung entsprechender Ereignisse entfallen kann. Da die Steuerung der Anwendung darüber hinaus in der Fernsteuerungskomponente isoliert werden kann, ist im Bedarfsfall auch ein anderweitiger Steuerungsmechanismus möglich, so dass insbesondere die Entkopplung zwischen Ausgangsplattform sowie Zielplattform gewährleistet wird.

- Es ist flexibel, da auch im Nachhinein noch weitere Ausgangsplattformen hinzugenommen werden können. Schlimmstenfalls muss hierfür nur die innere Funktionalität der Fernsteuerungskomponente ersetzt werden, während die Schnittstelle hin zur Integrationsumgebung gleich bleiben kann.

- Es ist realisierungstransparent, da die Fernsteuerung in jedem Falle ohne eine Modifikation der Anwendung auskommt. Entweder werden mittels Introspektion die Behandlungsroutinen der GUI Elemente aufgerufen oder, falls dieses nicht möglich ist, die entsprechenden Ereignisse im GUI System erzeugt.

- Es ist nutzungstransparent, da die Anwendung die entfernte Nutzung nicht wahrnimmt. Werden die Behandlungsroutinen von der Fernsteuerungskomponente aufgerufen, so geschieht dieses auf dieselbe Weise, wie es durch das GUI System erfolgen würde. Bei Generierung von Ereignissen erfolgt der Aufruf der Behandlungsroutinen sogar durch das GUI System selbst, so dass auch in diesem Falle die Anwendung keine Kenntnis von ihrer entfernten Nutzung erhält.

- Es ist schließlich generisch, da der Mechanismus auf einer grundsätzlichen Eigenschaft von GUI Systemen basiert, nämlich der Erzeugung und Verarbeitung von Ereignissen in Ereigniswarteschlangen bzw. dem Aufruf von Behandlungsroutinen in den von jenen Ereignissen betroffenen GUI Elementen. Es muss entsprechend keine Optimierung hinsichtlich einer bestimmten Anwendung erfolgen, und der Mechanismus kann letztlich für alle Anwendungen Verwendung finden, die auf der Basis eines bestimmten GUI Systems realisiert wurden.

Nachdem nunmehr die konzeptionellen Grundlagen der Analysekomponente, der Transformationskomponente sowie der Fernsteuerungskomponente dargestellt

worden sind, soll deren konkrete Umsetzung im Rahmen dieser Arbeit näher beschrieben werden. Der folgende Abschnitt bleibt dabei jedoch zunächst auf einem recht hohen Abstraktionsniveau. Er beschreibt die Zusammensetzung der Komponenten, ohne dabei auf konkrete Realisierungsdetails einzugehen. Deren Beschreibung erfolgt dann erst in Kapitel 6.

4.3.3 Realisierung der Abstraktionsschicht

Wie sieht nun die definierte Zwischenrepräsentation aus und wie ist diese in das Zusammenspiel von Schnittstellenanalyse und Fernsteuerung eingebettet? Dieser Abschnitt soll hierüber Auskunft geben. Es wird zunächst auf die entworfene Sprache zur abstrakten Beschreibung von grafischen Benutzungsschnittstellen eingegangen. Danach wird dann die in der Verschmelzung von Analyse und Fernsteuerung entstandene Adaptionskomponente eingegangen und schließlich auf wesentliche Aspekte der Transformation von Schnittstellenbeschreibungen in konkrete Zielrepräsentationen.

Als wesentliche Merkmale einer Beschreibungssprache wurde im vorigen Abschnitt deren Plattformneutralität, Ausdrucksstärke und Flexibilität gefordert. Des Weiteren wurde erläutert, dass sie zur Speicherung von Informationen über GUI Elemente, über Container– sowie Layout–Elemente sowie zur Speicherung von Meta–Informationen dienen soll, die zwischen Ausgangsplattform und Zielplattform ausgetauscht werden müssen. Im Sinne einer Entkopplung zwischen beiden Plattformen sollte dabei nach Möglichkeit die Plattformneutralität betont werden, zumindest jedoch die Erweiterbarkeit hinsichtlich der Unterstützung weiterer Ausgangs– bzw. Zielplattformen in jedem Falle erhalten werden.

Die im Rahmen dieser Arbeit entstandene Beschreibungssprache JSML[12] erfüllt weitestgehend die aufgestellten Forderungen. In der für XML üblichen Weise sind in JSML für jeden der Bereiche der GUI–Beschreibung, Layout–Beschreibung sowie der Kodierung von Meta–Informationen Bezeichner (Tags) definiert, die hier im Überblick skizziert werden sollen.

JSML umfasst alle notwendigen Elemente, um eine grafische Schnittstelle auszudrücken, wie sie auf ein entferntes Endgerät übertragbar ist. Zu den vorhandenen GUI–Elementen gehören unter anderem:

- Schaltflächen (Checkboxen, Radiogruppen, Auswahlknöpfe usw.)
- Listen (sog. Comboboxen, Auswahllisten usw.)
- Tabellen
- Textmarken, Textfelder
- Bäume

[12] Am Ausgangspunkt der Entwicklung des Konzepts zur benötigten Beschreibungssprache stand eine Analyse des auf der Programmiersprache Java aufsetzenden GUI Systems ‘SWING‘. Dieser Tatsache wurde mit der Bezeichnung der Beschreibungssprache als *Java Swing Markup Language* (JSML) Rechnung getragen.

Darüber hinaus ist JSML derartig gestaltet, dass eine Erweiterung des Sprachumfangs jederzeit möglich ist, um beispielsweise die zusätzlichen GUI–Elemente eines weiteren zu unterstützenden GUI–Systems bzw. einer weiteren Ausgangsplattform abbilden zu können.

In der Darstellung von Layouts wird in der Sprache konzeptbedingt relativ stark abstrahiert. Semantische Zusammenhänge können mittels Container–Strukturen ausgedrückt werden. Innerhalb eines Containers können beliebig viele GUI–Elemente vorhanden sein und diese so in einen logischen Zusammenhang zueinander gestellt werden. Da innerhalb eines Containers darüber hinaus auch weitere Container vorhanden sein können, kann entsprechend eine beliebige Schachtelung von Gruppen von Elementen erzielt werden. Mittels Containern kann die Aufteilung einer grafischen Schnittstelle in horizontaler bzw. vertikaler Richtung ausgedrückt werden, wobei die Elementgruppen als Schlüssel hierzu fungieren. Es hat sich gezeigt, dass angesichts der stark limitierten Möglichkeiten zur Anordnung von GUI–Elementen auf dem Anzeigegerät eines mobilen Geräts dieser Granularitätsgrad in der überwiegenden Zahl der Fälle als ausreichend anzusehen ist. Da für Benutzungsschnittstellen auf mobilen Geräten oftmals eine deutlich von herkömmlichen Schnittstellen unterschiedliche Aufteilung vorgenommen wird, bei der lediglich die semantischen Zusammenhänge der einzelnen Elemente der Schnittstelle wesentlich sind, wurde entsprechend vornehmlich darauf geachtet, dass diese Zusammenhänge darstellbar sind. Die in GUI–Systemen vorhandenen Mechanismen zur Behandlung von Größenänderungen des Anzeigefensters usw. sind in diesem Falle meist nicht notwendig. Entsprechend wurde in JSML auf die Ausgestaltung komplexerer Ausdrucksformen für Layouts verzichtet. Aufgrund seines modularen Aufbaus können diese jedoch im Bedarfsfall auch nachträglich noch in den Sprachumfang integriert werden.

Neben den unmittelbar aus der Benutzungsschnittstelle herrührenden Aspekten der GUI–Elemente sowie deren Anordnung in einem Layout dient die Sprache JSML darüber hinaus zur Kodierung von Meta–Informationen. Wie bereits erläutert, dienen diese zur Übertragung von Informationen, die im Zusammenhang mit der Kopplung zwischen Anwendung und Unterstützungsumgebung zum Tragen kommen, die jedoch nicht direkt der eigentlichen Schnittstelle zugerechnet werden können. Teilweise werden diese Informationen als Parameter von Schnittstellenelementen übertragen. Dieses ist z.B. bei der Titelzeile des Anwendungsfensters der Fall oder bei der laufenden Nummer des Analysevorgangs. Beide werden in den Code des Applikationselements integriert. In anderen Fällen wird Meta–Information auch in eigenen JSML–Elementen übermittelt. Als Beispiel hierfür kann das Gateway–Element herhalten, das zur Übertragung von Informationen über die aktive Sitzung und zu deren unmittelbarer Steuerung verwendet wird. So werden im Gatway–Element Befehle zur erneuten Durchführung der Analyse, zur Beendigung einer Anwendung oder zum vorübergehenden Stoppen der Anwendung an die Adaptionskomponente übergeben, welche diese sodann weiter auswerten kann.

Anhand von Beispielen soll die Sprache JSML nunmehr näher betrachtet werden. Abbildung 4.18 zeigt zunächst den schematischen Aufbau einer Schnittstellenbeschreibung in JSML. In der bereits beschriebenen und für GUI Systeme typischen Art erfolgt die Kodierung von Schnittstelleninformationen als Schachtelung der dargestellten Elemente. Als Basiselement dient in JSML immer das `Application`–Element. In diesem können sodann die GUI– bzw. Strukturele-

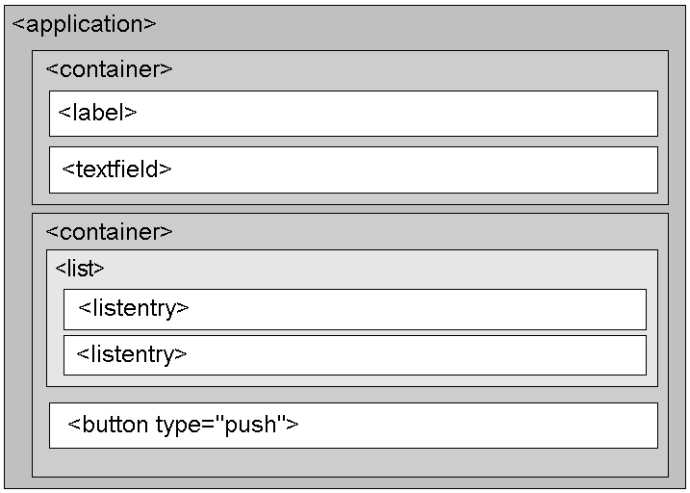


Abbildung 4.18: Struktur einer Schnittstellenbeschreibung in JSML

mente so folgen, wie sie durch die abzubildende Schnittstelle vorgegeben werden. In diesem Fall enthält die Schnittstelle zwei Container–Elemente, die jeweils konkrete GUI Elemente enthalten. Das erste Container–Element beherbergt ein Textfeld („Label“) sowie ein Eingabefeld („Textfield“). Das zweite Container–Feld enthält ein Listenfeld („List“) sowie einen Auswahlknopf („button“). Das Listenfeld schließlich enthält seinerseits wiederum mehrere Felder, nämlich zwei Einträge in die Liste („Listentry“–Felder).

Eine Darstellung des JSML Quelltextes, welcher der in Abbildung 4.18 beschriebenen Struktur entspricht, ist in Abbildung 17 dargestellt. Dieser Quelltext stellt das Resultat der Analyse der zugrunde liegenden Schnittstelle dar und dient als Eingabe in die Transformationskomponente.

Deutlich ist in der Abbildung die Zusammensetzung einer Beschreibung in JSML aus GUI–Elementen, Layout–Elementen sowie Meta–Information erkennbar. Der konkreten Kodierung von GUI–Elementen dienen Tags wie `listentry`, `button` oder `textfield`. Sie sind direkt der analysierten Schnittstelle entnommen. Hinzu kommen `container`, die als semantische Klammern der GUI–Elemente dienen. So sind im ersten Container ein Textfeld sowie ein Eingabefeld zusammengefasst, im zweiten Container eine Liste sowie ein Auswahlknopf. Darüber hinaus ist zusätzlich erkennbar, dass GUI–Elemente selbst als Container auftreten können: Das Element `list` enthält seinerseits weitere Elemente, eben die Einträge in die dargestellte Liste. Schließlich sind im dargestellten JSML–Code Meta–Informationen kodiert. Das Tag `application` selbst kann als eine Art von Meta–Information aufgefasst werden. In seinen Parametern werden Informationen über den Titel des Anwendungsfensters, über den Analysedurchgang sowie über den internen Namen der Anwendung kodiert. Aber auch GUI–Elemente enthalten Informationen, die zur internen Verarbeitung benötigt werden: Jedes GUI–Element verfügt in JSML über eine eindeutige Identifikation, die ihm u.U. erst im Laufe der Analyse beigeordnet wird.

Wie bereits angedeutet, sind die Funktionalität zur Analyse sowie zur Fernsteue-

```
<?xml version="1.0">
<!DOCTYPE application>

<application name="Ihr Name" title="frame0" scan="2">
  <container id="1">
    <label text="Name:"/>
    <textfield id="x234893675" text="administrator">
  </container>
  <container id="2">
    <list id="x432648765" type="list" select="single">
      <listentry selected="false" text="stefan"/>
      <listentry selected="true" text="administrator"/>
    </list>
    <button id="x363329571" text="Ok" type="push"/>
  </container>
</application>
```

Abbildung 4.19: Beispielhafte Schnittstellenbeschreibung in JSML

rung der Anwendungsschnittstelle in einer gemeinsamen Adaptionskomponente zusammengefasst, die aufgrund ihrer Überwachungsfunktion gegenüber der Anwendung als *Application Shadow* bezeichnet wird. Der Application Shadow (oder kurz: Shadow) nimmt die Analyse– und Steuerungsaufgabe in zwei Teilkomponenten wahr, nämlich dem *Scanner* und dem *Updater*. Der Scanner dient der Untersuchung der Anwendungsschnittstelle. Er durchläuft (d.h. „scannt“) die in einem Baum hierarchisch angeordneten GUI–Elemente und erzeugt hieraus eine interne Repräsentation der Schnittstelle in einem DOM–Baum (siehe oben), die dann innerhalb des Shadows in eine textuelle JSML Repräsentation umgeformt und sodann an andere Teile der Integrationsumgebung weitergereicht werden kann.

Der Updater nimmt auf der anderen Seite Befehle entgegen, die über das Netzwerk von der Integrationsumgebung an den Shadow übergeben werden. Solche Befehle können die Benutzung der Schnittstelle selbst betreffen (z.B. wenn ein Element der Schnittstelle auf dem entfernten Endgerät modifiziert wurde) oder auch von der Integrationsumgebung generiert werden. Dieses geschieht beispielsweise dann, wenn über einen definierten Zeitraum keine Aktualisierungen der Schnittstelle vom Endgerät angefordert wurden und die Integrationsumgebung entsprechend annehmen muss, dass die Verbindung permanent unterbrochen wurde. In diesem Fall wird dann von der Sitzungssteuerung der Befehl an den Shadow übergeben, die Anwendung zu beenden.

Wie aus Abbildung 4.20 ersichtlich ist, verfügt der Shadow darüber hinaus über zwei weitere funktionale Komponenten, nämlich die so genannte *ID Factory* sowie den *App Loader*. Die ID Factory dient dem Zweck, eine eindeutige Identifikation der in der Benutzungsschnittstelle vorhandenen GUI–Elemente zu ermöglichen und einen effizienten Zugriff auf die identifizierten Elemente durch den Updater zu schaffen. Beim Übergang von der ursprünglichen Repräsentation der

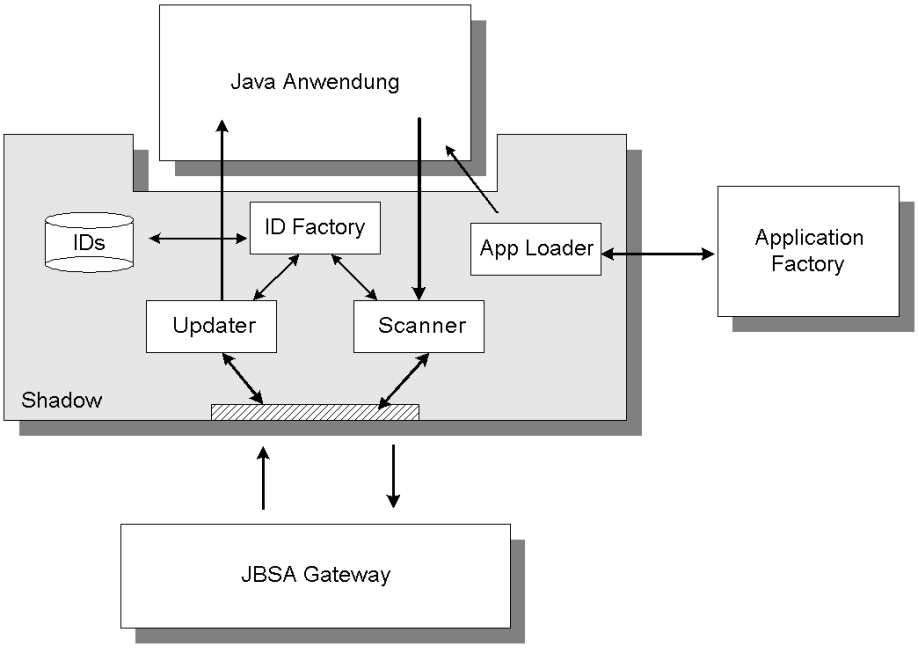


Abbildung 4.20: Schematische Darstellung des Application Shadows

Schnittstelle in die interne Darstellung im DOM–Baum bzw. in JSML muss darauf geachtet werden, dass jederzeit der Rückschluss von der Darstellung eines GUI–Elements in der internen Form auf die ursprüngliche GUI–Komponente möglich ist. Nur wenn dieses gewährleistet ist, kann eine auf dem Endgerät durchgeführte Aktion auf der dort dargestellten Schnittstelle tatsächliche Änderungen an der eigentlichen Anwendung hervorrufen. Jedes GUI–Element muss also mit einer eindeutigen Kennung belegt werden, die während der gesamten Laufzeit der Anwendung beibehalten wird und die eine solche Zuordnung ermöglicht. Dabei ist zu beachten, dass an die zu generierenden Kennungen besondere Anforderungen gestellt werden müssen, damit sie in JSML (als einem XML Dialekt) und insbesondere auch in hiervon abgeleiteten Zielrepräsentationen wie HTML, WML, VoiceML usw. ausgedrückt werden können. Beispielsweise können nicht einfach die in der konkreten GUI Implementation vorhandenen Objekt–Identitäten (z.B. sehr große Zahlen) verwendet werden, da diese unter Umständen negative Vorzeichen aufweisen, die in XML basierten Sprachen wie WML nicht korrekt kodiert werden können. Für manche GUI–Systeme könnte die Objekt–Identität eventuell gar nicht als alphanumerischer Wert ausdrückbar sein, wodurch eine direkte Benutzung als Kennung grundsätzlich unmöglich wäre.

Um derartigen Problemen begegnen zu können, besteht die Hauptaufgabe der ID Factory darin, eine Übersetzung zwischen der ursprünglichen Objekt–Identität und einer für die weitere Verarbeitung gültige Kennung zu schaffen. Wird im Rahmen der Analyse ein GUI–Element gefunden, so wird für dieses eine eindeutige Kennung generiert und diese gemeinsam mit der ursprünglichen Identität gespeichert. Wird im weiteren Verlauf ein Befehl an den Shadow ge-

sandt, der ein bestimmtes GUI–Element betrifft, so kann die im Befehl enthaltene Kennung in die ursprüngliche Objekt–Identität zurückübertragen werden, damit das richtige Element modifiziert werden kann. Des Weiteren dient die ID Factory einer signifikanten Beschleunigung des Verarbeitungsprozesses: Sofern dieses vom zugrunde liegenden GUI–System unterstützt wird, kann über die eigentliche Objekt–Identität eines GUI–Elements hinaus auch eine Referenz auf das Objekt verwaltet werden. Auf diese Weise können sodann notwendige Änderungen an der Benutzungsschnittstelle direkt durch Aufrufe an der gespeicherten Referenz erfolgen, ohne dass hierfür eine aufwendige Suche in der GUI Struktur notwendig wäre.

Der App Loader wird im Rahmen des Programmstarts benötigt, um die notwendige Kopplung zwischen Shadow und Anwendung zu erzielen. Aus systemtechnischen Gründen ist es notwendig, dass sowohl Anwendung als auch der zugehörige Shadow in derselben Laufzeitumgebung (z.B. in derselben Java Virtual Machine) ablaufen. Um dieses zu erreichen, erfolgt ein so genanntes „Bootstrapping“ bei dem zunächst eine Anwendung gestartet und mit ihr die Laufzeitumgebung initialisiert wird. Die zweite Anwendung wird sodann von der zuvor gestarteten Anwendung aufgerufen und von dieser in deren Laufzeitumgebung integriert. Im konkreten Fall des Application Shadows wird das Bootstrapping von der (im nachfolgenden Abschnitt beschriebenen) Application Factory koordiniert. Soll eine Anwendung zur weiteren Benutzung in der Integrationsumgebung mit einem Application Shadow gekoppelt werden, startet die Application Factory zunächst einen leeren Shadow und übergibt dessen App Loader die Informationen, welche Anwendung dieser analysieren und steuern soll. Der App Loader startet diese Anwendung sodann mit Hilfe der von der Programmiersprache angebotenen Systembefehle und integriert sie so in die eigene Laufzeitumgebung. Er übergibt sodann die Kontrolle an den eigentlichen Shadow, der seine Gegenwart (d.h. die Aufrufschnittstelle des Scanners sowie des Updaters) der Integrationsumgebung mitteilt. In der Folge kann dann die Integrationsumgebung über die Schnittstelle des Shadows Abbildungen in der Zwischenrepräsentation JSML erfragen oder Befehle weiterleiten, die vom entfernten Endgerät vermeldet wurden.

Wie bereits beschrieben besteht die Aufgabe der in dieser Arbeit vorgestellten Integrationsumgebung u.a. darin, Momentaufnahmen der grafischen Schnittstelle einer laufenden Anwendung in einem abstrakten Beschreibungsformat zu erfassen und diese Beschreibungen sodann in konkrete Repräsentationen zu überführen, die auf einer möglichst großen Zahl von Endgeräten angezeigt und benutzt werden können. Die Überführung der abstrakten Zwischenrepräsentation in eine konkrete Zielsprache wird als *Transformation* bezeichnet. Ihr kommt innerhalb der Integrationsumgebung eine zentrale Rolle zu, denn sie bildet quasi das Bindeglied zwischen der Ausgangsplattform auf der einen Seite und der Zielplattform auf der anderen Seite. Nach Abbildung 4.15 ist die Transformation dem Kern der Abstraktionsschicht zuzurechnen, welche durch die Integrationsumgebung zwischen beiden Plattformen etabliert wird.

Die Transformation der abstrakten Zwischenrepräsentation in eine bestimmte Zielsprache wird von der Transformationskomponente (oder kurz: vom Transformator) durchgeführt. Der Transformator bedient sich dabei eines Regelsatzes, der jedem in der Zwischenrepräsentation definierten GUI–Element eine oder mehrere Anweisungen in der Zielsprache zuordnet. In einem Übersetzungsprozess durchläuft der Transformator die in JSML vorliegende Beschreibung der

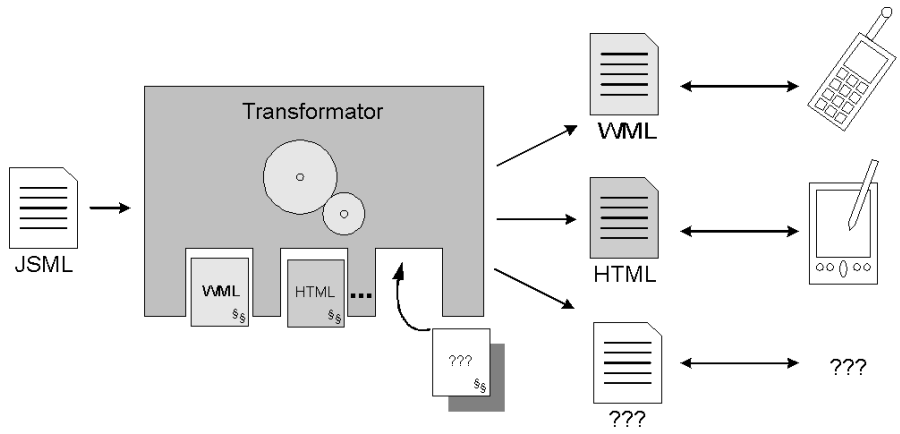


Abbildung 4.21: Schematische Darstellung des modularen Transformators

Schnittstelle, die er als Eingabe erhalten hat, und setzt jedes in der Beschreibung gefundene Element in die vom Regelsatz vorgeschriebene Folge von Anweisungen in der Zielsprache um.

Wie im vorigen Schritt der Analyse wird auch in der Transformation keine *semantische* Bearbeitung durchgeführt, sondern lediglich auf Basis der im Eingangstext erkannten Schlüsselworte agiert. Im Zusammenhang mit der Transformation bedeutet dies, dass beispielsweise ein gefundener Auswahlknopf mit der Aufschrift „Hilfe“ nicht anders umgesetzt wird als ein Auswahlknopf mit der Aufschrift „Abbruch“, obwohl prinzipiell aus der Beschriftung auf die Funktion rückgeschlossen werden könnte. Da, wie bereits angedeutet, aus der rein mechanischen Analyse einer grafischen Schnittstelle keine *eindeutigen* Aussagen über die Bedeutung der gefundenen Elemente abgeleitet werden können, wird hierauf sowohl in der Analyse selbst wie auch in der Transformation verzichtet. Es wird im Regelwerk lediglich bestimmt, wie das Element *Auswahlknopf* in die Zielrepräsentation umgesetzt werden soll.

Abbildung 4.21 stellt den Transformationsprozess zunächst einmal schematisch dar. Ein mittels einer Menge von Regelsätzen parametrisierter Transformator erhält eine abstrakte GUI–Beschreibung und übersetzt diese unter Zuhilfenahme eines bestimmten Regelsatzes in eine konkrete Repräsentation, die dann auf dem Endgerät dargestellt werden kann, für welches der verwendete Regelsatz gültig ist. Bei der abstrakten Beschreibung handelt es sich dabei um das zuvor im Analyseschritt erzeugte JSML Dokument, das vom Application Shadow erfragt und zur weiteren Verarbeitung an den Transformator übergeben wird.

Um die Forderung nach weitestgehender Flexibilität hinsichtlich der unterstützten Zielplattformen erfüllen zu können, ist der Transformator modular aufgebaut. Jederzeit können weitere Regelsätze (im Folgenden auch als *Stylesheet* bezeichnet) erzeugt und hinzugefügt werden. Noch zur Laufzeit der Integrationsumgebung kann so die Einbindung einer weiteren, zur Entwurfszeit der Umgebung unbekannten Zielplattform gewährleistet werden.

Der zu verwendende Regelsatz wird dem Transformator von der Integrationsumgebung jeweils bei Übergabe des JSML Dokuments vorgegeben. Der Trans-

formator wählt hierzu das entsprechende Stylesheet aus, transformiert anhand der darin enthaltenen Regeln die Zwischenrepräsentation in eine konkrete Zielrepräsentation und übergibt diese zur weiteren Verarbeitung an die Integrationsumgebung zurück. Über den in Abschnitt 4.3.4 beschriebenen Geräteadapter gelangt die so erstellte Beschreibung dann an das Endgerät, wo sie dargestellt werden kann. Ein sehr wichtiges Merkmal des Transformators ist dabei die Flexibilität, in welcher die Verwendung des Stylesheets bestimmt werden kann. So kann das verwendete Stylesheet auch während eines Anwendungslaufs noch geändert werden. Wird auf eine Anwendung z.B. nach deren Start zunächst über ein Mobiltelefon zugegriffen, so wird zunächst das Stylesheet für Mobiltelefone (z.B. zur Transformation in die Zielrepräsentation WML) ausgewählt. Soll im weiteren Verlauf die Verarbeitung nicht länger vom Mobiltelefon, sondern von einem PDA aus erfolgen, so kann ein neues Stylesheet (z.B. für die Zielrepräsentation HTML) ausgewählt werden, ohne dass die Anwendung hierfür gestoppt und neu gestartet werden müsste. Es wird lediglich zwischen zwei Transformationszyklen der Befehl an die Integrationsumgebung erteilt, das entsprechende Stylesheet auszuwählen. Bei der nächsten Abfrage erfolgt dann die Umsetzung in eine andere Zielrepräsentation. Stylesheets stellen den Kern des Transformationsprozesses dar. Sie machen die Generizität des Ansatzes aus, da sie, je nach dem zu einem bestimmten Zeitpunkt verwendeten Stylesheet, eine Umsetzung in eine andere Zielrepräsentation ermöglichen. Das Modularisierungsprinzip ist in Abbildung 4.21 skizziert.

Das Aussehen der Zielrepräsentation ist vom verwendeten Mechanismus geprägt: Da zur Transformation auf die in XML übliche Übersetzung mittels XSL (Extensible Stylesheet Language) gesetzt wird, können nur Zielrepräsentationen berücksichtigt werden, welche sich mit XSL erzeugen lassen. Insbesondere gilt hierbei, dass XSL nur die Transformation eines XML–Dialekts in einen anderen XML–Dialekt erlaubt (vgl. [ABK+00], S. 376). Bei der Zielrepräsentation muss es sich insofern wie bei der abstrakten Zwischenrepräsentation JSML um einen XML–Dialekt handeln. Bei WML und HTML ist dieses offensichtlich der Fall. Andere Zielrepräsentationen, z.B. gesprochene Sprache, können ebenso offensichtlich nicht als XML–Dialekt bezeichnet werden. Während dieses zunächst als deutliche Einschränkung vermutet werden kann, ist dieses tatsächlich nicht der Fall: Da die Kombination aus Transformator und Geräteadapter auch die Verkettung von Transformationen erlaubt, gilt die Forderung nach einer XML–Grundlage nur für die Form, in welcher die Zielrepräsentation die Integrationsumgebung verlässt, *nicht* jedoch unbedingt deren endgültige Form. Ein konkretes Beispiel hierfür wäre die Übersetzung in gesprochene Sprache. Innerhalb der Integrationsumgebung würde hierbei die Transformation von JSML in die Auszeichnungssprache VoxML erfolgen, einen XML–Dialekt, der zur Kodierung von Sprache verwendet wird [Mot99]. Dokumente in VoxML dienen ihrerseits als Eingabe für einen Interpreter, der diese in Phoneme umsetzt, die auf einem Lautsprecher ausgegeben werden können. Aus der Sicht der Integrationsumgebung wird VoxML als Zielrepräsentation angesehen und das JSML Dokument demgemäß in ein VoxML umgesetzt. Dieses wird dann über den Geräteadapter an den VoxML Interpreter weitergeleitet, der dann die endgültige Umsetzung vornimmt. Das Resultat dieser Umsetzung muss, wie aus dem Beispiel offensichtlich ergeht, nicht mehr zwingend auf einem XML–Dialekt beruhen.

Stylesheets sind zwar für eine bestimmte Plattform spezifisch, hängen jedoch

nicht von einer bestimmten Anwendung ab[13]. Dieses liegt vornehmlich darin begründet, dass keine über die bloße Struktur hinausgehenden Informationen über die Ausgangsrepräsentation erfasst werden können. Es können nur Informationen über die Existenz und Position von GUI–Elementen in der Zwischenrepräsentation kodiert werden, nicht jedoch solche über deren Bedeutung. Entsprechend kann auch nur sehr begrenzt die von Anwendung zu Anwendung unterschiedliche Bedeutung erkannt werden, die GUI–Elementen beigemessen wird.

Im einfachsten Fall findet im Zuge der Transformation eine Umsetzung eines bestimmten GUI–Elements der Ausgangsrepräsentation in ein entsprechendes Element in der Zielrepräsentation statt. So könnte beispielsweise ein in der Benutzungsschnittstelle des Java–Programms befindlicher Auswahlknopf in einen entsprechenden Auswahlknopf in HTML umgesetzt werden. Oftmals ist diese direkte Umsetzung jedoch nicht möglich, wenn die Zielrepräsentation nicht über eine vergleichbare Ausdrucksmächtigkeit wie die Ausgangsrepräsentation verfügt. In einem solchen Fall kann es dann notwendig sein, aus einem im Rahmen der Analyse erkannten GUI–Element mehrere Elemente in der Zielrepräsentation zu erzeugen.

Anhand eines Beispiels soll dieser Fall verdeutlicht werden. Während es im GUI–System SWING in der tabellarischen Anordnung von Einträgen die Möglichkeit zur Auswahl eines oder mehrerer Einträge gibt, die optisch besonders herausgestellt werden, ist dieses in HTML nicht vorgesehen. Im Zuge der Transformation muss demgemäß eine Umsetzung in eine alternative Darstellungsform von vergleichbarem Aussehen bzw. von vergleichbarer Aussagekraft stattfinden. Eine mögliche Umsetzung besteht darin, die Tabellenzeilen in der Zielrepräsentation jeweils um ein Markierungsfeld zu erweitern, das je nach Status der ursprünglichen Tabellenzeile ausgewählt ist. Kann in der Ursprungstabelle immer nur ein Element ausgewählt sein, müssen die Markierungsfelder zusätzlich als Radioknöpfe umgesetzt werden. Im Gegensatz zur ursprünglichen Darstellung ist die Zielrepräsentation in diesem Fall also um Markierungsfelder erweitert worden, die in dieser Form zuvor nicht vorhanden waren. Es ist natürlich in entgegengesetzter Richtung notwendig, die Auswahl der hinzugekommenen Markierungsfelder in die Auswahl einer bestimmten Tabellenzeile zurückzuübersetzen. Diese Aufgabe kommt dem oben beschriebenen Updater zu, der in Zusammenarbeit mit der ID Factory die Kennung des ausgewählten Markierungsfeldes als zur Tabelle zugehörig erkennt und demgemäß den Status der Tabelle aktualisiert.

Ähnlich wie der Application Shadow, der zur Analyse der grafischen Benutzungsschnittstelle und der hieraus resultierenden Generierung der abstrakten Zwischenrepräsentation dient, ist auch die Transformationskomponente als eine in sich abgeschlossene Funktionseinheit innerhalb der Integrationsumgebung konzipiert. So kann der rechenintensive Transformationsprozess auf einen leistungsfähigen Rechner ausgelagert werden oder zwecks zusätzlicher Leistungssteigerung in einer besonders performanten Programmiersprache realisiert werden. Da die Unterstützung verschiedener Zielrepräsentationen anhand voneinander unabhängiger Stylesheets realisiert wird, ist darüber hinaus auch eine massive Verteilung denkbar, bei der jeder Zielplattform ein eigener Netzwerkknoten

[13] In eng abgesteckten Grenzen lässt sich die Optimierung auf bestimmte Anwendungen über den Einsatz eines Klassifikationssystems erzielen, das Anwendungen einer bestimmten Klasse zuordnet und die Auswahl des zu verwendenden Stylesheets von der Anwendungsklasse abhängig macht. Auf dieses Vorgehen wird im weiteren Verlauf noch etwas näher eingegangen.

zugewiesen wird, auf dem die Transformation sodann stattfinden kann.

Während eine konkrete Beschreibung des Transformationsprozesses sowie des Aufbaus des Transformators und der für die Transformation notwendigen Stylesheets erst in Kapitel 6 erfolgen wird, soll an dieser Stelle zunächst mit der überblickartigen Skizzierung des gesamten Transformationsprozesses fortgefahren werden.

Wie oben bereits angedeutet wurde, besteht der Integrationszyklus aus den folgenden Schritten:

1. Analyse der Anwendungsschnittstelle
2. Generierung der Zwischenrepräsentation
3. Transformation der Zwischenrepräsentation in eine konkrete Zielrepräsentation
4. Übertragung der generierten Zielrepräsentation zum Endgerät
5. Entgegennahme von Befehlen vom Endgerät
6. Weiterleitung aufgefangener Befehle an die Anwendung

Konzeptionell lassen sich diese Schritte zu drei Hauptaufgaben der Integrationsumgebung zusammenfassen, die in Abbildung 4.22 verbildlicht sind:

- Inhaltserzeugung (*Content Generation*)
- Inhaltstransformation (*Content Transformation*)
- Inhaltsübertragung (*Content Mediation*)

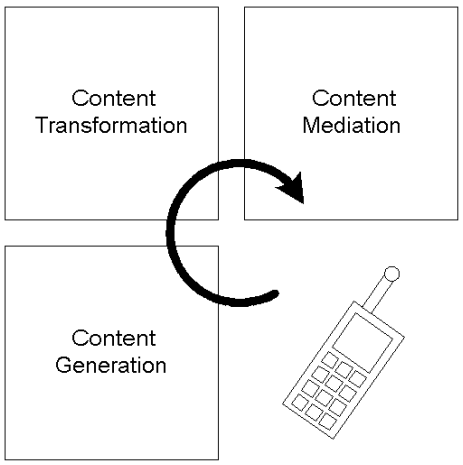


Abbildung 4.22: Hauptaufgaben der Integrationsumgebung

Die Schritte der Analyse und der Weiterleitung der aufgefangenen Befehle an die Anwendung (*Content Generation*) sowie der Generierung der Zwischenrepräsentation (*Content Transformation*) wurden zuvor bereits beschrieben. Für

die Content Generation ist der bereits eingeführte Application Shadow verantwortlich. Die Content Transformation wird wie soeben skizziert vom Transformator unter Verwendung geeigneter Stylesheets durchgeführt.

Im folgenden Abschnitt soll nun vertieft werden, wie die so gewonnene konkrete Umsetzung der Zwischenrepräsentation im Schritt der *Content Mediation* an das Endgerät zur entfernten Darstellung sowie Nutzung weitergeleitet werden kann.

4.3.4 Universelle Geräteintegration mittels generischen Adaptern

In ähnlicher Weise wie der Application Shadow zur Kopplung zwischen Anwendung und Integrationsumgebung dient, ist es Aufgabe des *External Communications Adapters* (ECA), dieses zwischen Integrationsumgebung und Endgerät zu tun. In beiden Fällen wird eine Adaption durchgeführt, die die jeweiligen Spezifika der einzubindenden Seite vor der anderen Seite verdeckt.

Dem ECA kommen hierbei zwei Teilaufgaben zu. Er muss einerseits dafür sorgen, dass die im Zuge der Transformation entstandene Zielrepräsentation zum Endgerät übertragen werden kann. Zum anderen realisiert er den Empfänger, über welchen Befehle vom Endgerät entgegengenommen werden. Es ist Sache des ECAs, derartige Befehle in eine interne Repräsentation umzusetzen und diese dann an die weiteren Komponenten der Integrationsumgebung (und hierbei insbesondere an den Application Shadow) weiterzuleiten. Fasst man beide Aufgaben zusammen, so wird durch den External Communications Adapter insgesamt ein *Nachrichtenkanal* etabliert, der zwischen Endgerät und Integrationsumgebung steht und durch den Informationen zwischen beiden Entitäten ausgetauscht werden.

Während die der Integrationsumgebung zugewandte Seite des Nachrichtenkanals im Kern hinsichtlich des verwendeten Kommunikationsprotokolls festgelegt ist, variiert die dem Endgerät zugewandte Seite je nach Typ des zu integrierenden Geräts. An die Einbindung eines herkömmlichen Telefons, genauer gesagt an die Realisierung eines Sprachkanals, sind offensichtlich andere Forderungen zu stellen als an die Einbindung eines WAP Telefons oder gar eines PDAs. Der für den jeweiligen Kanal zuständige ECA muss die zuvor vom Transformator erhaltene Zielrepräsentation in der geeigneten Form zur Übertragung an das Endgerät vorbereiten, was in erstgenanntem Beispiel die weitere Umsetzung in Phoneme und deren Übertragung über die Telefonleitung bedeuten könnte und im zweiten Beispiel die Übertragung an das Endgerät mittels HTTP.

Im Gegensatz zu anderen Komponenten der Integrationsumgebung besteht der External Communications Adapter nicht nur aus innerhalb der Umgebung selbst realisierten Teilkomponenten, sondern umfasst konzeptionell auch solche funktionalen Teile, die der eigentlichen Umgebung nicht zugerechnet werden können. Dieses lässt sich wiederum an den genannten Beispielen verdeutlichen: Besteht die Aufgabe des ECAs *konzeptionell* darin, die Zwischenrepräsentation vom Kern der Integrationsumgebung an das Endgerät auszuliefern, so kann sie bei näherer Betrachtung in eine (eventuell große) Anzahl von Teilschritten aufgegliedert werden. Im Fall des PDAs mit WAP Browser wären dieses:

1. Bereitstellung der Zwischenrepräsentation über einen HTTP–Server,

2. Umsetzung der vom HTTP–Server übernommenen Zwischenrepräsentation in das WAP Protokoll,
3. Übertragung der Zwischenrepräsentation über das Telefonnetz zum Endgerät und schließlich
4. Auslieferung der Zwischenrepräsentation an den WAP Browser auf dem Endgerät.

Es ist offensichtlich, dass nicht alle Schritte von der Integrationsumgebung selbst realisiert werden können bzw. realisiert werden müssen. Während der erste Schritt der Bereitstellung zumindest noch teilweise in den Aufgabenbereich der Integrationsumgebung fällt, ist bereits der zweite Schritt eher der allgemeinen Infrastruktur zuzurechnen; für den dritten und vierten Schritt ist dieses sogar selbstverständlich der Fall.

Ein ähnlicher Sachverhalt ergibt sich auch für das Beispiel des herkömmlichen Telefons, das über einen Sprachkanal an die Integrationsinfrastruktur angekoppelt wird. Der Sprachkanal ergibt sich hierbei durch folgende Schritte:

1. Übertragung der Zwischenrepräsentation (welche z.B. in VoxML vorliegen könnte) an einen Synthesizer,
2. Umsetzung der Zwischenrepräsentation in Phoneme,
3. Übertragung der Phoneme über das Telefonnetz an das Endgerät.

Auch in diesem Fall ist nur der erste Teilschritt durch die Integrationsumgebung selbst zu erledigen, während der zweite und dritte Schritt durch externe Funktionskomponenten übernommen werden. Da sie dennoch zur Realisierung des Nachrichtenkanals dienen, sollen sie konzeptionell dem External Communications Adapter zugeordnet werden. Dieser zerfällt danach in einen *internen* Teil und einen *externen* Teil. Auf den externen ECA wird fortan nur dort näher eingegangen werden, wo dieses für das Gesamtverständnis notwendig ist. Im Übrigen gilt das Augenmerk vornehmlich der Beschreibung des internen ECAs. Der Einfachheit halber wird der interne ECA dabei verkürzt als „ECA“ bezeichnet.

Der ECA tauscht Nachrichten zwischen Endgerät und Integrationsumgebung durch einen Nachrichtenkanal aus, wobei die Zwischenrepräsentation den Kanal in Richtung Endgerät passiert und in Gegenrichtung Befehle an die Anwendung übertragen werden. Beide Richtungen sollen nun gesondert betrachtet werden, wobei zunächst auf die Übertragung der Zwischenrepräsentation eingegangen wird.

Grundsätzlich lassen sich drei Modi feststellen, in denen Nachrichten zum Endgerät übertragen werden können:

Passiver Modus : Die aus der Transformation hervorgegangene Zwischenrepräsentation wird vom ECA bereitgestellt und kann dann vom Endgerät abgefragt werden (sog. Pull–Prinzip).

Aktiver Modus : Die aus der Transformation hervorgegangene Zwischenrepräsentation wird vom ECA selbst an das Endgerät ausgeliefert (sog. Push–Prinzip).

Semi–aktiver Modus : Die aus der Transformation hervorgegangene Zwischenrepräsentation wird vom internen ECA an den externen ECA übertragen, von wo sie dann vom Endgerät abgefragt werden kann.

Der passive Modus ist von der Funktionsweise des Worldwide Web wohlbekannt. Hierbei werden Dokumente in der Seitenbeschreibungssprache HTML von einem Webserver angeboten und können von einem Endgerät mittels eines Webbrowsers abgefragt werden. Das Endgerät „zieht“ quasi die Informationen über den Kommunikationskanal, bevor es sie darstellt. Der Webserver nimmt hierbei eine passive Rolle ein: Er tritt nur dann in Aktion, wenn eine Aufforderung hierzu an ihn ergeht.

Im Kontext verteilter oder gar mobiler Systeme ist der passive Modus recht einfach zu realisieren, da der Informationsanbieter nicht gezwungen ist, den Zielknoten zu lokalisieren. Es ist Sache des Zielknotens, sich im Bedarfsfall an den Anbieter zu wenden und die Daten von diesem zu erfragen. Entsprechend stellt diese Interaktionsform auch die gängigste Variante des Informationsaustausches dar: Das Worldwide Web basiert genauso hierauf wie das Wireless Application Protocol. Auch in der Realisierung des ECA stellt der passive Modus die einfachste Herangehensweise dar, die zur Realisierung einfacher Nachrichtenkanäle für WWW und WAP realisiert wurde. Hierbei übernimmt der interne ECA die Aufgabe, die Zwischenrepräsentation über einen HTTP–Server anzubieten und an das Endgerät auszuliefern, sofern dieses sich beim HTTP–Server meldet.

Der passive Modus besitzt insbesondere für mobile Anwendungen den Nachteil, dass das Endgerät gezwungen ist, selbst herauszufinden, ob sich eine Änderung der Information ergeben hat, z.B. im Zusammenhang mit der Integrationsumgebung, ob sich die angebotene Zwischenrepräsentation seit dem letzten Zugriff geändert hat. Ist dieses über längere Zeit nicht der Fall, ergibt sich so unter Umständen eine unnötige Kommunikation, die sowohl Energie verbrauchen als auch unnötige Kosten verursachen könnte.

Entsprechend bietet sich hier der aktive Modus an, bei dem die Interaktion zwischen Integrationsumgebung und Endgerät vom ECA ausgeht. Nur wenn sich die Benutzungsschnittstelle der Anwendung geändert hat, wird eine Transformation in die Zielrepräsentation durchgeführt und eine Übertragung der Zwischenrepräsentation an das Endgerät veranlasst. Das Endgerät muss demnach nicht mehr überprüfen, ob sich die Darstellung geändert hat, sondern wird hierüber vom ECA informiert, der nunmehr eine aktive Position einnimmt. Ein solches Vorgehen bietet sich insbesondere in mobilen Kontexten an, wo die Kommunikation nach Möglichkeit zu vermeiden ist. Andererseits setzt es aber auch die Fähigkeit des Endgerätes zur Entgegennahme von Nachrichten voraus, was aufgrund der Komplexität des notwendigen Empfängers selten der Fall ist. Konzeptionell stellt dieses Vorgehen jedoch den zu bevorzugenden Ansatz dar.

Bei näherer Betrachtung stellt sich heraus, dass der aktive Modus oftmals nicht in seiner reinen Form realisiert wird, sondern in einer hybriden Mischung aus aktivem sowie passivem Vorgehen, die am besten als *semi–aktiv* bezeichnet werden kann. Hierbei sendet der Anbieter der Information eine kurze Nachricht an den Empfänger und informiert ihn darüber, dass die eigentliche Information zur Abholung bereitsteht. Der Empfänger kann die Information dann wie im passiven Fall beim Anbieter abfragen. Im Falle des ECAs würde dieses bedeuten,

dass der ECA eine Nachricht an das Endgerät sendet und gleichzeitig die Zwischenrepräsentation zur Abholung bereitstellt. Das Endgerät kann diese dann in der üblichen Weise beim internen ECA erfragen.

Der semi–aktive Modus besitzt den Vorteil, gegenüber dem aktiven Modus einen höheren Grad an Flexibilität aufzuweisen. Da die Übertragung der Information von der Benachrichtigung über deren Verfügbarkeit getrennt ist, kann die vergleichsweise komplexe Benachrichtigung eventuell einfacher realisiert werden. So ist es im konkreten Anwendungsfall der Realisierung eines aktiven Nachrichtenkanals für HTML wesentlich einfacher, die Benachrichtigung über eine Kurzmitteilung (SMS) zu realisieren und dann für die Übertragung der Zwischenrepräsentation das herkömmliche HTTP Protokoll zu verwenden, anstatt eine vollständig aktive HTTP–Variante zu verwirklichen. Der semi–aktive ECA kann gegenüber den übrigen Teilkomponenten der Integrationsumgebung als aktiv auftreten und die tatsächliche Zweiteilung der Kommunikation nach außen verbergen.

Wie bereits angedeutet verfügt die Integrationsumgebung prinzipiell nicht über nur einen ECA, sondern kann je nach unterstützten Endgeräten über eine beliebige Menge unterschiedlicher Adapter verfügen. Ähnlich wie Application Shadow und Transformator ist auch der ECA als eine in sich eigenständige Komponente der Integrationsumgebung konzipiert und kann nach Bedarf in den Umgebungskern eingeklinkt werden. Wie für die Transformatoren gilt auch für die ECAs, dass diese sogar noch nachträglich, d.h. zur Laufzeit der Umgebung, eingebunden werden können, sofern ein weiterer Gerätetyp unterstützt werden soll. Auch in diesem Fall ist der modulare Ansatz einerseits hinsichtlich der Skalierbarkeit des Systems von Vorteil und trägt andererseits zur weitgehenden Flexibilisierung bei. Je nach Anforderung des zu etablierenden Nachrichtenkanals kann die optimale Plattform bzw. Realisierungssprache zur Implementation des jeweiligen ECAs gewählt werden und dieser dann unter Verwendung eines internen Kommunikationsprotokolls an den Umgebungskern angekoppelt werden.

Wie bereits angedeutet ist jeder ECA für einen bestimmten Kommunikationstyp spezifisch. Denkbar sind ECAs für Sprachkommunikation, für die Interaktion mit WAP– oder HTML–Browsern, für den Austausch von Emails oder auch die Kommunikation per CORBA–RPC. Am Ausgang des vom ECA etablierten Kanals muss ein vom Endgerät akzeptiertes Protokoll verwendet werden, während der Eingang des Kanals die Kommunikation mit einem umgebungsinternen Protokoll ermöglichen muss. Unterscheidet man in seiner Betrachtung des ECAs zwischen einem internen sowie externen Teil, so ist es durchaus möglich, dass der zur eigentlichen Integrationsumgebung gehörige interne Teil für mehrere Kanäle übereinstimmt. So wird im Falle der Unterstützung von HTML sowie WML als Zwischenrepräsentation derselbe Mechanismus verwendet, um die vom Transformator übergebenen Daten über das HTTP Protokoll auszuliefern. Erst in seinem externen Teil unterscheiden sich die ECAs voneinander: Der für WML verwendete ECA muss zusätzlich eine Umsetzung in das WAP Protokoll durchführen, bevor eine Übertragung der Daten an das Endgerät erfolgen kann, während im Falle des für HTML benutzten ECAs eine solche Umsetzung unterbleiben kann.

Vergleicht man den ECA mit dem Application Shadow, so fällt auf, dass beide zur Realisierung einer Abstraktionsschicht dienen. Der Application Shadow

abstrahiert vom verwendeten GUI–System bzw. der analysierten Anwendung, während der ECA vom verwendeten Endgerät abstrahiert. Mit Blick auf die Anwendungssitzung fällt dabei auf, dass eine bestimmte Instanz des Shadows für die gesamte Laufzeit einer Anwendung mit dieser verbunden ist. Der ECA kann andererseits jederzeit gewechselt werden, sofern von einem Nachrichtenkanal bzw. Endgerät zu einem anderen Nachrichtenkanal übergegangen wird. Während des Wechsels vermerkt die Sitzungssteuerung das Fehlen eines aktiven ECAs in der Anwendungssitzung. Wurde ein anderer ECA angekoppelt, so wird dieses nach Abschluss des Wechsels ebenso in der Anwendungssitzung verbucht. Somit kann auf eine bestimmte Anwendung während deren Laufzeit theoretisch zunächst über ein WAP Telefon zugegriffen werden, hiernach über einen PDA mit HTML Browser und schließlich von einem herkömmlichen Telefon. Der ECA wird dabei jeweils gewechselt, während der Shadow bestehen bleibt. Auf Details des Sitzungskonzepts und der zugehörigen Steuerungskomponente wird in Abschnitt 4.3.8 näher eingegangen.

Während insofern klar ist, dass nacheinander mehrere ECAs mit einer Anwendung verbunden werden können und verschiedene ECAs parallel unterschiedliche Anwendungen bedienen können, sieht das zugrunde liegende Konzept nicht vor, dass von verschiedenen ECAs *gleichzeitig* auf dieselbe Anwendung zugegriffen wird. Bildlich gesprochen darf der zwischen Endgerät und Anwendung liegende Nachrichtenkanal zu jedem Zeitpunkt nur über jeweils *einen* Ein– bzw. Ausgang verfügen. Genauso, wie es nicht möglich ist, dass gleichzeitig mehrere Personen die an einen PC angeschlossene Maus zur Bedienung einer Anwendung verwenden, kann die Bedienung über den Umweg der Integrationsumgebung auch nur von einem Endgerät geschehen.

Wie bereits mehrfach angedeutet, stellt der ECA eine Abstraktion zwischen Endgerät und Integrationsumgebung dar: Aus der Sicht des Endgerätes wird in der für das Gerät üblichen Weise auf eine dynamische Informationsquelle zugegriffen. Von der eigentlichen Beschaffenheit der Quelle, d.h. von Analyse, Transformation und Fernsteuerung erfährt das Endgerät dabei nichts.

Eine wichtige Rolle spielt der ECA als Abstraktionsschicht ebenso vom Standpunkt des Transformators bzw. der Integrationsumgebung. Während der Transformator die Zielrepräsentation kennen muss, die als Resultat des Transformationsprozesses generiert werden soll, gibt es Aspekte, die im Zuge des eigentlichen Transformationsprozesses unerheblich sind, die jedoch bei der Integration eines bestimmten Endgeräts beachtet werden müssen. Vor diesen spezifischen Aspekten wird die Integrationsumgebung vom ECA geschützt.

Anhand eines konkreten Problems und seiner Lösung im Kontext der Integrationsumgebung soll das Gesagte verdeutlicht werden. Im Zusammenhang mit der Darstellung von WML auf Mobiltelefonen spielt die maximale Größe eines WML Dokuments eine sehr wichtige Rolle. Übersteigt die Größe des Dokuments die (von Gerät zu Gerät variierende) Kapazität des Seitenspeichers, so wird der Ladevorgang abgebrochen und das gesamte Dokument verworfen. Greifen beispielsweise zwei Geräte, Gerät A mit einem Seitenspeicher von 1500 Zeichen und Gerät B mit einem Seitenspeicher von 1400 Zeichen, auf ein WML Dokument einer bestimmten Größe (z.B. 1450 Zeichen) zu, so kann das Dokument auf Gerät A dargestellt werden, während auf Gerät B nur eine Fehlermeldung angezeigt wird. Obwohl das Dokument als syntaktisch korrekt bezeichnet werden

muss, kann also nicht im Voraus sichergestellt werden, dass es tatsächlich als solches akzeptiert wird. Die Lösung besteht in diesem Fall darin, das Dokument zu *segmentieren*, d.h. in mehrere Teile zu zerlegen, die dann nacheinander angezeigt werden können. Indem der für WML spezifische ECA nun die maximale Größe eines WML Dokuments vom Gerät erfragen kann und das vom Transformator erfragte Dokument in genügend kleine Teile zerlegen kann, realisiert er aus Sicht der Integrationsumgebung eine Abstraktionsschicht. Der Transformator muss lediglich um die Syntax von WML wissen. Gerätespezifische Merkmale wie die maximale Größe eines WML Dokuments können bei der Transformation hingegen ignoriert werden. Sie werden vom WML ECA behandelt.

Zwei wichtige konzeptionelle Forderungen an den ECA lassen sich aus dem vorigen Absatz entnehmen:

1. Der ECA muss prinzipiell *aktiv* sein können. Er muss also die Merkmale des Endgerätes erfragen können und die vom Transformator stammende Zwischenrepräsentation weiterverarbeiten können.
2. Der ECA muss die Zwischenrepräsentation selbst interpretieren können, um diese gegebenenfalls weiter an die Bedürfnisse eines bestimmten Endgerätes anpassen zu können.

Die erste Forderung wurde bereits in den vorigen Absätzen verdeutlicht. Der ECA muss die Merkmale des Endgerätes erkennen, damit er die mögliche Anpassung bzw. Optimierung der Zielrepräsentation durchführen kann. In diesem Zusammenhang besitzen die so genannten *Geräteprofile* eine große Bedeutung, die im Folgenden noch in Abschnitt 4.3.5 im Detail behandelt werden.

Die zweite Forderung soll wiederum am Fall des „1400 Byte Problem“ des für WML spezifischen Nachrichtenkanals erläutert werden[14]. Bei WML handelt es sich, ebenso wie bei HTML, um eine auf XML basierende deklarative Sprache. Semantische Blöcke werden jeweils von einem bestimmten Schlüsselwort eingeleitet und von einem korrespondierenden zweiten Schlüsselwort abgeschlossen. Soll nun die Aufteilung eines vom Transformator stammenden Dokuments in mehrere Teile erfolgen, so ist es nicht möglich, das Dokument einfach in Segmente der geforderten Größe zu zerlegen, da die resultierenden WML Dokumente syntaktisch falsch wären, sofern sie nur über eines der beiden Schlüsselworte verfügen würden, während das korrespondierende Schlüsselwort in einem nachfolgenden Teildokument stünde. Folglich muss der ECA das Ursprungsdokument analysieren und die Teildokumente jeweils um die abschließenden bzw. öffnenden Tags erweitern, bevor er diese an das Endgerät weiterleitet[15].

Da es natürlich auch möglich sein muss, zwischen den einzelnen Segmenten der Darstellung hin und her zu wechseln, kann die Forderung nach Aktivität des ECAs sogar noch erweitert werden. So muss er in der Lage sein, die vom Transformator übernommene Zielrepräsentation zwischenzuspeichern und Zugriffe vom

[14] Das Problem von WML, Dokumente einer bestimmten Größe nicht anzeigen zu können, wird oft als „1400 Byte Problem“ bezeichnet, da im entsprechenden, vom WAP Forum verfassten Standarddokument[For99f] die minimal von einem WAP Browser zu akzeptierende Dokumentengröße mit 1400 Bytes definiert ist. Viele Geräte erlauben exakt diesen Minimalwert, weswegen sich mögliche Probleme genau ab dieser Dokumentgröße einstellen.

[15] Weitere Implementationsdetails zum segmentierenden ECA sind dem entsprechenden Abschnitt in Kapitel 6 zu entnehmen.

Endgerät zu puffern, um gegebenenfalls diese Form der Navigation innerhalb einer Darstellung der Benutzungsschnittstelle zu ermöglichen. Der ECA muss erkennen, ob eine Bewegung innerhalb der Zustandsrepräsentation erwünscht ist, oder ob ein weiterer Analysezyklus initiiert werden soll, ohne dass der Kern der Integrationsumgebung hiervon Kenntnis erhält.

Andererseits ist eine solche Behandlung der aus der Transformation hervorgegangenen Zielrepräsentation nur als Option anzusehen, die nur für ganz bestimmte Geräteklassen zum Tragen kommen soll. Auch in diesem Zusammenhang ist wiederum die Flexibilität des Konzeptes zu betonen. Externe Kommunikationsadapter müssen hinsichtlich der durch sie unterstützten Geräteklasse in ihrer Funktionalität angepasst und von einem gemeinsamen Grundtyp aus verfeinert werden können. Verfügt beispielsweise der für die Behandlung von HTML zuständige ECA lediglich über die Fähigkeit, HTML Dokumente über einen HTTP Server dem Endgerät bereitzustellen, so stellt der für WML verwendete segmentierende ECA (sog. Segment–ECA) die besagte Verfeinerung vom Grundtyp dar: Er benutzt wie der ECA einen HTTP Server zur Auslieferung der WML Dokumente, nutzt darüber hinaus jedoch Informationen über den Gerätetyp, um die maximale Seitengröße erfassen und die bereits transformierte Zielrepräsentation in der oben skizzierten Weise zu behandeln.

Wie in der Erweiterung des allgemeinen (HTTP–) ECAs zum Segment–ECA sind auch in anderen Bereichen Verfeinerungen denkbar. ECAs können in ihrem Funktionsumfang zu einem gemeinsamen Grundtyp variiert werden und im Realisierungsschritt voneinander abgeleitet werden. Das in sich abgeschlossene Konzept des Externen Kommunikationsadapters begünstigt ein solches Vorgehen. Da die Interaktion mit dem Kern der Integrationsumgebung ausschließlich über ein definiertes Protokoll erfolgt, kann für die Realisierung des ECAs die Programmiersprache bzw. das Programmiermodell verwendet werden, das die Wiederverwendung von Funktionalität und die Unterstützung der Geräteklasse optimal unterstützt. Auf diese Weise können dann Vererbungskonzepte zwischen den einzelnen ECAs greifen.

Am Beispiel des für HTML zuständigen ECAs wird ein weiteres Merkmal deutlich, das bei der Konzeption des Externen Kommunikationsadapters betrachtet werden musste, nämlich dessen Fähigkeit zur Realisierung eines *Sitzungskontextes*. Während auf den Sitzungsbegriff in Abschnitt 4.3.8 noch näher eingegangen wird, an dieser Stelle zunächst soviel: Jede Interaktion eines Endgerätes mit der Integrationsumgebung bzw. indirekt durch sie mit der Benutzungsschnittstelle der Anwendung findet grundsätzlich im Rahmen einer Sitzung statt. Diese bildet einen Kontext, in dem Endgerät, ECA, Transformator und Shadow miteinander in Beziehung gestellt werden und für die Dauer eines Anwendungslaufs eine logische Einheit bilden. Bezogen auf den ECA ergibt sich nun ein Problem darin, dass es zwar einerseits Zugangskanäle gibt, die die Zuordnung zu einem bestimmten Sitzungskontext sehr einfach machen, es andererseits jedoch auch solche Kanäle gibt, in denen das Konzept einer Sitzung zunächst nicht bekannt ist und nur schwer nachträglich integriert werden kann.

Als Beispiel für die erste Kategorie mag man den Kanal für gesprochene Sprache heranziehen. Durch das Telefonnetz und die Art, wie durch Anwahl einer bestimmten Telefonnummer der Zugang hierzu realisiert ist, wird grundsätzlich ein wohldefinierter Kommunikationskontext realisiert. Stellt man sich als ECA einen

digitalen Anrufbeantworter vor, der gesprochene Sprache in Textnachrichten umsetzen kann und diese an die Integrationsumgebung weitergibt, so ist für den ECA jederzeit die Zuordnung eines „Endgeräts“(d.h. eines Anrufers) zu einem Kanal möglich; die Verquickung der Kommunikation mit mehreren Endgeräten über einen Kanal ist nicht möglich. Auf der anderen Seite kann als Beispiel für letztere Kategorie das HTTP Protokoll angeführt werden. HTTP definiert keinen Sitzungskontext, sondern behandelt jeden Zugriff völlig unabhängig von vorangegangenen Zugriffen. Wird der ECA in diesem Fall durch einen HTTP Server realisiert, so ist zunächst einmal nicht klar, welches Endgerät (bzw. welcher Browser) einer bestimmten Sitzung zuzuordnen ist. Entsprechend können beim HTTP Server eingehende Befehle nicht ohne weiteres einem bestimmten Application Shadow zugeordnet werden oder vom Transformator stammende Zustandsabbilder in der Zielrepräsentation nicht an ein bestimmtes Endgerät weitergeleitet werden. Die Realisierung eines Sitzungskontextes, d.h. eines *technischen* Sitzungskontextes (siehe hierzu Abschnitt 4.3.8) stellt somit eine sehr wichtige Aufgabe des ECAs dar. Erst durch den technischen Sitzungskontext wird es möglich, dass ein ECA gleichzeitig für mehrere Endgeräte zuständig sein kann, wie dieses für den HTTP–ECA der Fall ist.

Der Realisierungsweg, der dabei eingeschlagen wird, hängt stark von den Eigenschaften des Mediums ab, für das der Sitzungskontext geschaffen werden soll. Für das HTTP–Protokoll könnte eine Gerätesitzung unter Verwendung eindeutiger Kennungen (UIDs, Cookies) etabliert werden, die zwischen Endgerät und ECA migrieren, um die Sitzung zu kennzeichnen, für andere Medien jedoch auch gänzlich hiervon unterschiedliche Methoden zum Einsatz kommen. Zu beachten ist an diesem Punkt der Arbeit vor allem, dass der ECA auch in diesem Zusammenhang als Abstraktionsschicht Verwendung findet: Wie der Geräte–Sitzungskontext realisiert wird, ist alleiniges Realisierungsdetail des ECAs, während auf der Basis der eindeutigen Identifikation einer bestimmten Gerätesitzung dann eine Anwendungssitzung etabliert werden kann, welche die eigentliche Verbindung zwischen Endgerät und Shadow darstellt. Nur letzteres Konzept wird mit den übrigen Teilen der Integrationsumgebung geteilt. Ausführlicher wird das Sitzungskonzept in Abschnitt 4.3.8 behandelt.

In der Behandlung des ECAs wurde bislang der Weg von Informationen aus der Integrationsumgebung heraus in Richtung des Endgerätes betrachtet. Auf diesem Weg wird die zuvor transformierte Zwischenrepräsentation über den auf die Geräteklasse abgestimmten Nachrichtenkanal an das Endgerät übertragen und dort in der vom Gerät angebotenen Weise dargestellt. Auch hinsichtlich der Aufnahme und Weiterleitung von Befehlen, die vom Endgerät ausgehen, übernimmt der ECA eine Abstraktionsfunktion. Abhängig von den Geräteeigenschaften werden Befehle entgegengenommen und diese dann in der für die Integrationsumgebung wohldefinierten Form an die Kernkomponenten und letztlich an den Application Shadow bzw. die Anwendung weitergeleitet.

Aus der Sicht der Integrationsumgebung ist es zunächst nur wichtig zu erfahren, welches GUI–Element bedient wurde und ggf. in welcher Weise dieses geschehen ist. Der ECA nimmt die vom Endgerät eingehenden Informationen hierzu entgegen und ordnet sie einer Geräte–Sitzung zu. Er extrahiert die Befehle aus dem Trägerprotokoll (z.B. HTTP) und setzt sie dann in ein generisches Format auf XML–Basis um, das dann über den identifizierten Kanal an den für die zu bedienende Anwendung verantwortlichen Application Shadow weitergeleitet

wird. Dabei werden gerätespezifische Aspekte vom ECA gekapselt, während die beim Shadow eingehenden Befehle lediglich noch das GUI–Element sowie Informationen über die Art der Zustandsänderung enthalten.

4.3.5 Gerätespezifische Adaption durch Geräteprofile

Nachdem in den vorigen Abschnitten die Einführung einer Abstraktionsschicht für Anwendungen auf der einen Seite und Geräte auf der anderen Seite beschrieben und ein Vergleich zwischen Geräteabstraktion und Anwendungsabstraktion durchgeführt wurde, soll nunmehr auf einen weiteren wesentlichen Baustein für ein universelles Integrationskonzept eingegangen werden, der sich in direkter Weise aus den zuvor beschriebenen Bausteinen ergibt: Die Definition von Geräteprofilen, deren Verallgemeinerung zu Geräteklassen und schließlich die Nutzung dieser beiden Konzepte zur Realisierung optimaler Strategien für die Unterstützung mobiler Geräte in einer Integrationsumgebung.

Da über die zuvor beschriebenen generischen Geräteadapter nahezu beliebige Geräte eingebunden werden können, stellt sich die Frage, wie die interne Unterstützung hierfür bewerkstelligt wird. Der vorige Abschnitt gab hierzu keine erschöpfende Antwort, da auf der Ebene des generischen Geräteadapters die Unterstützung explizit realisiert wird: Jeder Adapter ist genau einer Klasse von Geräten zugeordnet. Die Zuordnung zwischen Gerät und Adapter ergibt sich aus technischen Gegebenheiten: Ein WAP Telefon kann nur über den WAP Adapter eingebunden werden, ein Webbrowser nur über den HTML Adapter et cetera. Auf dieser Ebene ist die Unterstützung also bereits durch die Wahl des passenden Zugangskanals gegeben.

Anders stellt sich die Situation für den Kern der Integrationsumgebung dar. Hier muss die grobe Einordnung, die über den Geräteadapter vorgenommen wird, verfeinert werden und in eine optimale Unterstützungsstrategie münden. Der Kern der Integrationsumgebung muss für verschiedene Gerätearten nutzbar sein, wobei die Anpassung nicht explizit erfolgen kann, ohne dass ein flexibler Einsatz der Umgebung in Gefahr gerät. Es muss möglich sein, Klassen von Geräten zu bilden und möglichst automatisch zu identifizieren und dann entsprechend zu handeln. Für einen PDA mit WAP Browser muss anders gehandelt werden als für ein WAP Telefon, für einen PDA mit Webbrowser anders als für einen PC. Obwohl jeweils in beiden Fällen derselbe Mechanismus zur Übergabe und Darstellung von Informationen verwendet wird (hier WAP, dort HTML), muss auf der Basis der speziellen Merkmale des jeweiligen Geräts eine Anpassung hinsichtlich der Aufbereitung der Information vorgenommen werden. Eigenschaften, Geräteprofile sowie Geräteklassen, die den Kern dieses Abschnitts darstellen, sind der wesentliche Mechanismus hierfür.

Das in diesem Abschnitt behandelte Thema der Geräteprofile und Geräteklassen kann im Gesamtzusammenhang dieser Arbeit als äußerst relevant angesehen werden. Als wesentliches Ziel wurde die optimale Unterstützung von möglichst beliebigen mobilen Geräten in ihrem Zugriff auf in verteilten Systemen gespeicherten Informationen formuliert. Es ist klar, dass einem solchen Anspruch nur gerecht zu werden ist, wenn die Anforderungen des jeweiligen Geräts genau spezifiziert werden und in einer entsprechenden Strategie Verwendung finden können. Soll von einem Gerät zugegriffen werden, so muss es möglich sein, die

passende Strategie zu identifizieren und gemäß ihres Inhalts zu handeln. Sowohl die Spezifikation als auch die Auswahl von Strategien ist nur dann möglich, wenn die Eigenschaften von Geräten identifiziert werden können. Genau dieses ist das Ziel von Geräteklassen. Insofern behandelt der Abschnitt ein für die zu entwerfende Integrationsinfrastruktur elementares Konzept.

Dabei wird zunächst allgemein dargestellt, was sich hinter Eigenschaften und Geräteprofilen verbirgt, wie diese aufgebaut sind, wie sie verwaltet und wie sie verwendet werden. Während das Geräteprofil als direktes Abbild eines konkreten Gerätes verstanden werden kann, handelt es sich bei der Geräteklasse um dessen abstrakte Vorlage. Der folgende Abschnitt behandelt eingehend dieses Konzept und seine Rolle im Rahmen der Geräteintegration. Schließlich wird diskutiert, warum und vor allem wie Geräteklassen miteinander in Beziehung gesetzt werden können und wie dieser Übergang vom Profil zur Klasse die Auswahl einer optimalen Strategie ermöglicht. Insgesamt soll also zunächst die theoretische Grundlage beschrieben werden, auf der eine Geräteklassenverwaltung basieren kann, wie sie in der in dieser Arbeit entworfenen Integrationsinfrastruktur nutzbar ist.

Im Anschluss werden dann konkrete Realisierungstechniken erläutert, die zur Kodierung von Profilen und Klassen verwendet werden können und anhand derer diese sich zueinander in Beziehung stellen lassen. Es wird das Resource Description Framework (RDF) [LS99, BG99] skizziert und das auf RDF aufbauende Composite Capability/Preference Profile (CC/PP) [RHDS99, NHO00] erklärt, das zur Kodierung von Geräteklassen geeignet ist. Bei CC/PP handelt es sich um einen vergleichsweise generischen Ansatz, der zunächst keine besondere Spezialisierung auf einen bestimmten Einsatzzweck vorsieht. Da im Rahmen dieser Arbeit die Unterstützung mobiler Geräte im Vordergrund steht, wird im Anschluss UAProf [Gro99] beschrieben, eine Erweiterung von CC/PP, die speziell für den Einsatz im Zusammenhang mit WAP entwickelt wurde. Zum Abschluss des Abschnitts werden existierende Arbeiten in diesem Bereich skizziert und diese in Relation zu den in diesem Abschnitt beschriebenen Konzepten gesetzt.

4.3.5.1 Merkmale und Profile

Jedes Gerät, das in einer Integrationsumgebung berücksichtigt werden soll, besitzt eine Reihe von Merkmalen, die es eindeutig kennzeichnen. Diese Merkmale können sich einerseits auf eher äußerliche Aspekte beziehen (Farbe, Gewicht, Größe usw.). Merkmale können des Weiteren interne Aspekte des Geräts beschreiben. Hierbei weist zunächst die Hardware selbst eine Reihe von Merkmalen auf, die das Gerät kennzeichnen und von anderen Geräten unterscheiden. Zu dieser Gruppe zählen die verwendete CPU, der vorhandene Arbeitsspeicher, die Auflösung und Farbtiefe des Bildschirms oder das Vorhandensein einer Tastatur. Auch können Merkmale der Systemsoftware erfasst sein wie z.B. das verwendete Betriebssystem, die Betriebssystemversion und die vorhandenen Betriebssystemerweiterungen. Auch Anwendungen können als kennzeichnende Merkmale des Geräts aufgefasst werden. Die Version des installierten WAP Browsers, die installierten WMLScript Bibliotheken, die vom Browser verstandenen Videoformate wären Beispiele für derartige Merkmale. Schließlich lassen sich auch Einstellungen durch den Anwender als Merkmale auffassen, da sich das Gerät auch durch

sie zu einem bestimmten Zeitpunkt von anderen Geräten unterscheidet. Solche Einstellungen könnten die gewählte Textfarbe betreffen, die Dringlichkeitsstufe, auf der eingehende Nachrichten erlaubt sind o.ä.

Die Erfassung und Übermittlung von Merkmalen zwischen Gerät und Unterstützungssystem ist von großer Wichtigkeit für das Erbringen eines optimalen Unterstützungsdienstes. Nur wenn die Eigenschaften des Gerätes und die Präferenzen des Benutzers bekannt sind, kann die notwendige Adaption in einer Weise durchgeführt werden, wie sie für eine effiziente Nutzung des Gerätes zu fordern ist. In der Literatur wird in diesem Zusammenhang oft von *Content Negotiation* gesprochen (vgl. [RHDS99]), der Aushandlung von Dienstgüte bzw. Dienstinhalt. Im Gegensatz zu herkömmlichen verteilten Systemen muss die Anpassung nach Möglichkeit außerhalb des Gerätes stattfinden, da eine Anpassung auf dem Gerät oft aufgrund begrenzter Ressourcen ausgeschlossen oder durch die erschwerte Handhabung nicht praktikabel ist.

Es ist offensichtlich, dass sich nicht alle Merkmale eines Gerätes gleichermaßen zur Erfassung und Verwendung in einem Unterstützungssystem eigenen und nicht alle Merkmale, die sich zur Erfassung eignen, würden auch tatsächlich sinnvoll Verwendung finden können. Die Relevanz hängt stark von der Art des zu erbringenden Dienstes durch das Unterstützungssystem ab.

Zunächst einmal haben äußere Merkmale wie Gewicht und Farbe des Gerätes wahrscheinlich keinen Einfluss auf die Dienstgüte des Unterstützungssystems. Obwohl sie erfasst und im Gerät abgelegt sein könnten, scheint ihre Verwendung wenig sinnvoll. Auch mag der installierte RAM Speicher unerheblich sein, während andererseits der vorhandene freie Arbeitsspeicher sehr wohl relevant sein kann. Die oben genannte Version des WAP Browsers ist schließlich im Zusammenhang mit der Aufbereitung und Übertragung von WML Seiten wichtig, nicht jedoch bei der Übertragung eines Videostroms. Relevanz ergibt sich also a) aus der Verfügbarkeit der Eigenschaft bzw. aus deren Auslesbarkeit durch das System und b) aus der Nutzbarkeit im Sinne der Anwendung. Merkmale können also in Klassen unterteilt werden, wobei sich die folgenden Klassen zunächst anbieten:

Hardware : Merkmale, welche die Hardware des Geräts beschreiben

Netzwerk : Merkmale, welche die Kommunikationsumgebung des Gerätes beschreiben

Software : Merkmale, welche die Systemsoftware des Gerätes beschreiben

Anwendung : Merkmale, welche die Anwendungsumgebung des Geräts beschreiben

Merkmale können unveränderlich sein (Hersteller, CPU, Farbe) oder variabel (verfügbarer RAM Speicher, geladene WMLScript Bibliotheken). Sie können für eine ganze Klasse von Geräten gleich sein (CPU im PalmPilot III) oder zwischen den einzelnen Geräten variieren (Benutzerkennung im PalmPilot III). Merkmale können unverändert sein (Werkseinstellung bzgl. Textfarbe) oder durch Benutzerpräferenzen überlagert werden (Einstellung der Textfarbe durch den Benutzer). Schließlich können Merkmale von Hardware zu Hardware und von Anwendung zu Anwendung verschieden sein oder gänzlich fehlen (Merkmal „Textfarbe“

existiert nicht bei einer Smartcard, Merkmal „Geschwindigkeit“ hat unterschiedliche Bedeutung für eine Netzwerkkarte im Vergleich zu einem Video–Codec). Abbilung 4.23 zeigt, wie Merkmale als Teil einer Geräteklasse zum Einsatz kommen können.

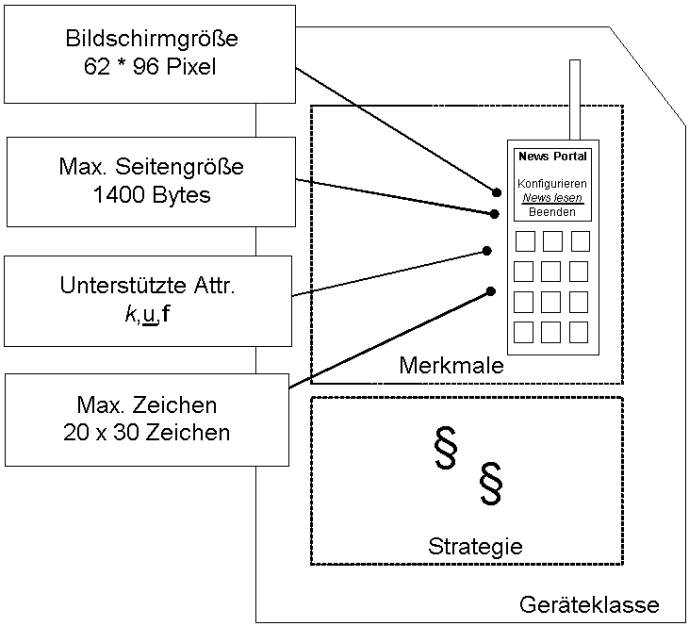


Abbildung 4.23: Merkmale und Strategien bilden den Kern einer Geräteklasse

Zunächst einmal kann also festgestellt werden, dass Merkmale dazu dienen, die Dienstleistung eines Unterstützungssystems den Anforderungen eines Gerätes anzupassen, dass es verschiedene Ebenen gibt, auf denen Merkmale eines Gerätes erfasst werden können, dass diese zum Teil unerheblich sind, zum Teil aber als essentiell für das Erbringen optimierter Dienstleistungen angesehen werden können und dass die diesbezügliche Bewertung vornehmlich vom Kontext abhängt, in dem das Gerät verwendet wird.

Gerätemerkmale können auf verschiedene Arten gespeichert und übertragen werden. Zunächst ist in jedem Falle ein Vorgehen möglich, bei dem Sender und Empfänger sich auf ein gemeinsames Austauschformat einigen, in welchem die Merkmale kodiert und dann vom Sender an den Empfänger übergeben werden. Die Merkmale werden in diesem speziellen Format im Gerät abgelegt und bei Bedarf an die Gegenstelle übermittelt, wo sie ausgelesen und weiterverarbeitet werden können.

Die Nutzung eines proprietären Formats bietet zwar den Vorteil der maximalen Speicherungs- und Übertragungseffizienz, ist aber nur sinnvoll in Szenarios, in denen sich alle beteiligten Partner auf das Austauschformat einigen können und es allen bekannt ist. Gerade im Bereich mobiler Massengeräte wie Mobiltelefone, PDAs usw. mit einer großen Menge an Herstellern und einer ständig wachsenden Anzahl von Gerätetypen ist ein solcher Ansatz nicht praktikabel. Hier muss vielmehr ein standardisiertes Format zur Erfassung und zum Austausch von Gerätemerkmalen geschaffen werden. Auch sollte ein solches For-

mat nicht auf einen bestimmten Anwendungsbereich, z.B. die Beschreibung von Merkmalen mobiler Geräte, eingeschränkt sein, sondern vielmehr universell zur Beschreibung beliebiger Dinge nutzbar sein. Nur so lässt sich ein entstehendes Beschreibungssystem ohne logische Brüche in einen größeren Zusammenhang übernehmen.

4.3.5.2 Geräteprofile als Abbild konkreter Eigenschaften

Die konzeptionelle Grundlage einer solchen vereinheitlichten Beschreibung wird durch die Geräteprofile dargestellt. In Geräteprofilen werden die zuvor nur allgemein zum Gerät zugehörigen Eigenschaften bewertet und organisiert. Es werden die im Anwendungskontext nicht relevanten Eigenschaften ausgeblendet und die relevanten Eigenschaften zu einer logischen Einheit zusammengefasst. Dabei bezieht sich ein Geräteprofil immer auf ein bestimmtes, eindeutig identifizierbares Gerät aus der realen Welt. Es sind genau die Eigenschaften dieses einen Gerätes (d.h. dieser *Instanz*), die erfasst und in ein geeignetes Format übertragen werden.

Definition Geräteprofil: Ein Geräteprofil stellt die logische Gruppierung von Merkmalen bzw. Eigenschaften eines Gerätes dar. Das Geräteprofil umfasst all diejenigen Merkmale, die im Kontext einer Anwendung von Bedeutung sind. Dieses können sowohl unveränderliche Merkmale der Hardware des Gerätes sein als auch Merkmale der Software, die auf dem Gerät zur Anwendung kommt. Merkmale werden in ihren Ausprägungen (Wertbelegungen) im Geräteprofil zusammengefasst, wobei für jedes Merkmal ein definierter Wertebereich existiert.

Das Geräteprofil stellt also die abstrakte Entsprechung zum konkreten Gerät innerhalb eines Anwendungskontextes dar. Dabei lässt sich eine gute Übereinstimmung zum Konzept des Objektes aus der Objektorientierung sehen. Objekte gelten als Entsprechung zu Gegenständen aus der realen Welt. Sie beinhalten deren Merkmale und ihren Zustand und fassen darüber hinaus auch die Handlungen zusammen, mit denen die Objekte beeinflusst werden können. Es sind genau die Merkmale und der Zustand, der sich aus ihnen ergibt, die in Geräteprofilen kodiert sind.

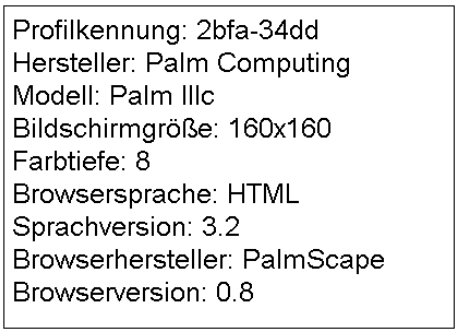


Abbildung 4.24: Beispiel für ein einfaches Geräteprofil

Die Abbildung 4.24 zeigt das Beispiel eines einfachen Geräteprofils. In diesem sind nur wenige Eigenschaften zusammengefasst, wie sie für einen bestimmten Anwendungskontext benötigt werden, nämlich die Anpassung von Inhalten an

einen auf dem Gerät verwendeten Browser. Da es in diesem Kontext notwendig ist, die Ausgabe an die Größe und Fähigkeiten des Bildschirms, den verwendeten Dialekt des Browsers und dessen Fähigkeiten anzupassen, sind genau die hierfür notwendigen Merkmale im Profil zusammengefasst. Erkennbar ist auch, dass das Profil genau einem bestimmten Gerät zuzuordnen ist. Hierzu ist eine Profilkennung angegeben, die dieses Profil eindeutig kennzeichnet und die idealerweise einer Kennung entspricht, die das Gerät eindeutig kennzeichnet. Bei der Profilkennung könnte es sich beispielsweise um die Seriennummer des Gerätes handeln.

Im Geräteprofil werden nicht alle möglichen Merkmale des Gerätes zusammengefasst, sondern nur diejenigen, die im Kontext eines Anwendungssystems relevant sind. So existieren für den im Beispiel beschriebenen PalmPilot eine Reihe weiterer Merkmale (RAM, OS Version, Sprache, Merkmale anderer Anwendungsprogramme usw.), die im gezeigten Profil nicht erfasst sind. Während das vorgestellte Profil überaus restringiert ist und in der tatsächlichen Anwendung eine Einschränkung sicherlich nicht auf einem derartig kleinen Anwendungsbereich stattfinden würde, so steht doch fest, dass eine solche Einschränkung immer existieren wird. Dieses ist vornehmlich aus Gründen der Effizienz der Fall, da Geräteprofile in einer entsprechend kodierten Form zwischen den beteiligten Dienstnehmern und –erbringern ausgetauscht werden müssen. Daneben bedeutet die Erfassung und Verwaltung von Profilinformationen einen mit deren Größe zunehmenden Verwaltungsaufwand, der insbesondere auf dem mobilen Gerät zu einer inakzeptablen Nutzung von Ressourcen führen und letztlich die Nutzbarkeit des gesamten Gerätes einschränken könnte.

Gerade in der konkreten Umsetzung in Ansätzen wie RDF, CC/PP et cetera werden in Geräteprofilen nicht ausschließlich die Merkmale des Gerätes selbst gespeichert, sondern darüber hinaus auch die Präferenzen, die der Anwender des Gerätes bezüglich bestimmter Merkmale hat, bzw. die das Gerät im Zusammenspiel mit entfernten Diensten aufweisen soll. Während Gerätemerkmale vornehmlich indirekt zu einer Anpassung des entfernten Dienstes führen, sollen die Präferenzen den Dienst direkt beeinflussen. Wird bei der Übergabe der Bildschirmgröße als Merkmal also nur indirekt formuliert, dass die an das Gerät zu liefernden Informationen an die Höhe und Breite des Bildschirmes anzupassen ist, so legt die geforderte Bildschirmfarbe als Präferenz explizit fest, wie die Aufbereitung stattzufinden hat.

Insgesamt kann die Definition des Geräteprofils nunmehr also wie folgt erweitert werden:

Definition erweitertes Geräteprofil: Ein erweitertes Geräteprofil stellt die logische Gruppierung von Merkmalen bzw. Eigenschaften dar, welche die Verwendung eines Gerätes in einem Anwendungskontext betreffen. Das Geräteprofil umfasst all diejenigen Merkmale, die im Kontext einer Anwendung von Bedeutung sind. Dieses können sowohl unveränderliche Merkmale der Hardware des Gerätes sein als auch Merkmale der Software, die auf dem Gerät zur Anwendung kommt. Diejenigen Merkmale, die nicht das Gerät selber beschreiben, sondern die Anforderungen an den Dienst enthalten, der in Anspruch genommen werden soll, werden als Präferenzen bezeichnet. Merkmale und Präferenzen werden in ihren Ausprägungen (Wertbelegungen) im Geräteprofil zusammengefasst, wobei für jedes Merkmal ein definierter Wertebereich existiert.

Im Gegensatz zum einfachen Geräteprofil ist für das erweiterte Geräteprofil zu beachten, dass hier die Wertigkeit des Anwendungskontextes stark erhöht ist. Die Formulierung von Präferenzen macht nur dann Sinn, wenn sie hinsichtlich der Nutzung von Diensten geschieht. Dieses können allgemein sowohl Dienste sein, die auf dem Gerät selbst ablaufen, als auch externe Dienste, auf die über Kommunikationskanäle zugegriffen wird. Insbesondere in der Kombination mit Eigenschaften des Gerätes selbst ist jedoch klar, dass es genau die Nutzung entfernter Dienste ist, für die erweiterte Geräteprofile Verwendung finden.

Im Sinne einer konsistenten Begrifflichkeit soll im weiteren Verlauf der Begriff des Geräteprofils synonym sowohl für das einfache Geräteprofil als auch für das erweiterte Geräteprofil verwendet werden.

4.3.5.3 Geräteklassen als Basis optimierter Diensterbringung

Nachdem nun geklärt ist, wie die Merkmale von Geräten in Form von Eigenschaften zu Geräteprofilen zusammengefasst werden können, soll nunmehr darauf eingegangen werden, wie die so gebündelten Informationen über kommunizierende Geräte dazu eingesetzt werden können, eine optimale Dienstanpassung bei der Interaktion mit ihnen zu gewährleisten. Es geht dabei vornehmlich um die Verallgemeinerung der konkreten Form von Geräteinformationen, dem Geräteprofil, in dessen abstraktes Pendant, die Geräteklasse. Wie oben angedeutet bildet die Erhebung und Verarbeitung von Wissen über die charakteristischen Merkmale von Geräten die Grundlage zur Umsetzung von Informationen in eine Repräsentation, wie sie vom anfragenden Gerät optimal verstanden und dargestellt werden kann. Im Zuge des Verbindungsaufbaus werden Eigenschaften an den Diensterbringer übermittelt, und es wird dann von diesem entschieden, wie er diese am besten berücksichtigen und die Ausgabe entsprechend anpassen kann.

Eine solche Anpassung kann grundsätzlich auf zwei verschiedene Arten erfolgen:

Zunächst kann der Diensterbringer jede der Eigenschaften isoliert betrachten und den Inhalt gemäß ihrer Ausprägung anpassen. Er geht die übermittelten Eigenschaften Schritt für Schritt durch und erhält schließlich als Resultat die gewünschte (bzw. geforderte) Repräsentation. Dieses Verfahren ist aufwendig und kann zu unbefriedigenden Ergebnissen führen, sofern ein Endgerät nicht alle vorhandenen Eigenschaften überträgt.

Eine andere Möglichkeit besteht darin, die wiederkehrenden und gleich bleibenden Eigenschaften einer ganzen Gruppe von Geräten zusammenzufassen und stellvertretend hierfür lediglich eine Referenz auf diese Information an den Diensterbringer zu senden. Kann der Diensterbringer den Verweis auf die referenzierte Gruppe von Eigenschaften zuordnen, etwa durch Nachfrage bei einem entsprechenden Katalog, so ist es ihm möglich, alle Eigenschaften der Gruppe zu berücksichtigen, ohne dass es hierbei zu Abweichungen durch nicht übermittelte Eigenschaften kommen könnte. In Abwandlung dieses Vorgehens kann das Endgerät auch eine Referenz übermitteln und ein Teilprofil, das lediglich die Abweichungen des Gerätes von den referenzierten Eigenschaften enthält. Der Diensterbringer kann dann eine Vorverarbeitung anhand der Referenzwerte durchführen und dann eine weitere Anpassung anhand des Teilprofils. Die in

einem logischen Block zusammengefassten Defaultwerte werden als Klasse bzw. Geräteklasse bezeichnet.

Definition Geräteklasse: Eine Geräteklasse umfasst all diejenigen Eigenschaften und deren Wertbelegungen, die für alle Instanzen der Klasse unverändert gelten bzw. die angenommen werden sollen, wenn für die entsprechende Instanz keine expliziten Angaben über die Wertbelegung einer jeweiligen Eigenschaft gemacht wird.

Die Verwendung von Geräteklassen bringt Vorteile mit sich. Wesentliche Aspekte wurden dabei im Verlauf des Abschnitts bereits angedeutet: Zunächst führt der Rückgriff auf Geräteklassen zu einer Effizienzsteigerung hinsichtlich der zu übertragenden Datenmengen zwischen Endgerät, zwischengeschalteten Stellvertretern und Diensterbringern. Im Zusammenhang mit UAProf Profilen (siehe unten) konnte gezeigt werden, dass die Nutzung von Referenzen anstelle von ganzen CPIs das Endgerät von der Kodierung und Übertragung von Geräteprofilen entlasten und diese in das Festnetz verlagern kann, wo das Gateway und weitere Kommunikationsknoten die entsprechenden Informationen aus zentralen Profilverzeichnissen beziehen können.

Ein weiterer Vorteil ergibt sich bei der Einbindung von älteren Geräten, die nicht über die Möglichkeit zur Verarbeitung von Profilen verfügen, wohl aber über die Möglichkeit zur Übermittlung einer eindeutigen Kennung. Dieses ist etwa bei WAP Geräten der Fall, wo die Gerätekennung sowie die Kennung des verwendeten WAP Browsers bei der Kontaktaufnahme an den entfernten WAP Server übertragen wird. Werden diese Kennnungen als Referenzen verstanden, so kann mit ihrer Hilfe der Zugriff auf ein vorgefertigtes Profil erfolgen, das dann weiterverwendet wird. Schließlich können Geräteklassen verwendet werden, um unvollständige Angaben über die Fähigkeiten eines Gerätes auszugleichen, in dem die entsprechende Eigenschaft aus der zugrunde liegenden Geräteklasse übernommen wird.

Eine derartige direkte Nutzung von Klasseninformationen lässt sich recht schnell bewerkstelligen, sofern eine Verbindung zwischen der konkreten Geräteinstanz und der zugrunde liegenden Klasse hergestellt werden kann. Dieses kann anhand von eindeutigen Kennungen geschehen, die als Teil der Geräteklasse abgelegt werden. In einem Diensterbringer oder einem Profilverzeichnis wird dann eine entsprechende Tabelle angelegt, die nach eingehenden Kennungen durchsucht werden kann.

Eine weitere Möglichkeit besteht in der Analyse der vom Endgerät übertragenen Eigenschaften und deren Wertbelegungen. Hierbei wird der Versuch unternommen, die aus dem Geräteprofil stammenden Eigenschaften mit den bekannten Geräteklassen zu vergleichen und anhand der Übereinstimmung zwischen hier wie dort vorhandenen Eigenschaften auf die Klasse zu schließen, die dem Geräteprofil zugrunde liegt. Ein solches Vorgehen ist immer dann notwendig, wenn die zu berücksichtigenden Geräte keine eindeutige Kennung übermitteln, weil sie keine Unterstützung von Geräteprofilen besitzen, oder wenn diese zwar in der Lage sind, Eigenschaften in Profilen zu übermitteln, sie dabei jedoch keinen Verweis auf die zugrunde liegende Geräteklasse übertragen.

Es ist offensichtlich, dass es bei diesem Vorgehen zu Fehlern kommen kann. So ist denkbar, dass verschiedene Geräte über die gleiche Farbtiefe, die gleiche

CPU und denselben Hersteller verfügen und sie diese Eigenschaften auch kodiert an den Empfänger übermitteln. Variieren diese Geräte nun stark in ihrer Bildschirmauflösung, die nicht als kodierte Eigenschaft übermittelt wird, so kann es abhängig von der zuerst als passend identifizierten Geräteklasse sein, dass die resultierende Datenrepräsentation vom Endgerät nicht dargestellt werden kann.

Eine mögliche Herangehensweise stellt in diesem Fall der Aufbau von Beziehungen zwischen Geräteklassen dar, die dann die Verwandtschaft von Geräteklassen untereinander repräsentieren und das treffsichere Schließen von Eigenschaften auf die zugrunde liegende Klasse vereinfachen. Solche Beziehungen können manuell eingerichtet oder dynamisch durch Analyse der Eigenschaften bekannter Klassen hergeleitet sein. Dann kann durch Navigation durch das Geflecht bekannter Knoten die Eingrenzung möglicherweise passender Klassen vorgenommen werden, bevor dann eine bestimmte Geräteklasse als passend angenommen wird.

Bevor auf die konzeptionelle Seite näher eingegangen wird, soll das Gesagte anhand eines stark vereinfachten Beispiels illustriert werden: Seien einem Profilverzeichnis Geräteklassen mit den folgenden Eigenschaften bekannt:

- Hersteller
- Modell
- Bildschirmgröße
- Farbtiefe
- Browsersprache
- Sprachversion
- Browserhersteller
- Browserversion



Abbildung 4.25: Beispiele für verschiedene Geräteklassen

Auf der Basis dieses Vokabulars seien die in Abbildung 4.25 dargestellten Geräteklassen definiert und diese dann in der illustrierten Weise zueinander in Beziehung gesetzt. Diese Klassen können nun in einer einfachen Hierarchie zueinander angeordnet werden. Dabei werden gemeinsame Merkmale zusammengefasst und als Oberklasse definiert, während die Merkmale, in denen sich die Klassen unterscheiden, zur Definition von Unterklassen führen. Für die angeführten Klassen ließe sich eine Hierarchie wie in Abbildung 4.26 ableiten.

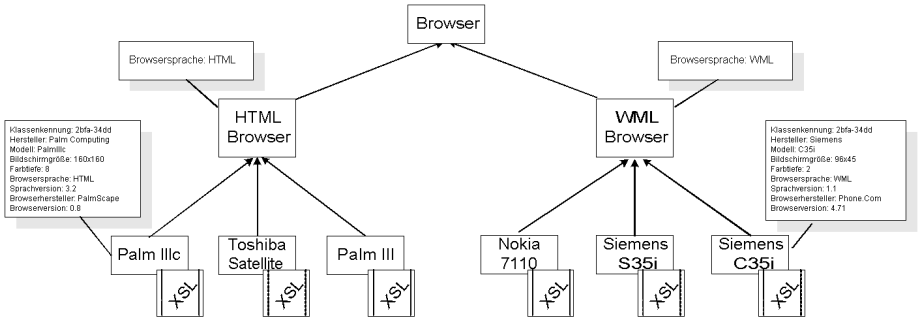


Abbildung 4.26: Eine einfache Gerätehierarchie

In dieser einfachen Gerätehierarchie wird zunächst davon ausgegangen, dass alle betrachteten Geräte als Browser bezeichnet werden können. Das Merkmal „Browser“ wird entsprechend als Wurzelknoten definiert. Die in Abbildung 4.25 aufgeführten Geräte lassen sich nach der Art des Browsers unterscheiden. Einige von ihnen sind HTML–Browser (d.h. Webbrowser), die übrigen sind WML–Browser (d.h. WAP Browser). Der jeweiligen Gruppe lassen sich schließlich die Geräte selbst in ihren Klassen zuordnen. In diesem Beispiel ist auf der Ebene der Blätterknoten jeweils ein XSL Stylesheet symbolisiert. Dieses soll stellvertretend für die Strategie stehen, die einer jeden Klasse beigeordnet ist: Kann durch Navigation im Baum eine bestimmte Geräteklasse identifiziert werden, so kann das mit der Klasse zusammenhängende Regelwerk ermittelt werden und anhand dieses Regelwerkes sodann die Anpassung von Informationen an das Gerät durchgeführt werden.

Zur Verdeutlichung sei nochmals betont, dass es sich bei den im Baum abgelegten Informationen um Klasseninformationen und nicht um Profilinformationen handelt. Das Profil stellt eine konkrete Instanz der Klasse dar, die durch eventuell von der Klasse abweichende Benutzerpräferenzen überlagert oder in der Angabe der Eigenschaften unvollständig sein könnte. Das Profil wird verwendet, um bei der Navigation durch die Klassenhierarchie Vergleiche mit den jeweiligen Knoten anzustellen und um so die passende Klasse zu ermitteln.

Soll nun ein Gerät bedient werden, das sein Geräteprofil als Teil der Anfrage übergeben hat, so kann der Baum anhand der in dem Profil gespeicherten Werte nach der passenden Geräteklasse durchsucht werden. Hierzu wird ausgehend vom Wurzelknoten für dessen Unterklassen geprüft, ob für alle sowohl im Geräteprofil als auch in der Geräteklasse vorkommenden Eigenschaften eine Übereinstimmung vorliegt. Ist dieses der Fall, so wird mit derjenigen Unterklasse des Wurzelknotens fortgefahren, für die dieses der Fall war. Dabei wird dann genauso verfahren wie zuvor, bis schließlich keine Navigation in eine Unterklasse

mehr möglich ist, was im optimalen Fall genau bei Erreichen eines Blattknotens geschieht. Sind beispielsweise die Eigenschaften

- Browsersprache: WML
- Browserhersteller: Phone.Com
- Browserversion: 4.71

im Geräteprofil abgelegt, so wird im Zuge der Navigation durch die Klassenhierarchie schließlich die Geräteklasse „C35i“ aufgefunden werden und das hiermit verbundene Stylesheet zur Aufbereitung der Information eingesetzt werden. Siehe hierzu auch Abbildung 4.27.

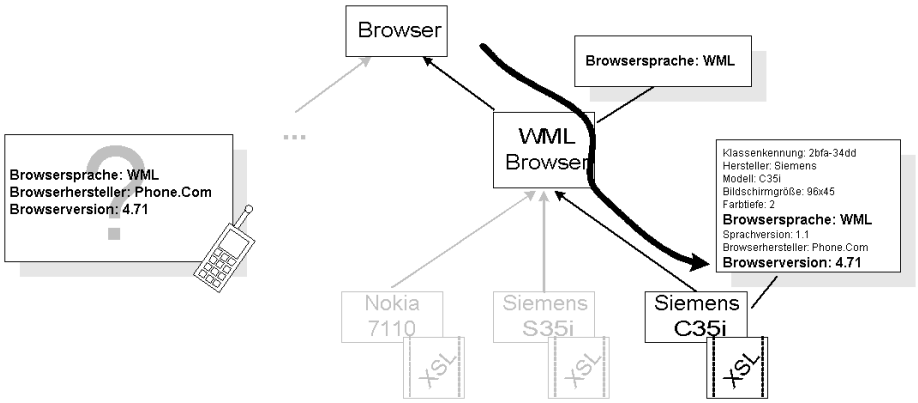


Abbildung 4.27: Navigation durch die Klassenhierarchie

Während das stark vereinfachte Beispiel darstellt, wie die Verwendung einer Typhierarchie die Zuordnung von Geräteprofilen zu einer bestimmten Geräteklasse ermöglicht, lässt es doch natürlich auch eine Reihe von Fragen aufkommen:

- Auf welcher Ebene sollen Strategien definiert sein? Sollen diese nur für konkrete Geräteklassen verfügbar sein oder auch für allgemeinere Klassen (wie z.B. die Klassen „Browser“, „HTML–Browser“ bzw. „WML Browser“ in Abbildung 4.26)?
- Ist im Beispiel die Verfeinerung nach Browsertypen gerechtfertigt? Was ist mit Geräten, die nicht über einen Browser verfügen? Oder allgemeiner: Nach welchen Kriterien soll die Hierarchie aufgebaut werden?
- Wäre im angeführten Beispiel nicht eine Klassifikation nach Herstellern als übergeordneter Hierarchiestufe sinnvoll, sofern das Profil einen Hersteller nennt? Oder allgemeiner gefasst: Muss eine Klassenhierarchie statisch sein oder wäre eine dynamische Anordnung nicht vorzuziehen?
- Was ist mit Geräten, die vom gleichen Hersteller stammen und von denen das eine lediglich die erweiterte Version (also eine Verfeinerung) des anderen ist: Müsste diese Abhängigkeit nicht in der Hierarchie berücksichtigt sein? Oder allgemeiner: Wie soll die Verfeinerung über mehrere Ebenen vonstatten gehen?

- Wie können Geräte berücksichtigt werden, die über mehrere Browser verfügen und somit sowohl als WML Browser als auch als Webbrowser verwendet werden können? Oder allgemeiner: Können mehrere Knoten in der Klassenhierarchie dieselbe Geräteklasse verkörpern?

Um diese Fragen beantworten zu können, soll das soeben Gesagte durch ein näheres Eingehen auf die konzeptionellen Hintergründe vertieft werden, wie sie im Bereich der objektorientierten Programmierung bestehen.

Am Ausgangspunkt soll nochmals kurz verdeutlicht werden, welchem Zweck die Verwendung eines Klassifikationssystems gilt. Hier lassen sich im Wesentlichen zwei Motivationen anführen. Wie im Beispiel geschildert, dient eine Klassenhierarchie vor allem der Organisation von Strategien und der sicheren Auswahl einer Strategie anhand übergebener Geräteeigenschaften. Noch vor der Auswahl von Strategien steht der zweite Grund, aus dem der Aufbau einer Hierarchie sinnvoll ist, nämlich die eigentliche Erstellung von Strategien. Im konkreten Beispiel werden Stylesheets verwendet, um die Informationen, die an ein Gerät übergeben werden sollen, in eine Form zu bringen, die von diesem Gerät optimal dargestellt werden kann. Soll nun für ein weiteres Gerät eine Strategie entwickelt, d.h. ein Stylesheet entworfen werden, so kann eine Klassenhierarchie hierbei helfen. Von einem bestimmten Knoten ausgehend, der die maximale Übereinstimmung mit dem neuen Gerät bietet, wird eine Verfeinerung gebildet, in der das Stylesheet der Oberklasse als Ausgangspunkt verwendet und entsprechend der Abweichungen der zu bildenden Geräteklasse zu seiner Oberklasse modifiziert wird. Die so konstruierte Unterklasse kann dann in den Baum eingefügt werden und als Ausgangspunkt für weitere Modifikationen dienen.

In der Objektorientierung werden hierarchische Typsysteme oftmals eingesetzt. Hier werden dann allgemeinere Konzepte (Oberklasse, Superklasse) durch Erweiterungen zu spezielleren Konzepten (Unterklasse, Subklasse) verfeinert, die in der verbildlichten Hierarchie dann unterhalb der Superklasse angeordnet werden. Beispiel: Lebewesen – Säugetier – Primat – Mensch. Durch den Aufbau von Sub– und Supertypbeziehungen lassen sich danach sehr komplexe Hierarchien aufbauen. Wichtig ist hierbei jedoch, dass sich jederzeit bestimmen lässt, worin die Verfeinerung zwischen Super– und Subtyp besteht. Beispiel: „Aufzucht der Kinder“ zwischen Lebewesen und Säugetier, „Anzahl der Finger an Händen und Füßen“ zwischen Säugetier und Primat, „Fähigkeit zum selbstbestimmten Handeln“ zwischen Primat und Mensch.

Doch ist nicht von vornherein klar, wie die Beziehungen zu verstehen sind und wie sich hieraus Hierarchien ergeben. Heuer führt in [Heu92] drei Ansätze ein, anhand derer hierarchische Beziehungen zwischen Objekten formalisiert werden können (Vgl. S.203ff) :

Is–A–Hierarchie, Spezialisierung : Bei der „Is–a“ Hierarchie handelt es sich um eine Beziehungsform, in der die Unterklasse sich aus der Verfeinerung der Oberklasse ergibt. Die in der Unterklasse vorhandenen Objekte stellen jeweils Spezialfälle der in der Oberklasse vorhandenen Objekte dar, die sich dadurch ergeben, dass zu den Eigenschaften der Oberklasse neue Eigenschaften hinzugefügt werden. Die Unterklasse unterscheidet sich also genau in den Eigenschaften, die sie zusätzlich zu den Eigenschaften der

Oberklasse besitzt. Im Zusammenhang mit dem oben genannten Beispiel stellt „Browser“ eine Oberklasse dar, die durch eine „Is–A“ Beziehung mit der Klasse „Webbrowser“ verbunden ist. Der Webbrowser stellt also eine Spezialisierung des Browsers dar. In gleicher Weise stellt dann die Klasse „Palm III“ eine Spezialisierung der Klasse „Webbrowser“ dar, der mit farbigem Bildschirm bestückte „Palm IIIc“ eine Spezialisierung der Klasse „Palm III“ usw.

Typhierarchie : Die Typhierarchie legt Beziehungen zwischen Klassen in der Form fest, dass die Methoden und Attribute von Objekten des Obertyps auch auf Objekte des Untertyps angewendet werden können. Heuer nennt in diesem Zusammenhang den Begriff der Substituierbarkeit: Jedes Objekt des Obertyps kann durch ein Objekt des Untertyps ersetzt, d.h. substituiert werden, ohne dass er sich hierbei in seinem Verhalten unterscheidet. Zieht man das obige Beispiel heran, so kann wiederum der „Webbrowser“ als Obertyp bezeichnet werden, „Palm III“ hingegen als Untertyp. Wird ein Webbrowser gefordert, so kann der Typ „Palm III“ verwendet werden, da er dasselbe Verhalten in den im Obertyp definierten Eigenschaften aufweist wie der Typ „Webbrowser“, nämlich die Fähigkeit zur Anzeige von HTML Dokumenten. Deutlicher wird das Gesagte noch zwischen den Typen „Palm III“ sowie „Palm IIIc“. So kann überall dort, wo der Typ „Palm III“ gefordert wird auch der Untertyp „Palm IIIc“ zum Einsatz kommen, der sich nur in der Anzahl verfügbarer Bildschirmfarben unterscheidet.

Klassenhierarchie : In der Klassenhierarchie wird in der Terminologie von objektorientierten Programmiersprachen eine Beziehung zwischen Klassen ausgedrückt, bei der die Oberklasse als Ausgangspunkt im Entwurf der Unterklasse dient. Die Oberklasse wird in diesem Zusammenhang auch als Implementierungsumgebung der Unterklasse bezeichnet. Sie stellt alle Eigenschaften zur Verfügung, die zunächst in der Unterklasse erscheinen und im weiteren Verlauf dann abgeändert werden können. Sieht man im Beispiel das Stylesheet als eine Eigenschaft der Klasse an, so könnte das Stylesheet für allgemeine Webbrowser, welches der Klasse „Webbrowser“ angehängt ist, als Ausgangspunkt für den Entwurf eines Stylesheets für die Klasse „Toshiba Satellite“ dienen. Ebenso, und hierin liegt ein Unterschied zu der in der Abbildung gezeigten Hierarchie, könnte das Stylesheet der Klasse „C35i“ als Ausgangspunkt für den Entwurf der Klasse „S35i“ dienen, bzw. das Stylesheet der Klasse „Palm III“ als Ausgangspunkt für den Entwurf der Klasse „Palm IIIc“, bei dem dann die Fähigkeit zur Darstellung von Farben hinzugenommen wird. Andererseits – und hierin liegt gleichsam das Problem dieser Formalisierung – könnte natürlich auch das Stylesheet zur Klasse „Palm IIIc“ als Ausgangspunkt für den Entwurf der Klasse „Palm III“ verwendet werden, wobei in diesem Fall das Ziel durch einen Wegfall der Eigenschaft zur Darstellung farbigen Inhalts erreicht würde.

In Anlehnung an [Heu92] sollen die drei Konzepte in einer Abbildung nochmals zusammengefasst werden, wobei die Unterschiede in der Spezialisierung verdeutlicht werden (vgl. Abb. 4.28). In der „Is–A“ bzw. *Spezialisierungshierarchie* wird die nächste Hierarchiestufe erreicht, indem weitere Eigenschaften in

die Geräteklasse aufgenommen werden. Heuer spricht hier von einer Integritätsbedingung, in der die Unterklasse eine Übermenge der Oberklasse darstellt. In der *Typhierarchie* wird die nächste Stufe hingegen erreicht, indem eine Erweiterung des Verhaltens durchgeführt wird. Wichtig ist in diesem Fall das gleiche Verhalten, das auf beiden Ebenen vorliegen muss. In der *Klassenhierarchie* wird die nächste Stufe schließlich erreicht, indem die Implementation der Oberklasse als Ausgangspunkt für den Entwurf der Unterklasse herangezogen wird. Heuer spricht in diesem Zusammenhang davon, dass die „Is–A“ Hierarchie eine logische Unterteilung darstellt, die Typhierarchie eine Unterteilung aus der Sicht des Anwenders und die Klassenhierarchie schließlich eine Unterteilung aus der Sicht des Entwicklers.

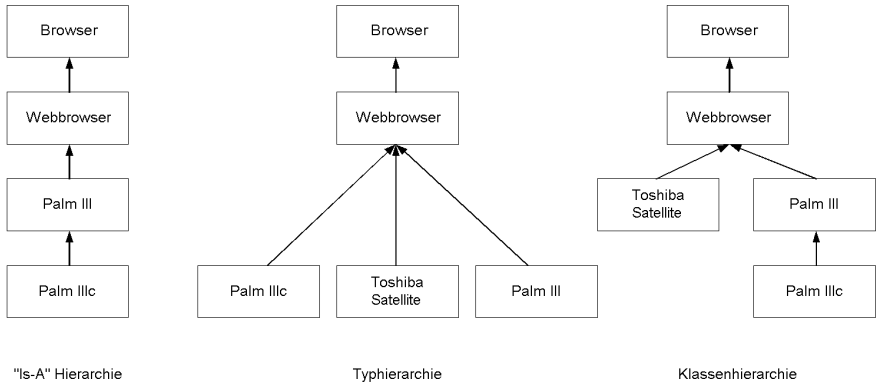


Abbildung 4.28: Verschiedene Beziehungshierarchien im Vergleich

Wie lässt sich das Gesagte auf die zwei formulierten Ziele anwenden, also die Konstruktion geeigneter Strategien (konkret: passender Transformationsregelsätze) sowie die Selektion einer passenden Strategie für das zu unterstützende Gerät? Denn obwohl die Abbildungen den Versuch unternehmen, die aus der Objektorientierung stammenden Konzepte in den Kontext dieser Arbeit zu übernehmen, handelt es sich letztlich doch um Aspekte der Softwaretypisierung – nicht jedoch der Hardwaretypisierung.

Für die Entwicklung von Stylesheets scheint die Klassenhierarchie am ehesten geeignet zu sein. In ihr kann jeweils an die Stelle im Baum navigiert werden, wo die größte Übereinstimmung zwischen dem Stylesheet jenes Knotens und dem zu erstellenden Stylesheet besteht. Dieses kann dann als Vorlage verwendet und entsprechend den Anforderungen an das neue Gerät angepasst werden. Der so entstehende Knoten kann in die Hierarchie eingefügt werden und dann im Zuge der weiteren Verfeinerung als Ausgangspunkt Verwendung finden. Heuer nennt die Klassenhierarchie ein *„... reines Implementierungswerkzeug...“* und in diesem Sinne leistet es beim Entwurf von Strategien einen wertvollen Dienst. Für die Auswahl geeigneter Strategien eignet sich die Klassenhierarchie hingegen nur bedingt. Wird nämlich die Unterklasse stark in ihren Eigenschaften gegenüber der Oberklasse abgewandelt, so kann eine Situation eintreten, in der ein Stylesheet der Oberklasse nicht mehr adäquat verwendet werden kann, um ein Gerät einer möglichen Unterklasse zu unterstützen. Beispiel: Weist eine Oberklasse als Eigenschaft einen größeren Bildschirm als eines der Unterklasse auf (z.B.

„Toshiba Satellite“ im Vergleich zu „Toshiba Libretto“), so kann die hierfür optimierte Ausgabe auf einem Gerät der Unterklasse unter Umständen nicht mehr dargestellt werden.

Zur Auswahl einer geeigneten Strategie ist hingegen die „Is–A“ Hierarchie vorzuziehen, bei der jede Unterklasse eine Spezialisierung ihrer Oberklasse darstellt. Da nach dieser Hierarchie die Unterklasse jede Eigenschaft der Oberklasse aufweist und an deren Stelle eingesetzt werden kann, würde also das oben genannte Problem nicht auftauchen, „Toshiba Libretto“ stünde also nicht in „Is–A“ Beziehung zu „Toshiba Satellite“. Auf der anderen Seite könnte die oben dargestellte Beziehung zwischen „Palm III“ und „Palm IIIc“genutzt werden, da der „Palm IIIc“ eine echte Obermenge zu „Palm III“ darstellt. Das Stylesheet des allgemeineren Geräts (ohne Farbdarstellung) könnte also zur Adaption von Informationen an das speziellere Gerät (mit Farbdarstellung) genutzt werden, wobei diese speziellen Fähigkeiten natürlich ungenutzt blieben. Zur Konstruktion neuer Klassen bzw. den ihnen beigeordneten Stylesheets eignet sich die „Is–A“ Hierarchie hingegen nur eingeschränkt. Da Eigenschaften der Unterklasse nur eine Spezialisierung der Eigenschaften aus der Oberklasse darstellen dürfen, ergibt sich eine sehr begrenzte Ausdrucksmächtigkeit zur Definition der Unterklasse.

Wie lässt sich nun die Typhierarchie bewerten, die in der objektorientierten Programmierung genau die Substituierbarkeit der Oberklasse durch die Unterklasse sicherstellt? Zunächst scheint sich hieraus ein Ausschlusskriterium für die Verwendung der Typhierarchie im Kontext dieser Arbeit zu ergeben. Die Möglichkeit zur Substitution ergibt sich wie beschrieben dadurch, dass die Eigenschaften der Oberklasse durch Vererbung an die Unterklasse weitergegeben werden. Nach einer Substitution kann die Unterklasse deswegen an Stelle der Oberklasse verwendet werden, weil lediglich die Eigenschaften (in der objektorientierten Programmierung also die Attribute und Methoden) in Anspruch genommen werden, die aus dieser Vererbungsbeziehung stammen. In diesem Bereich ist das Verhalten der Unterklasse also stimmig. Vordergründig führt dieses zu Problemen bei der Auswahl von Strategien und insbesondere bei der Nutzung von Stylesheets. Hier würden bei einer Substitution nicht nur die Anteile aus dem Stylesheet der Unterklasse genutzt, die von der Oberklasse übernommen wurden, sondern das gesamte Stylesheet, d.h. die gesamte, in Teilen abweichende Strategie. Das Verhalten wäre also nicht gleich, sondern würde in wesentlichen Punkten abweichen. Konkret würde also ein Stylesheet verwendet werden, das eine Zielrepräsentation erzeugt, die eben nicht in ihrem Verhalten der Repräsentation entspricht, die mit dem Stylesheet der Oberklasse erzeugt werden würde. Wenn also eine Substitution in der Art der objektorientierten Programmierung stattfinden würde (also das Allgemeine durch das Spezielle ersetzend), so würde dieses zu unbefriedigenden Ergebnissen führen. Für die Strategiewahl kann nun jedoch festgestellt werden, dass die Substitution – sofern sie überhaupt notwendig wird – in genau entgegengesetzter Richtung stattfindet: Kann keine Geräteklasse für ein spezielles Gerät ermittelt werden, so wird stattdessen die Klasse eines allgemeineren Gerätes ausgewählt und die hiermit verbundene Strategie befolgt.

Bei der Konstruktion von Geräteklassen bzw. Stylesheets, bei der es in erster Linie um die Verwendung einer vorhandenen Implementation als Ausgangspunkt für eine Abwandlung hiervon geht, steht im Kontext dieser Arbeit nicht primär die Vererbung von Verhalten im Vordergrund, sondern vielmehr die Vererbung

von Eigenschaften. Soll eine neue Geräteklasse eingeführt werden, so kann eine bestehende Klasse hierfür als Vorlage dienen, von der ausgegangen wird. Ähnlich soll beim Entwurf einer neuen Strategie die existierende Strategie hergenommen und durch Abwandlung zu einer angepassten Strategie umgeformt werden. Es geht also nicht um die bloße Anwendbarkeit von ererbten Eigenschaften, sondern um deren Abwandlung. Insbesondere für den Spezialfall der Stylesheets, für die kein objektorientiertes Konzept definiert ist, stellt die Typhierarchie also auch für den Entwurf kein adäquates Modell dar.

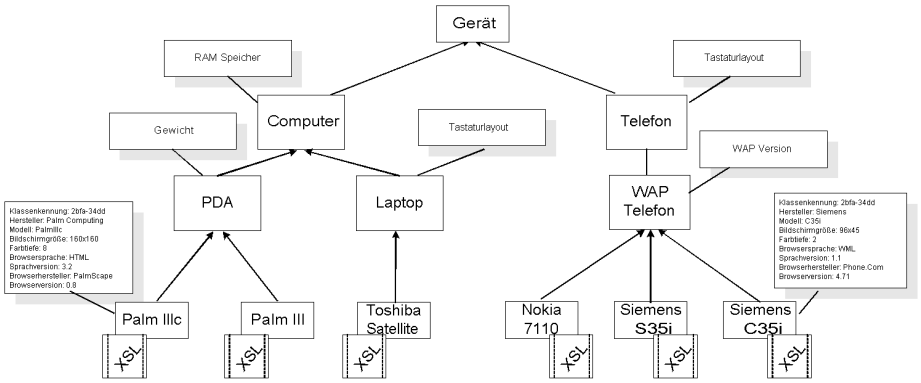


Abbildung 4.29: Abgewandelte „Is–A“ Hierarchie mit geändertem Beziehungsgeflecht

Wie sich bereits aus Abbildung 4.28 ergibt, muss die für den Entwurf von Klassen bzw. Strategien verwendete Hierarchie nicht unbedingt derjenigen für die Auswahl der passenden Strategie gleichen. Je nach Verwendungszweck können sich sogar deutliche Abweichungen in den Bäumen ergeben, die sich aus den Beziehungen zwischen den Knoten herleiten lassen. Aber auch innerhalb eines Anwendungsgebiets liegt nicht a priori fest, wie die Geräteklassen zueinander in Beziehung stehen. Die tatsächliche Relation ergibt sich aus dem Entwurfsvorgang, je nachdem, auf welche Eigenschaften der Betrachtungsschwerpunkt gelegt wird. In Abbildung 4.26 ist beispielsweise eine gültige „Is–A“ Hierarchie dargestellt, bei welcher der auf dem Gerät vorhandene Browser als wesentliche Eigenschaft zur Strukturierung der Hierarchie benutzt wurde. Würde nun andererseits die Art des Gerätes oder dessen Hersteller als wesentliches Merkmal angesehen, so könnte sich hingegen eine Hierarchie wie in Abbildung 4.29 ergeben. Während hier immer noch dieselben Strategien vorhanden und in Stylesheets ausgedrückt sind, ergibt sich durch die komplexere Hierarchie im Falle der Auswertung eines Geräteprofils abhängig von dessen Inhalt unter Umständen ein abweichendes Ergebnis.

Weiter oben in diesem Abschnitt wurden mehrere Fragen formuliert, die sich hinsichtlich einer Verwendung von Hierarchien im Zusammenhang dieser Arbeit ergeben. Diese lassen sich mit Blick auf die Ausführungen zu den verschiedenen Beziehungstypen nunmehr wie folgt beantworten.

1. Können Strategien nur Blattknoten zugeordnet werden? Dem ist nicht unbedingt so. Abhängig vom Einsatzzweck der Hierarchie zum Entwurf

bzw. zur Verwendung von Strategien hängt es vor allem von der Art der Spezialisierung ab, die sich durch den fraglichen Knoten ergibt, ob hier eine Strategie zugeordnet werden kann. Je abstrakter die Verfeinerung ist (z.B. zwischen „Gerät“ und „Computer“) desto unwahrscheinlicher ist es, dass hier eine geänderte Strategie notwendig ist. Je konkreter die Verfeinerung andererseits ist (z.B. zwischen „WAP Telefon“ und „S35i“), desto sinnvoller ist die Zuordnung.

2. Nach welchen Kriterien soll die Hierarchie aufgebaut werden? Dieses hängt zunächst vom Anwendungszweck ab. Die „Is–A“ Hierarchie zur Strategieauswahl dürfte von der Klassenhierarchie zur Strategiekonstruktion abweichen. Innerhalb eines Verwendungszwecks ergibt sich die tatsächliche Hierarchie, abgesehen von der Forderung nach Spezialisierung von Stufe zu Stufe, vornehmlich aus den zu erwartenden Geräteprofilen. Die Hierarchie sollte die Auswahl der jeweils optimalen Strategie zulassen und entsprechend gewählt werden.

3. Muss eine Klassenhierarchie statisch sein? Ein formales Argument lässt sich hierfür nicht anbringen. Obwohl der Realisierungsaufwand hierfür deutlich höher einzuschätzen ist, wird aus den vorigen Punkten klar, dass eine statische Hierarchie insbesondere zur Strategieauswahl nicht sinnvoll ist. Je nach Gewichtung der Eigenschaften könnte sich eine andere Hierarchie als geeigneter erweisen, um zur Auswahl der optimalen Strategie zu kommen. Auch sich wandelnde Geräteprofile könnten eine Umstrukturierung der „Is–a“ Hierarchie erforderlich machen.

4. Wie sollte die Verfeinerung über mehrere Ebenen vonstatten gehen? Neben den oben beschriebenen allgemeinen Bedingungen für den Aufbau einer Spezialisierungs- bzw. Klassenhierarchie gibt es hierzu keine formale Vorgabe. Die konkrete Hierarchie ergibt sich also aus den Eigenschaften der zu klassifizierenden Geräte und aus den zu erwartenden Eigenschaften der übermittelten Geräteprofile.

5. Können mehrere Knoten in der Klassenhierarchie dieselbe Geräteklasse verkörpern? Formal ist dieses zunächst nicht möglich, da sich die Hierarchie aus der Verfeinerung von Eigenschaften ergibt. Eine Geräteklasse kann im Prinzip also nicht auf mehreren Ebenen bzw. in verschiedenen Bereichen der Hierarchie auftauchen. Eine solche Möglichkeit ergibt sich allenfalls dann, wenn für die verschiedenen Eigenschaften eines Gerätes eigene Klassen geschaffen werden. Für ein Gerät, auf dem der Browser „Opera“ installiert ist und der sowohl zur Anzeige von HTML Seiten als auch von WML Seiten in der Lage ist, müssten anstelle der Geräteklasse „Opera“ die Klassen „Opera–WAP“ sowie „Opera–Web“erstellt und mit den entsprechenden Eigenschaften versehen werden. Beide könnten sodann an verschiedenen Stellen in die Hierarchie eingeordnet werden.

Es wurde auch in den letzten Ausführungen deutlich, dass die vornehmliche Motivation hinter der Verwendung von Hierarchien im Zusammenhang mit dieser Arbeit in der Formulierung und der sich hieraus ergebenden Auswahl passender Strategien für die Behandlung von Geräten liegt. Nachdem zu Beginn des Abschnitts näher auf Eigenschaften, Profile und Klassen eingegangen wurde und

dann Betrachtungen angestellt wurden, wie diese in Hierarchien strukturiert werden können, soll nunmehr die Auswahl von Strategien beleuchtet werden.

4.3.5.4 Strategien zur optimalen Diensterbringung

Wie oben angedeutet, bilden die Merkmale zu unterstützender Geräte die Grundlage zum Aufbau eines Klassifikationssystems und dieses System wiederum die Basis für die Auswahl optimaler Strategien. Jeder Klasse von Geräten wird eine bestimmte Strategie zugeordnet und eine ausreichende Menge an Merkmalen, durch welche die Klasse eindeutig gekennzeichnet werden kann. Obwohl in der Literatur kaum explizite Hinweise auf die Verwendung von Integrationsstrategien im Zusammenhang mit Geräteklassen existieren, sind grundsätzlich verschiedenste Strategien denkbar, die einer Klasse zugeordnet sein können. Der Strategiebegriff hängt vornehmlich von der Integrationsinfrastruktur ab, in deren Kontext er verwendet wird. Im vorigen Kapitel wurde deutlich, dass Integrationsansätze jeweils charakteristische Optimierungsziele besitzen, die anhand bestimmter Strategien verfolgt werden.

Im Zusammenhang mit der in dieser Arbeit entworfenen Integrationsinfrastruktur bedeutet „Strategie“ konkret, dass die bestmögliche Umsetzung einer abstrakten Beschreibung der Benutzungsschnittstelle einer Anwendung in eine konkrete Repräsentation zur Anzeige auf einem Mobilgerät zu wählen ist. Für alle Strategien gilt jedoch, dass anhand von Merkmalen des Geräts die passende Klasse gefunden und dann entsprechend der in ihr gespeicherten Strategie gehandelt werden muss.

Für die Verbindung zwischen Klasse und Strategie gibt es im Wesentlichen zwei Möglichkeiten. Die Strategie kann als Teil der Klasse realisiert werden bzw. einen eindeutigen Verweis auf eine externe Strategiedefinition enthalten. In diesem Fall ergibt sich die Vorgehensweise direkt, sofern anhand der ermittelten Merkmale die entsprechende Klasse bestimmt werden konnte. Diese Vorgehensweise ist insofern etwas inflexibel, als bei der Definition von Klassen bereits feststehen muss, dass die Geräteklassen zur Auswahl von Strategien verwendet werden sollen und wie diese Strategien aussehen, bzw. wie auf diese verwiesen werden kann. Andererseits kann die zu wählende Strategie besser als Teil der Eigenschaft des Geräts aufgefasst werden und als solche gemeinsam mit den eigentlichen Merkmalen des Geräts gespeichert werden. Da vorhandene Strategien bei diesem Vorgehen bereits vorab feststehen, sollen diese als *statische* Strategien bezeichnet werden.

Eine weitere Möglichkeit besteht darin, völlig auf die Verwendung von Geräteklassen zu verzichten und Strategien direkt aus den übermittelten Geräteprofilen abzuleiten. Überträgt ein Gerät die Information, dass es eine Bildschirmgröße von 20 mal 20 Zeichen besitzt oder zur Darstellung von Farben fähig ist, so könnte dieses entsprechend des in dieser Arbeit gewählten Strategiebegriffs in eine passende Transformationsregel umgesetzt werden, anhand derer dann die Informationen auf dieses Merkmal des Geräts angepasst werden können. Die Strategie ergibt sich hierbei also nicht aus der Identifikation eines bestimmten Gerätetyps, sondern nur aus der Übermittlung bestimmter Fähigkeiten. Da in diesem Fall die Strategie nicht vorab feststeht und sich erst zur Laufzeit aus den übermittelten Eigenschaften ergibt, wird sie als *dynamische* Strategie bezeichnet.

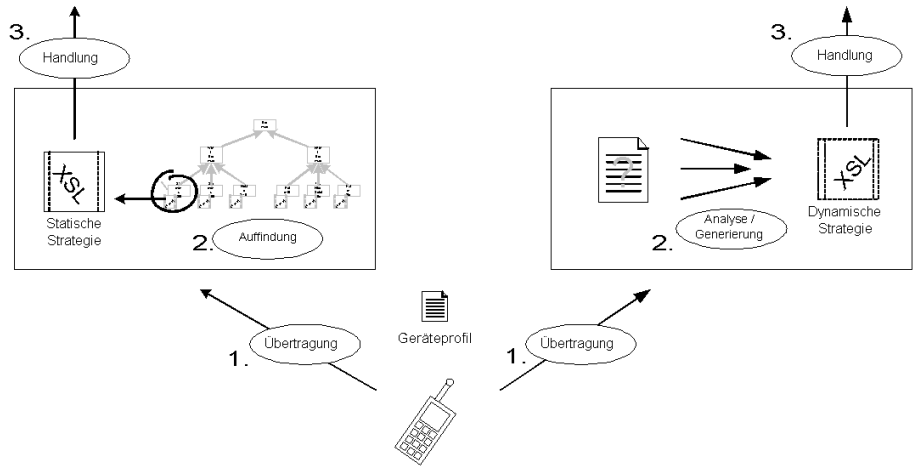


Abbildung 4.30: Ermittlung statischer (links) und dynamischer (rechts) Strategien im Vergleich

Der Vorteil dynamischer Strategien besteht darin, dass bei der Definition und Kodierung von Merkmalen und deren Zusammenstellung zu einer Geräteklasse kein Wissen über Strategien oder überhaupt einen Kontext vorhanden sein muss, in dem diese verwendet werden. Die Definition einer Strategie und deren Umsetzung ist ausschließlich Sache der Anwendung, welche die Geräteprofile analysiert und hierauf aufbauend eine bestimmte Strategie realisiert. Der Nachteil liegt andererseits darin, dass auf diese Weise keine explizite Verbindung mehr zwischen der Geräteklasse und der für sie geltenden Strategie sichtbar ist. Die zu verwendende Strategie ist aus dem Blickwinkel der Integration ein natürlicher Teil der Geräteklasse, wird aber bei diesem Vorgehen nicht als solcher repräsentiert.

Die Ermittlung dynamischer bzw. statischer Strategien ist in Abbildung 4.30 skizziert. Nachdem im ersten Schritt das Geräteprofil an das Anwendungssystem übertragen wurde, wird abhängig vom eingeschlagenen Weg vorgegangen. Wie in der Abbildung links dargestellt, wird bei statischen Strategien im zweiten Schritt aus dem Geräteprofil eine korrespondierende Klasse hergeleitet und die hiermit verbundene Strategie als Grundlage der im dritten Schritt erfolgenden Handlung herangezogen. Im Gegensatz dazu wird bei der Verwendung dynamisch generierter Strategien, in der Abbildung rechts dargestellt, aus dem Geräteprofil direkt eine Strategie abgeleitet und dann die dynamisch generierte Strategie als Basis der folgenden Handlung hergenommen.

Wie genau statische Strategien mit der ihnen zugrunde liegenden Klasse verbunden sind, ist primär keine konzeptionelle Frage, sondern ergibt sich vielmehr aus dem Strategieverständnis und aus der gewählten Implementation. Prinzipiell ist nur wichtig, dass die Zuordnung zwischen Klasse und Strategie eindeutig ist. Dieses kann konkret durch die Einführung von Referenzen in die Klassendefinition erreicht werden oder durch direkte Integration der Strategiedefinition in die Klasse. Kann eine Strategie, wie im Falle von XSL Regelsätzen zur Transformation von XML, direkt durch Regeln beschrieben werden, so ist die direkte Integration einfacher. Basiert die zu wählende Strategie hingegen auf komplexen

Algorithmen, so wird die Strategie wahrscheinlich durch eine eindeutige Referenz gekennzeichnet und lediglich diese in die Klassendefinition integriert. Die eigentliche Strategie wird dann extern kodiert und vom Anwendungssystem anhand der Referenz identifiziert und ausgewählt. Dieses ist aber letztlich weniger eine konzeptionelle Frage, sondern eine der Implementation.

4.3.6 Konkrete Realisierungstechniken

Nachdem im vorigen Abschnitt auf konzeptionelle Aspekte eingegangen wurde, soll nunmehr auf konkrete Realisierungstechniken aus dem Bereich der Geräteprofile und Geräteklassen eingegangen werden. Dabei wird die Auswahl vor dem Hintergrund des Kontextes dieser Arbeit durchgeführt, und es werden entsprechend Techniken vorgestellt, die zur Nutzung im Zusammenhang mit mobilen Geräten und der Transformation von abstrakten Informationen in konkrete Repräsentationen auf der Basis von XML geeignet sind.

4.3.6.1 Resource Description Framework

Das Resource Description Framework (RDF) [BG99] ist eine universelle Sprache zur Definition von Metabeschreibungen [ABK+00]. Sie wurde ursprünglich mit dem Ziel entwickelt, maschinenlesbare Eigenschaften von Webinhalten (vgl. [RHDS99], Abschnitt 1.0) beschreiben zu können. Mit Hilfe solcher Metadaten sollte die Klassifikation von Inhalten ermöglicht und in der Folge ein optimierter Zugriff auf diese Inhalte anhand formulierbarer Kriterien gewährleistet werden. Mittels RDF Beschreibungen können Suchmaschinen eine Auswahl aus Dokumenten ermöglichen, die über den bloßen Vergleich von Schlagworten hinausgeht, Dokumente in digitalen Bibliotheken beschriftet werden oder Komponenten in einem Softwarebaukasten näher bezeichnet werden können. Zwar ist RDF ursprünglich nicht als XML Anwendung entstanden, es existiert jedoch eine Spezifikation, welche die Kodierung von RDF in XML beschreibt [BG99].

RDF basiert auf der Formulierung von Tripeln aus Ressource (Subjekt), Eigenschaftsname (Prädikat) sowie Eigenschaftswert (Objekt). Folgen solcher Tripel können zusammengefasst werden und ergeben dann eine Aussage. Über solche Aussagen können Komponenten in ihren Eigenschaften beschrieben werden. Die Aussage stellt mithin das Kernkonzept von RDF dar.

Formal werden die Begriffe wie folgt definiert (vgl. [NHO00]):

Definition Ressource: Alles, was in RDF durch Ausdrücke beschrieben werden kann, wird als Ressource bezeichnet. Solche Ressourcen können Webseiten, Teile von Webseiten oder auch ganze Sammlungen von Webseiten sein. Darüber hinaus können auch externe Objekte wie z.B. Bücher o.ä. in RDF als Ressourcen verstanden werden.

Definition Eigenschaft: Eine Eigenschaft ist ein bestimmter Aspekt, ein Merkmal oder Attribut, das eine Ressource beschreibt. Jede Eigenschaft hat einen bestimmten Sinn und einen definierten Wertebereich. Für jede Eigenschaft ist festgelegt, welche Art von Ressource sie beschreibt und in welcher Beziehung zu anderen Eigenschaften sie steht.

```
<rdfs:Class rdf:ID="Rakete"
 <rdfs:subClassOf
    rdf:resource="http://www.w3.org/TR/WD-rdf-schema#Resource/">
</rdfs:Class>

<rdfs:Class rdf:ID="ChemischeRakete"
 <rdfs:subClassOf
    rdf:resource="http://www.w3.org/TR/WD-rdf-schema#Resource/">
</rdfs:Class>
```

Abbildung 4.31: Ressourcendefinition in RDF

Definition Aussage: Die Kombination aus Ressource, Eigenschaft und Eigenschaftswert wird als Aussage bezeichnet. Als Eigenschaftswert kann ein einfacher Wert entsprechend eines einfachen Datentyps oder der Verweis auf eine Ressource vorliegen.

Anhand eines Beispiels soll das Gesagte verdeutlicht werden. Grundlage für die Formulierung von Aussagen ist zunächst die Spezifikation von Ressourcen. Wie bereits gesagt werden in RDF keine Beschränkungen hinsichtlich der definierbaren Ressourcen auferlegt. In einem einfachen Beispiel sollen Aussagen über Raumfahrzeuge gemacht werden (vgl. [ABK+00] S.260 ff.). Hierzu werden zunächst in der in Abbildung 4.31 dargestellten Weise die Ressourcen „Rakete“ und deren Verfeinerung „chemische Rakete“ definiert. Anschließend können die Eigenschaften dieser Ressourcen definiert werden. Jede Eigenschaft besteht dabei aus einem Eigenschaftsnamen und einem zugehörigen Eigenschaftswert. Im vorliegenden Beispiel soll ausgesagt werden können, welche Art von Treibstoff zum Betreiben einer bestimmten Rakete verwendet werden können. Entsprechend muss zunächst definiert werden, dass die Eigenschaft „Treibstoff“ der Ressource „ChemischeRakete“ zugeordnet sein soll. Siehe hierzu Abbildung 4.32.

Schließlich können über die zuvor definierten Ressourcen und unter Verwendung der definierten Eigenschaften Aussagen getroffen werden. Abbildung 4.33 zeigt beispielsweise, wie der zur Befüllung der Rakete geeignete Treibstoff festgelegt werden kann.

Bei RDF handelt es sich um einen sehr allgemeinen Ansatz. Zwar wurde RDF

```
<rdfs:Class rdf:ID="Treibstoff"
  <rdfs:subClassOf
     rdf:resource="http://www.w3.org/TR/WD-rdf-schema#Resource/">
  </rdfs:Class>

  <rdfs:Property ID="Treibstoff">
    <rdfs:range rdf:resource="#Treibstoffe" />
    <rdfs:domain rdf:resource="#chemischeRakete/">
  </rdfs:Property>
```

Abbildung 4.32: Definition von Eigenschaften in RDF

```
<? Xml version="1.0" ?>
 <rdf:RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <ChemischeRakete ID="Luna" xmlns="urn:my-rdf-rocket-schema"/>
  <rdf:Description about="Luna">
     <Treibstoff>Wasserstoff</Treibstoff>
  </rdf:Description>
</rdf:RDF>
```

Abbildung 4.33: Formulierung von Aussagen über zuvor definierte Ressourcen in RDF

zur Definition von Metabeschreibungen für das Worldwide Web geschaffen. Aufgrund seines generischen Aufbaus ist es jedoch zur Beschreibung von Ressourcen in fast beliebigen Szenarios zu verwenden, und durch die Verwendung von XML wird seine Adaption an spezielle Anwendungsgebiete stark vereinfacht. Die Beschreibung von Geräteeigenschaften und Benutzerpräferenzen stellt ein solches Anwendungsgebiet dar. Auch hier sollen im Prinzip Metadaten erfasst werden, nämlich Informationen über Geräte und deren Parametrisierung.

4.3.6.2 Composite Capability/Preference Profile

Das Composite Capability/Preference Profile Rahmenwerk (etwa: *„zusammengesetztes Profil aus Fähigkeiten bzw. Präferenzen“*) (CC/PP) stellt einen Ansatz dar, die hierbei anfallenden Metadaten auf der Basis von RDF Ressourcen zu erfassen. Nilsson beschreibt in einer Arbeitsvorlage zur Standardisierung von CC/PP den Bezug zu RDF folgendermaßen: *„Fundamentally, the CC/PP framework starts with RDF and then overlays a CC/PP–defined set of semantics that describe profiles.“* [NHO00]. CC/PP nutzt also die soeben beschriebenen Mechanismen zur Definition von Ressourcen und zur Formulierung von Aussagen über diese Ressourcen, um hiermit die in einem Profil zu vereinenden Eigenschaften zu beschreiben. Im Gegensatz zu RDF stehen bei CC/PP jedoch nicht das Worldwide Web und die dort verwendeten Webseiten im Vordergrund der zu erfassenden Metadaten, sondern Geräte und deren Benutzer in einem verteilten System. Wie bereits aus dem Namen deutlich wird, geht es primär um Fähigkeiten (Capability) und Präferenzen (Preference), für die eine optimale Repräsentation geboten werden sollen.

Die Speicherung von Fähigkeiten und Präferenzen geschieht bei CC/PP im Sinne einer optimalen Anpassung von Inhalten an die Anforderungen des zugreifenden Benutzers bzw. Gerätes. Eine solche Anpassung kann sich in der Übertragung in eine andere Datenrepräsentation äußern (WML anstelle von HTML für ein WAP Telefon). Sie kann aber auch die Übersetzung in eine andere Sprache bedeuten, wenn der Benutzer als Präferenz die Darstellung der Information in dieser Sprache gefordert hat. Natürlich sind auch komplexe Anpassungen denkbar, wenn der Benutzer beispielsweise unter mehreren Sprachen eine bestimmte Sprache bevorzugt, die Kodierung in WML aber nur in einer der weniger gewünschten Sprachen möglich ist. Hier muss auf der Basis von Fähigkeiten und Präferenzen

ein Algorithmus verwendet werden, der die tatsächliche Ausgabe näherungsweise zu bestimmen vermag. Die Erleichterung solcher Vorgänge wurde bei CC/PP als wesentliches Ziel formuliert.

Dabei sollte die universelle Einsetzbarkeit von CC/PP durch das Aufsetzen auf RDF und die Verwendung von XML in jedem Fall gewährleistet werden. Während CC/PP primär auf die Verwendung im Kontext von Browsern und die Behandlung der in diesem Zusammenhang anfallenden Metadaten ausgerichtet wurde, können jederzeit Erweiterungsschemata hinzugefügt werden, um auch Metadaten in beliebigen anderen Kontexten zu erfassen.

Was zuvor noch allgemein als Merkmal bezeichnet wurde, wird in CC/PP als Eigenschaft (property) benannt. CC/PP unterscheidet zwischen Präferenzeigenschaften (preferential properties), Eigenschaften der Hardwareplattform (hardware platform properties) sowie Eigenschaften der Softwareplattform (software platform properties). Im Wesentlichen herrscht hierbei eine Übereinstimmung mit den oben genannten Gruppen. CC/PP liegt ein recht einfaches Datenmodell zugrunde: Alle erfassten Eigenschaften werden in Tabellen zusammengefasst. Jede Zeile einer solchen Tabelle enthält eine RDF Aussage, bestehend aus Ressource, Eigenschaftsname sowie einem einfachen, atomaren Eigenschaftswert (Subjekt/Prädikat/Objekt, siehe oben). Tabellen bestehen entsprechend aus Sammlungen von RDF Aussagen. Eine oder mehrere Tabellen ergeben zusammen ein Profil bzw. Geräteprofil. Ein Profil ist jeweils einer Kommunikationssitzung oder einer Transaktion zugeordnet. In ihrem Rahmen werden sie zwischen Sender und Empfänger ausgetauscht, wobei sie Informationen tragen, die in diesem Kontext relevant sind.

Definition Geräteprofil: Ein Geräteprofil ist eine Sammlung von Ressourcen, Eigenschaftsnamen und Eigenschaftswerten, die den Zustand des Gerätes zu einem bestimmten Zeitpunkt eindeutig beschreiben oder besteht aus Referenzen auf diese. Profile sind Sitzungen oder Transaktionen beigeordnet und sind in deren Kontext gültig.

Werden nicht nur aktuelle Wertbelegungen, sondern auch Defaultwerte kodiert, so entwickelt sich die einfache Tabelle in eine komplexere Datenstruktur, die sich in Anlehnung an die RDF Notation als ein RDF Graph darstellen lässt. Knoten stellen hierbei die Ressourcen dar, Kanten die Eigenschaften und Blätter deren Belegungen.

Abbildung 4.34 stellt einen solchen RDF Graphen zu einem einfachen Beispiel dar. In dem dargestellten Profil sind Eigenschaften der Hardwareplattform und der Softwareplattform kodiert. Neben direkten Belegungen (Bluetooth, Images usw.) sind auch Defaultwerte angegeben. Das gesamte Profil ergibt sich entsprechend durch die Überlagerung von Defaultwerten und aktuellen Belegungen, wobei die aktuellen Belegungen jeweils Vorrang besitzen.

Im konkreten Beispiel sind verschiedene Eigenschaften der aktuellen Netzwerksitzung eines Gerätes in zwei Profilen (Softwareplattform bzw. Hardwareplattform) zusammengefasst. Jede der vermerkten Eigenschaften verfügt über einen Namen und eine Wertbelegung. Die Menge aller bekannten Eigenschaften wird als Vokabular oder Attributvokabular bezeichnet[Nil00]. CC/PP verfügt über ein kleines Kernvokabular, das über beliebig viele Erweiterungsvokabularien zur Beschreibung weiterer Eigenschaften ausgebaut werden kann. Eine detaillierte

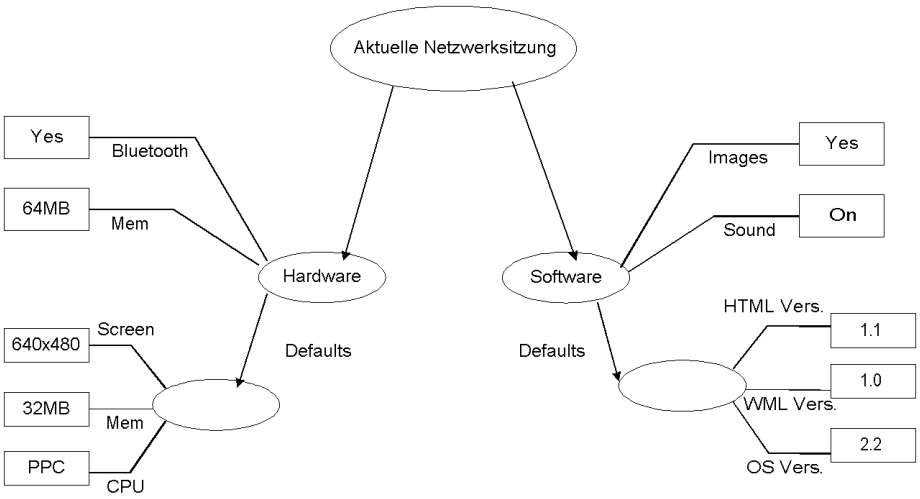


Abbildung 4.34: CC/PP Profil mit Defaultwerten als RDF Graph (nach [RHDS99])

Beschreibung des CC/PP Kernvokabulars sowie einer Reihe von Erweiterungsvokabularien kann [Kly00] entnommen werden. Das Kernvokabular enthält Basisdefinitionen wie „Component“, „Property“ oder „Attribute“, die als Grundlage zur Definition von Profilen Verwendung finden.

Erweiterungsvokabularien werden einerseits zur weiteren Verfeinerung von CC/PP verwendet. So gibt es ein Erweiterungsvokabular für Stellvertreter (sog. Proxies), mit dem Eigenschaften spezifiziert werden können, wie sie in Profilen von Diensten vorkommen, die in die Interaktion zwischen Endgerät und Informationsquelle eingeschleift sein können (z.B. Webproxy, WAP Gateway usw.). Ein Erweiterungsvokabular, auf das im Weiteren noch eingegangen werden soll, ist UAProf [Gro99], das zur Definition von Profilen für WAP Geräte entworfen wurde. Weitere Vokabularien dienen zur Beschreibung von Objekten in MPEG–4, MPEG–7 usw. Andererseits werden Erweiterungsvokabularien zur Einbindung von ansonsten inkompatiblen Ansätzen zur Beschreibung von Ressourcen verwendet. So existiert ein Erweiterungsvokabular für JetSend, ein Rahmenwerk zur Beschreibung von Druckereigenschaften [Pac00], für Salutation, ein Rahmenwerk zum Austausch von Profilinformationen in Netzwerkumgebungen usw.

Jedes dieser Vokabularien definiert eine Menge von Eigenschaftsnamen, die zur Zusammenstellung von Eigenschaften in Profilen verwendet werden können. Erweiterungsvokabularien können in Profilen auch kombiniert werden, wodurch die Ausdrucksmächtigkeit in CC/PP Profilen auch nachträglich erweitert werden kann.

Während die interne Repräsentation von CC/PP Profilen dem Anwendungsentwickler überlassen bleibt, ist als externes Austauschformat für Profile XML vorgesehen. Die verwendete Notation entspricht der zugrunde liegenden RDF XML Notation. Das oben dargestellte Beispiel besitzt als CC/PP Austauschformat das in Abbildung 4.35 dargestellte Aussehen.

```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:prf="http://www.w3.org/TR/WD-profile-vocabulary#">
  <rdf:Description about="HardwarePlatform">
    <prf:Defaults Screen="640x480" Mem="32MB" CPU="PPC" />
    <prf:Modifications Memory="64MB" Bluetooth="yes" />
  </rdf:Description>

  <rdf:Description about="SoftwarePlatform">
    <prf:Defaults HTMLVers="1.1" WMLVers="1.0" OSVers="2.2" />
    <prf:Modifications Images="YES" Sound="On" />
  </rdf:Description>
</rdf:RDF>
```

Abbildung 4.35: CC/PP Geräteprofil in XML Notation mit eingebetteten Parameterwerten

Da die in der Abbildung dargestellten Defaultwerte direkt in das Profil eingebettet sind, werden sie als interne Referenzen bezeichnet. Diese haben den Vorteil, dass sie direkt ausgewertet werden können, nachdem sie vom Sender zum Empfänger gelangt sind. Anderseits stellt ihre Kodierung und Übertragung eine unnötige Verwendung von Ressourcen dar, zumal dann, wenn es sich um Eigenschaften handelt, die für eine ganze Klasse von Geräten gleich sind, beispielsweise bei Voreinstellungen eines Neugerätes. In diesem Fall bietet sich die Verwendung von indirekten Referenzen an, die nicht direkt in das Profil integriert sind, sondern durch eine Referenz auf eine externe Quelle ausgedrückt werden. Solche Eigenschaften werden dann im Bedarfsfall von der referenzierten Quelle erfragt. Abbildung 4.36 zeigt das veränderte Geräteprofil, in dem die Defaultwerte von einer externen Quelle bezogen werden.

Eine weitere Möglichkeit zur Reduktion des Kommunikationsaufwands wird durch eine Kompression des XML Dokuments erreicht. Hierbei werden die Schlüsselworte des XML Dokuments durch Symbole ersetzt, die einem jeden Schlüsselwort eindeutig zugeordnet sind. In der so gewonnenen komprimierten Form kann das Dokument dann übertragen und die Symbole im Anschluss daran wieder in Schlüsselworte zurückgewandelt werden.

Bei der Übertragung von CC/PP Profilen wird davon ausgegangen, dass der Informationsaustausch zwischen Sender und Empfänger in HTTP stattfinden kann. Als Kommunikationsprotokoll wird entsprechend auch eine Erweiterung zum HTTP Extension Framework [NLL99] vorgeschlagen. Das CC/PP Exchange Protocol [OH99] definiert ein Austauschprotokoll, das zur Übermittlung von Profilen verwendet wird. Ausgehend von HTTP Protokollnachrichten werden Header spezifiziert, die entweder Kontroll– oder Statusinformationen bzw. Profildaten enthalten können. Im Einzelnen sind Profile–Header, Profile–Diff–Header und Profile–Warning–Header definiert:

Profile Header : Diese Nachricht wird als Eröffnungsnachricht gesendet. Der Profile Header enthält eine eindeutige Referenznummer und Verweise auf die nachfolgenden Profildaten bzw. Referenzen auf extern definierte

```
<?xml version="1.0"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:prf="http://www.w3.org/TR/WD-profile-vocabulary#">
  <rdf:Description about="HardwarePlatform">
    <prf:Defaults>
      <rdf:li resource="http://www.HardwareVendor.com/profiles/4711">
    </prf:Defaults>
    <prf:Modifications Memory="64MB" Bluetooth="yes" />
  </rdf:Description>

 <rdf:Description about="SoftwarePlatform">
   <prf:Defaults>
     <prf:li resource="http://www.SoftwareVendor.com/profiles/browser">
     </prf:Defaults>
     <prf:Modifications Images="YES" Sound="On" />
  </rdf:Description>
</rdf:RDF>
```

Abbildung 4.36: CC/PP Geräteprofil in XML Notation mit externen Parameterwerten

CC/PP Profile. Ein Profile Header kann Verweise auf beliebig viele nachfolgende Profile oder externe Profile enthalten.

Profile–Diff–Header : Diese Nachricht enthält die Daten eines Profils, das als Teil der Kommunikation übertragen wird und auf das in der vorausgehenden Header Nachricht verwiesen wurde. Die Daten sind in regulären XML Ausdrücken kodiert.

Profile–Warning–Header : Diese Nachricht kann als Antwort auf einen Profile–Header oder auf einen Profile–Diff–Header versendet werden. In der von HTTP bekannten Weise enthalten Warnungen eine eindeutige Meldungsnummer (z.B. 102 für „Not used profile“), einen Verweis auf den beanstandeten Inhalt (z.B. „http://www.SoftwareVendor.com/browser“) sowie den zur Meldungsnummer gehörenden Meldungstext (z.B. „not used profile“).

Abbildung 4.37 illustriert die Verwendung des CC/PP Exchange Protokolls zur Übertragung der im obigen Beispiel definierten Profile vom Gerät zum Kommunikationspartner.

Werden die Defaultwerte nicht direkt in das Profil eingebunden, sondern wird auf diese nur indirekt verwiesen, so ändern sich der Profile–Header und die Profile–Diff–Header. Im Profile–Header werden nun die externen Quellen benannt, wobei sich eine Priorisierung ergibt: Je weiter rechts eine Quelle genannt ist, desto höher ist deren Priorität. Höher priorisierte Quellen überdecken Quellen mit niedrigerer Priorität. Die Profile–Diff–Header enthalten nur noch die

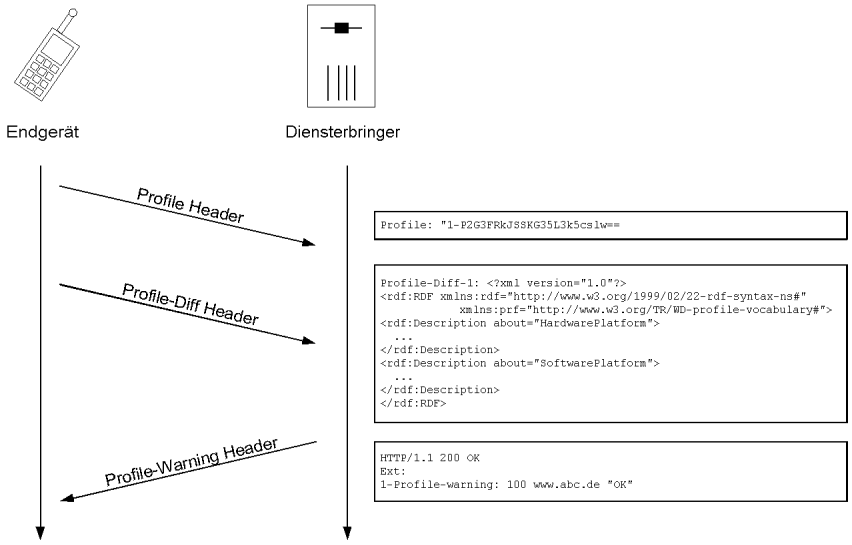


Abbildung 4.37: Interaktion zwischen Gerät und Diensterbringer nach dem CC/PP Exchange Protocol

Abweichungen von den Defaultwerten. Die geänderten Protokolldaten sind in Abbildung 4.38 dargestellt.

Da es sich bei CC/PP um reguläres XML handelt, kann es mit den üblichen Werkzeugen für XML verarbeitet werden. Mit Hilfe von Validierern kann die Gültigkeit von CC/PP Profilen sichergestellt werden, Profile können mittels SAX Parsern (Simple API for XML) untersucht werden und mit DOM Bibliotheken (Document Object Model) in eine interne Repräsentation überführt oder aus dieser zur Übertragung oder Abspeicherung in XML zurückübertragen werden.

CC/PP ist nicht für einen speziellen Anwendungsfall entwickelt worden, sondern generell zur Erfassung beliebiger Gerätemerkmale in Profilen und zu deren Übertragung zwischen Sender und Empfänger. Sollen die Aspekte eines besonderen Szenarios berücksichtigt werden und dabei die Eigenschaften einer bestimmten Geräteklasse erfasst und verarbeitet werden, so kann eine weitere Verfeinerung von CC/PP über Erweiterungsvokabularien stattfinden. UAProf (User Agent Profile) [Gro99] stellt eine solche Erweiterung für die Definition von Eigenschaften im Kontext von WAP dar. Sie eignet sich besonders zur Beschreibung von Eigenschaften mobiler Geräte, die auf entfernte Informationssysteme zugreifen, und ist daher auch zur Nutzung im Kontext dieser Arbeit einzusetzen.

4.3.6.3 UAProf

Wie CC/PP selbst dient auch UAProf zur Definition von Geräteprofilen. In Abweichung zur CC/PP Terminologie werden diese nicht als Profile, sondern als Capability and Preference Information (CPI) bezeichnet. Im Übrigen handelt es sich bei CPIs jedoch um Profildefinitionen gemäß der von CC/PP vorgegebenen Struktur, die lediglich um ein Vokabular für die speziellen Eigenschaften von

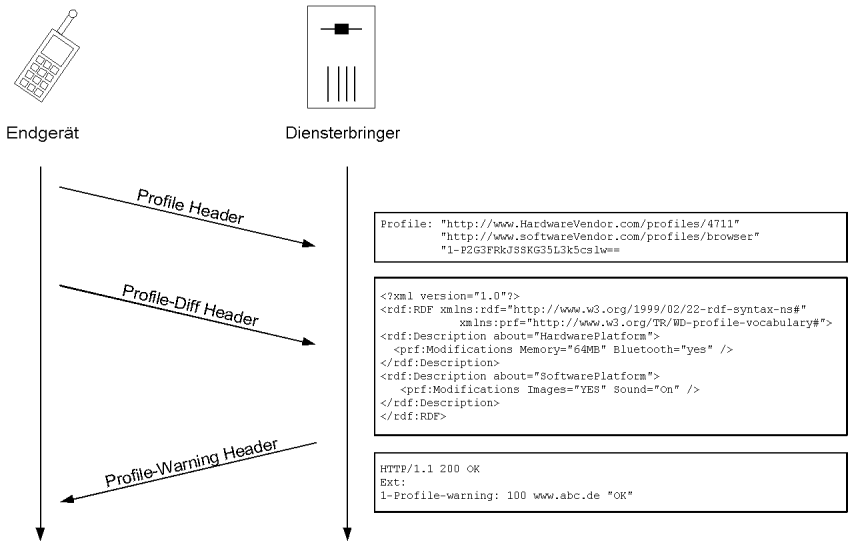


Abbildung 4.38: CC/PP Protokollnachrichten bei extern referenzierten Defaultwerten

WAP Geräten erweitert wurde. CPIs werden zwischen Sender und Empfänger ausgetauscht, wobei UAProf diese drei definierten Klassen zuordnet, nämlich Client, Intermediate Server und Origin Server. Client und Origin Server entsprechen Dienstnehmer und Diensterbringer, die über eine beliebige Anzahl von Intermediate Servern miteinander verbunden sind. Letztere dienen dabei zur gegebenenfalls notwendigen Umsetzung z.B. beim Übergang in ein anderes Kommunikationsmedium (etwa durch ein WAP Gateway) oder zur Protokollanpassung (z.B. durch einen Proxy). Mit dem für WAP obligatorischen WAP Gateway ist somit immer zumindest ein Intermediate Server vorhanden.

Wie in CC/PP werden auch in UAProf die definierten Eigenschaften einer übergeordneten Klasse zugeordnet. Zu den bereits durch CC/PP definierten Klassen „SoftwarePlatform“ sowie „HardwarePlatform“ treten dabei die folgenden Erweiterungsklassen:

BrowserUA : Klasse von Eigenschaften, welche den verwendeten Browser beschreiben.

NetworkCharacteristics : Klasse von Eigenschaften, welche die zugrunde liegende Netzwerkinfrastruktur beschreiben (Trägerdienst, Dienstgüte usw.).

WAPCharacteristics : Klasse von Eigenschaften, welche die WAP Umgebung auf dem Gerät beschreiben. Dieses umfasst neben Aspekten des eigentlichen WAP Browsers (WML Version, anzeigbare Attribute usw.) auch Aspekte der Unterstützungsfunktionen auf dem Gerät (WTLS Fähigkeit, WAP Push Fähigkeit et cetera).

Die folgende Tabelle enthält einige Eigenschaften, die als CPI in einem Profil kodiert und an das WAP Gateway übermittelt werden können:

Eigenschaft	Beschreibung	Auflösung	Mögl. Belegung
Beispiele aus der Klasse „HardwarePlatform“			
ImageCapable	Unterstützung grafischer Elemente	Locked	„Yes“
BitsPerPixel	Farbtiefe der Anzeige	Override	„2“
Model	Typenbezeichnung des Gerätes	Locked	„7110“
Beispiele aus der Klasse „SoftwarePlatform“			
OSName	Name des Betriebssystems	Locked	„WindowsCE“
VideoInput-Encoder	Liste unterstützter Videoencoder	Append	„MPEG–1“, „MPEG–2“
Beispiele aus der Klasse „NetworkCharacteristics“			
SupportedBearers	Unterstützte Trägerdienste	Locked	„GPRS“, „USSD“
Beispiele aus der Klasse „BrowserUA“			
BrowserName	Name des Browsers	Locked	MSIE
CcppAccept-Encoding	Unterstützte Kodierungen	Append	„Base64“, „quoted–printable“
FramesCapable	Unterstützung für Frames	Override	„No“
Beispiele aus der Klasse „WapCharacteristics“			
WmlVersion	Unterstützte WML Version	Append	„1.0“, „1.1“
WmlDecksize	Max. Größe für WML Seiten	Locked	„1400“

Tabelle 4.1: Beispiele für in UAProf definierte Eigenschaften

Bei den genannten Beispielen handelt es sich lediglich um einen Auszug der im Standard definierten Eigenschaften. Darüber hinaus erlaubt UAProf die Erweiterung des eigenen Vokabulars bzw. die Einführung zusätzlicher Erweiterungsvokabularien zur Unterstützung weiterer Geräte in deren spezifischen Merkmalen. Wichtig ist in diesem Fall, dass alle beteiligten Kommunikationspartner ein gemeinsames Verständnis von den unterstützten Vokabularien besitzen und die übermittelten Eigenschaften gleich interpretieren. Die hierzu notwendige semantische Grundlage wird weder durch CC/PP noch durch UAProf gelegt. Sie ist Sache der Kommunikationsknoten selbst.

Soll eine Interaktion mit einem entfernten Origin Server stattfinden, so wird zunächst eine komprimierte CPI an das zwischengeschaltete WAP Gateway gesendet, wobei die CPI Daten als Teil eines WSP (Wireless Session Protocol) Datenpakets kodiert sind. Es ist dann Aufgabe des WAP Gateways, die Daten in ein reguläres CC/PP Profil zu überführen und dieses dann mittels CC/PP Exchange Protokoll an den Origin Server bzw. den nächsten Intermediate Server weiterzuleiten, der zwischen Endgerät und Origin Server positioniert ist. In entgegengesetzter Richtung muss das WAP Gateway dann etwaige Profile–Warning–Header in entsprechende WSP Nachrichten umsetzen, die dann an das Endgerät zurückübermittelt werden können.

Ein wichtiges Merkmal von CPIs ist, dass diese nicht unverändert zwischen Endgerät und Origin Server übertragen werden müssen. So wie jeder zwischengeschaltete Intermediate Server einen Dienst erbringt und möglicherweise notwendige Anpassungen an die Fähigkeiten des Gerätes durchführt, ändern sich aus der Sicht des jeweils nächsten Intermediate Server und schließlich aus der Sicht des Origin Servers die Eigenschaften des Endgerätes. Führt beispielsweise ein Intermediate Server die Konvertierung von HTML nach WML durch, so erscheint es aus der Sicht des Origin Servers so, als ob das Endgerät in der Lage wäre, HTML Dokumente anzuzeigen. Daher kann auch das vom Endgerät übermittelte Profil nicht gleich bleiben. Kodiert dieses wahrheitsgemäß, dass es lediglich zur Anzeige von WML Dokumenten in der Lage ist, und wird dieses Profil ohne Modifikationen an den Origin Server weitergeleitet, kann dieser nur dann seinen Dienst erbringen, wenn er zur Ausgabe von WML Dokumenten fähig ist – obwohl der zwischengeschaltete Konvertierer sehr wohl eine Umsetzung vornehmen könnte.

In UAProf (wie auch in CC/PP) wird dieses Problem gelöst, indem jeder Knoten, den ein CPI durchläuft, dieses analysieren und gegebenenfalls erweitern kann. Ein Proxy, der zur Konvertierung von HTML Dokumenten nach WML in der Lage ist, würde dem CPI die Eigenschaft zur Anzeige von HTML Dokumenten hinzufügen, ein Konvertierer von WML 1.2 nach WML 1.1 die Fähigkeit zur Interpretation von WML 1.2 Dokumenten usw. Da eine Überlagerung oder Erweiterung von Eigenschaften nicht in jedem Fall gewünscht sein dürfte, werden Eigenschaften in UAProf in drei Klassen unterteilt:

Gesperrt (Locked) : Gesperrte Eigenschaften dürfen neben dem Defaultwert maximal eine zusätzliche Belegung innerhalb des CPIs besitzen. Beispiel: Die Seriennummer eines Mobiltelefons.

Überschreibbar (Override) : Überschreibbare Eigenschaften können von jedem zwischengeschalteten Intermediate Server überschrieben und entsprechend angepasst werden. Beispiel: Fähigkeit zur Anzeige von HTML Dokumenten.

Erweiterbar (Append) : Erweiterbare Eigenschaften können von jedem zwischengeschalteten Intermediate Server um einen weiteren Eintrag erweitert werden, wobei der Origin Server die Liste aller Einträge erhält. Beispiel: Liste verstandener Zeichensätze.

Das beim Origin Server eintreffende CPI kann demnach wesentlich vom ursprünglichen CPI abweichen. Der Server kann die angeforderten Informationen entsprechend des endgültigen CPIs zusammenstellen, es ist dann Sache der Intermediate Server, die Information jeweils wieder in eine Form zu bringen, die der nächste Knoten in der Kette interpretieren kann, so dass schließlich die Repräsentation entsteht, die mit den Angaben im ursprünglichen CPI konform geht und die vom Endgerät interpretiert werden kann. Die Abbildung 4.39 dokumentiert das bisher Gesagte in einer schematischen Darstellung.

Nur wenn alle Knoten zwischen Endgerät und Origin Server dazu fähig sind, Informationen über Eigenschaften in CPI oder CC/PP Profilen zu kodieren bzw. derart kodierte Informationen zu interpretieren und entsprechend zu handeln,

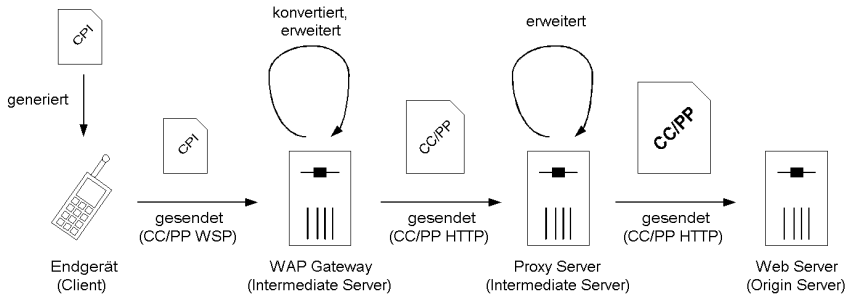


Abbildung 4.39: Profilinformationen auf dem Weg zwischen Client und Origin Server

kann die gewünschte Anpassung der Dienstgüte erfolgen. Zwar ist eine Umformung von CC/PP Profilen in einfachere, verfügbare Ansätze (z.B. HTTP Accept Header) anhand von Konvertern denkbar, da diese Ansätze jedoch eine wesentlich geringere Ausdrucksmächtigkeit besitzen, stellt ein solches Vorgehen allenfalls ein beschränktes Hilfskonstrukt dar. An dieser Stelle muss daher daraufhin gearbeitet werden, möglichst alle Diensterbringer dahingehend zu erweitern, dass sie zur Verarbeitung von Profilinformationen in der Lage sind.

Ein weiteres Problem stellt die Unterstützung von UAProf durch mobile Endgeräte dar. Dies betrifft einerseits bereits in Benutzung befindliche Geräte, bei denen eine entsprechende Fähigkeit fehlt. Andererseits betrifft es Geräte, bei denen eine unidirektionale Kommunikation in Richtung des Gerätes stattfindet (z.B. Pager, Aktoren usw.). Auch hier kann das Gerät selbst seine Eigenschaften nicht an den Server übermitteln, um eine Anpassung der zu übermittelnden Daten zu veranlassen. In beiden Fällen muss eine zwischengeschaltete Instanz die Generierung und Weiterleitung von Profilen übernehmen.

Es ist nahe liegend, eine derartige Funktionalität in das WAP Gateway zu integrieren. Das WAP Gateway muss hierzu für jedes ihm bekannte Gerät ohne UAProf Fähigkeit ein vorgefertigtes Profil bereithalten. Trifft nun eine Anfrage von einem Gerät ein und enthält diese Anfrage keine gültigen CPI Daten, so wird ermittelt, welcher Klasse das anfragende Gerät angehört und die Anfrage um die vorgefertigten Profilinformationen erweitert, bevor sie an den nächsten Knoten weitergeleitet wird. Eine andere Variante besteht darin, ganze Profilverzeichnisse aufzubauen, in denen die vorgefertigten Profile registriert werden. Diese können dann bei veralteten Geräten die selbst gefertigten Profile ersetzen und bei neueren Profilen zur Verwaltung von Defaultwerten genutzt werden und so für eine effizientere Kommunikation zwischen Endgerät und den weiteren Knoten sorgen. Abbildung 4.40 skizziert beide Varianten.

Zur optimierten Generierung von Inhalten im Kontext von WAP (und allgemeiner auch im Kontext des Worldwide Web) sind also folgende Bedingungen durch die beteiligten Diensterbringer und Dienstnehmer zu erfüllen:

1. Endgeräte sollten Profile generieren, speichern und im Zuge ihrer Kommunikation mit Diensterbringern weiterleiten können.

2. Gateways sollten Profile interpretieren, ggf. erweitern und diese dann in

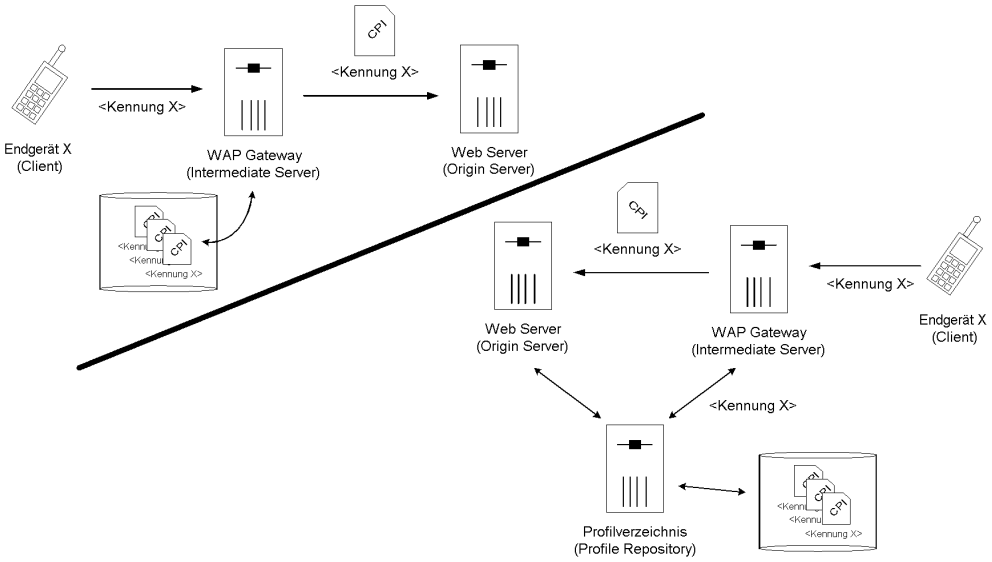


Abbildung 4.40: Vorgefertigte CPIs können im Gateway oder einem dedizierten Profilverzeichnis lagern

Richtung des Diensterbringers weiterleiten können.

3. Zentrale Profilverzeichnisse (Repositories) sollten vorgefertigte Profile zwischenspeichern und an Gateways, Intermediate Server und Origin Server übermitteln können.

4. Intermediate Server sollten Profilinformationen interpretieren, entsprechend ihrer Fähigkeiten anpassen und dann in Richtung des Diensterbringers weiterleiten können.

5. Origin Server sollten Profile interpretieren und den angebotenen Dienst entsprechend der aus dem Profil ersichtlichen Anforderungen anpassen können.

Zur Übermittlung von Profilen ist es darüber hinaus notwendig, dass alle beteiligten Knoten das CC/PP Exchange Protocol beherrschen, entweder in der direkt auf HTTPext basierenden Form oder in der speziellen Variante für WAP.

4.3.7 Geräteprofil– und Geräteklassenkonzepte im Vergleich

Die Nutzung von Geräteprofilen und Geräteklassen als wesentlicher Baustein einer optimalen Unterstützung mobiler Geräte in einem verteilten Anwendungssystem ist in der Literatur bislang wenig dokumentiert. Während die Eigenschaften einzubeziehender Geräte einen elementaren Einfluss auf den zu erbringenden Dienst besitzen und inhärent allen Integrationsumgebungen zugrunde liegen, werden diese zumeist nicht formal in Datenstrukturen zusammengefasst.

Insofern überwiegt die direkte Auswertung von Merkmalen im Rahmen der dynamischen Strategiewahl gegenüber Ansätzen, bei denen die verfügbaren Strategien vorab feststehen.

Der Begriff des Profils ist im Bereich mobiler Systeme schon seit längerem geprägt, wobei er überwiegend in anderem Zusammenhang verwendet wird. So stellen Imielinski und Badrinath in [BI93] das Konzept des Mobility Profile vor. Dieses beschreibt nicht das mobile Gerät selbst, sondern dessen Nutzung. Anhand eines Bewegungsprofils soll die Verwendung des Gerätes dokumentiert und so eine optimale Verteilung von Daten ermöglicht werden, auf die vom Gerät zugegriffen wird.

Rakotonirainy und Bond stellen in [RB97] das Konzept der *Characteristics Of The Environment* (COE) vor. COEs beschreiben die Merkmale eines jeden Objektes in einem verteilten Anwendungssystem. Sie beinhalten Konfigurationsparameter, Bildschirmgröße et cetera, die in Tupeln zusammengefasst werden und so einen mehrdimensionalen Definitionsraum ergeben. Auf der Basis von COEs können einfache Hierarchien definiert werden, indem die Tupel gemäß ihrer Anforderungen an die Umgebung auf einer Skala abgebildet werden. Der Hierarchiebegriff bei Rakotonirainy und Bond ist nicht absolut, da die Skala bzw. die Bewertung der COEs bezüglich einer solchen Skala vom Anwendungskontext abhängt und die beteiligten Kommunikationspartner sich a priori auf ein gemeinsames Wertesystem festlegen müssen. Die mittels einer gemeinsamen Hierarchie fixierte Ordnung von COEs kann schließlich zur Aushandlung von Dienstgütemerkmalen verwendet werden, die bei Rakotonirainy und Bond in einer eigenen Beschreibungssprache (Architecture Description Language, ADL) formal niedergelegt werden. Während COEs eine gewisse Parallele zu den in dieser Arbeit beschriebenen Geräteprofilen und Geräteklassen besitzen, liegt ein wesentlicher Unterschied in der sich auf ihrer Basis ergebenden Hierarchie. Diese ist im Fall der COEs zunächst eindimensional und bedingt vor allem die explizite Abbildung von COEs in dem aufgestellten Wertesystem. Ein solcher Schritt ist bei den Geräteprofilen in dieser Arbeit nicht notwendig. Die Zuordnung eines Geräteprofils zu einer bestimmten Position innerhalb der Hierarchie, vertreten durch eine Geräteklasse, kann auf der Basis der vorhandenen Merkmale automatisch geschehen.

Rodden et al. erläutern in [RCDD98] eine sehr allgemeine Taxonomie für mobile Geräte. Sie benutzen diese jedoch nicht konkret zum Entwurf eines Unterstützungssystems, sondern lediglich zur Herausarbeitung von Merkmalen, die im Zuge eines Entwurfsprozesses für eine Integrationsumgebung zu beachten sind. Die von den Autoren vorgestellte Taxonomie stellt mithin kein formales Wertesystem dar, es dient vielmehr zur Motivation folgender Arbeiten. Dennoch könnte es als Grundlage für Geräteklassen und eine hierauf aufbauende Hierarchie herhalten.

Perkins regt in [Per98] die Verwendung eines *Profile Managers* im Zusammenhang mit der Realisierung weitgehender Ortstransparenz von Mobilgeräten mittels MobileIP an. Dem Profile Manager soll nach Perkins die Aufgabe zukommen, die Eigenschaften und Anforderungen von Geräten (bzw. Kommunikationsknoten auf der Ebene der TCP/IP Kommunikation) zusammenzufassen, um so eine optimale Unterstützung durch die Kommunikationsinfrastruktur zu ermöglichen. Während die Komponente in [Per98] nicht im Detail beschrieben ist, lässt sich

vermuten, dass auch in diesem Fall die Definition von Geräteprofilen und deren Übermittlung an die Dienste der Infrastruktur zur Auswahl geeigneter Strategien angedacht ist.

Jones und Woundy schlagen in [JW01] Geräteklassen als zusätzliches Merkmal des Dynamic Host Configuration Protocol (DHCP) vor. Die Geräteklassen sollen als Teil der DHCP Agent Optionsfelder verwendet werden, um DHCP Diensten die Auswahl einer passenden Strategie zu ermöglichen, wobei sie auf die in den entsprechenden Optionsfeldern kodierten Informationen über die Merkmale des Gerätes zurückgreifen. Die von Jones und Woundry vorgeschlagenen Klassen können Informationen zu Hardwarefähigkeiten, Softwarefähigkeiten und Präferenzen umfassen.

Nicht weiter vertieft wurde schließlich die Verwendung von Feature Structures zur Konstruktion von Klassifikationssystemen und zu deren Nutzung im Sinne einer optimierten Strategiewahl. Mit Hilfe dieses in [Wie01] beschriebenen Konzeptes könnte es möglich werden, ohne vorherige Definition starrer Hierarchien aus Geräteklassen eine Zuordnung von Geräteprofilen zu Geräteklassen zu erlangen und so die sichere Auswahl von an Geräteklassen gebundenen Adaptionsstrategien zu erzielen.

In diesem Abschnitt wurden Konzepte und Realisierungstechniken zur Definition von Geräteprofilen und Geräteklassen besprochen, welche gemeinsam die Grundlage für eine Auswahl optimaler Strategien zur Einbindung von mobilen Geräten in verteilte Anwendungssysteme bilden. Es wurde in diesem Zusammenhang gezeigt, wie Eigenschaften zu Geräteprofilen zusammengefasst werden und in welcher Beziehung Geräteprofile ihrerseits mit Geräteklassen stehen. Es wurde ferner auf verschiedene Ansätze eingegangen, nach denen Geräteklassen miteinander in Beziehung gesetzt werden können und schließlich angerissen, wie die hierbei entstehenden Klassenhierarchien genutzt werden können, um die Auswahl von Integrationsstrategien zu vereinfachen. Es wurde deutlich gemacht, dass es keinen eindeutigen Strategiebegriff gibt, sondern die Strategie – ebenso wie Form und Inhalt der Profile und Klassen – vom jeweiligen Anwendungszusammenhang abhängt. Sofern es gelingt, von einem Gerät, das durch bestimmte Eigenschaften beschrieben wird, auf eine Geräteklasse zu schließen, die in sich einen Verweis auf eine Strategie besitzt, ist das weitere Vorgehen nicht von vornherein festgelegt, sondern hängt davon ab, welcher Strategiebegriff im Integrationskontext gewählt wurde. Als Beispiel wurde der Strategiebegriff herangezogen, welcher dem Integrationskonzept zugrunde liegt, das in dieser Arbeit konzipiert und realisiert wurde. Hier werden abstrakte Informationen anhand von Regelsätzen in konkrete Repräsentationen transformiert. Entsprechend enthalten Geräteklassen einen Verweis auf den für das jeweilige Gerät gültigen Regelsatz. Schließlich wurden Ansätze vorgestellt, die als Realisierungsgrundlage dienen können. Angefangen beim sehr generischen *Resource Description Framework* über dessen Spezialisierung in *Composite Capability/Preference Profiles* bis hin zu den aus dem WAP Bereich stammenden *User Agent Profiles* wurde anhand von Beispielen die Nutzung der jeweiligen Technik gestreift.

Geräteprofile und –klassen dienen zur Auswahl von Integrationsstrategien. Im Kontext dieser Arbeit werden mit ihrer Hilfe Transformationsregelsätze ausgewählt, anhand derer abstrakte Beschreibungen von Benutzungsschnittstellen in konkrete Repräsentationen übersetzt werden, die sodann auf dem jeweiligen

Gerät dargestellt werden können und so die Nutzung von Anwendungen auf dem Endgerät ermöglichen. Um eine solche Nutzung nicht nur einem bestimmten Gerät, sondern beliebigen Geräten zu ermöglichen, und um von einem bestimmten Gerät nicht nur eine einzige Anwendung, sondern hierauf dann mehrere Anwendungen nutzen zu können, bedarf es eines Koordinationsmechanismus. Den hierzu benötigten Konzepten und Realisierungstechniken widmet sich der nächste Abschnitt.

4.3.8 Sitzungssteuerung auf Anwendungsebene

Die vorangegangenen Abschnitte dienten der Beschreibung aller wesentlichen Konzepte, die für eine Erfassung und Übertragung von Momentaufnahmen der grafischen Benutzungsschnittstelle einer Anwendung auf ein entferntes Endgerät notwendig sind. Beginnend bei der Analyse der Schnittstelle durch eine entsprechende Analysekomponente über die Generierung einer abstrakten Zwischenrepräsentation, über deren Transformation in ein konkretes Zielformat unter Berücksichtigung der Merkmale des zu bedienenden Endgeräts, bis hin zur Weiterleitung der Zielrepräsentation wurde die gesamte Wegstrecke zwischen Ausgangspunkt und Ziel beleuchtet. Auch die in entgegengesetzter Richtung notwendige Aufnahme von Aktionen auf dem Endgerät und deren Weiterleitung hin zur Ereigniswarteschlange der Anwendung wurde hinsichtlich ihrer konzeptionellen Anforderungen betrachtet. An dieser Stelle ist es lediglich die Koordination der einzelnen Schritte, deren Erläuterung aussteht: Die Beschreibung des Steuerungsmechanismus also, der die Reihenfolge der notwendigen Schritte herstellt und die Aktivierung der Teilkomponenten der Integrationsumgebung in eben dieser Reihenfolge sicherstellt. In diesem Abschnitt soll die Sitzungssteuerung behandelt werden, die genau diesem Zweck dient.

Die grundsätzliche Aufgabe der Sitzungssteuerung ist im vorigen Absatz bereits angeklungen: Sie dient der Verzahnung der Integrationsschritte von Analyse, Transformation, Übertragung von Informationen sowie Aufnahme von Rückmeldungen, wie sie in den vorigen Abschnitten beschrieben wurden. Im Zusammenhang mit dieser grundsätzlichen Koordinierungsaufgabe ergeben sich darüber hinaus folgende weitere Aufgaben:

Authentifizierung / Autorisierung : Es ist Sache einer Sitzungssteuerung zu entscheiden, wer in eine Interaktion mit der Integrationsumgebung eintreten darf. Die Sitzungssteuerung muss das Endgerät bzw. dessen Benutzer identifizieren und dessen Berechtigungen zur Nutzung einzelner Funktionen der Umgebung erkennen können.

Verwaltung von Benutzerprofilen : Um Berechtigungen bestimmten Benutzern zuordnen zu können, ist es notwendig, Benutzerprofile anzulegen und diese zu organisieren. Die Verwaltung von Benutzerprofilen ist der Sitzungssteuerung zuzuschlagen.

Behandlung eigenständiger Geräte : Insbesondere im Zusammenhang mit der Integration mobiler Geräte ist davon auszugehen, dass diese bereits über ein eigenes Sitzungskonzept (d.h. Authentifizierung bzw. Autorisierung) verfügen. Eine Sitzungssteuerung muss dieses gegebenenfalls in Betracht ziehen und unter Umständen in die Verarbeitung einbeziehen.

Behandlung der Fehleranfälligkeit : Die Unterstützung mobiler Geräte erfordert ein hohes Maß an Berücksichtigung möglicher Fehlerfälle wie Verbindungsabbrüche, häufige Änderungen der Verbindungsqualität et cetera, die sich auf den Verlauf der entfernten Anwendungsnutzung auswirken können. Die Sitzungssteuerung muss bereits auf konzeptioneller Ebene in der Handhabung solcher Fehlerfälle ansetzen.

Behandlung von Gerätewechseln : Mobile Geräte weisen ein sehr begrenztes Nutzungsspektrum auf, so dass der häufige Wechsel zwischen einzelnen Geräten als sehr wahrscheinlich zu erachten ist. Die Sitzungssteuerung muss entsprechend unterstützend wirken, um die Anwendungsnutzung auch über den jeweiligen Wechsel hinaus zu ermöglichen.

Insbesondere aus letztgenanntem Punkt ergeben sich eine Reihe weiterer Forderungen an eine zu entwerfende Sitzungssteuerung wie die der Notwendigkeit zur Entkopplung von Anwendungslauf und entfernter Anwendungsnutzung, zur temporären Persistierung von Anwendungen sowie zur Realisierung eines plattformneutralen Sitzungssteuerungskonzepts.

Bevor nun darauf eingegangen wird, wie die Sitzungssteuerung diese Forderungen erfüllen kann, soll zunächst der Sitzungsbegriff selbst behandelt werden. Wie lässt sich dieser fassen? Anhand eines Beispiels soll dieser Frage näher gekommen werden. Will der Besitzer eines PDAs über die Integrationsumgebung auf eine entfernte Anwendung zugreifen, so muss er zunächst eine Kommunikationsverbindung herstellen und über diese den externen Kommunikationsadapter ansprechen. Der Adapter generiert einen Anmeldedialog und nach erfolgreicher Authentifizierung steht dem Benutzer eine Liste möglicher Anwendung zur Verfügung, die er laut Benutzungsprofil über den PDA entfernt verwenden darf. Nach Auswahl der gewünschten Anwendung wird diese mitsamt korrespondierendem Application Shadow gestartet, und der Analyse– und Transformationszyklus beginnt in der bereits geschilderten Weise. Will der Benutzer nun vom verwendeten PDA zu seinem Laptop überwechseln und dabei weiterhin die zuvor ausgewählte Anwendung benutzen, so muss er diese zunächst kurzzeitig stoppen und sich von der Integrationsumgebung abmelden. Vom Laptop aus kann er sodann wiederum eine Verbindung zum ECA aufbauen und sich erneut bei der Integrationsumgebung anmelden. Neben den zum Start verfügbaren Anwendungen wird ihm nun zusätzlich die bereits laufende Anwendung angeboten, deren Benutzung er durch eine entsprechende Auswahl erneut aufnehmen kann. Will er während der Nutzung der Anwendung eine zweite Anwendung starten, so kann er diese aus der im Profil gespeicherten Liste auswählen und nach deren Start zwischen den nunmehr zwei parallel ablaufenden Anwendungen wechseln. Wird eine Anwendung nicht länger gebraucht, so kann sie durch einen entsprechenden Menüpunkt beendet werden. Nachdem die letzte laufende Anwendung so gestoppt wurde, führt die Integrationsumgebung den Benutzer automatisch wieder zum Anmeldeschirm des ECAs zurück.

Allgemein gesprochen dient eine *Sitzung* der logischen Verbindung aufeinander folgender Handlungen in einem gemeinsamen Kontext. Im Zusammenhang dieser Arbeit wird der Sitzungsbegriff dabei oftmals zur Verdeutlichung zeitlicher Zusammenhänge verwendet: Als Sitzung wird entsprechend die Zeitspanne betrachtet, in der das Kriterium erfüllt ist, anhand welchem die Handlungen miteinander in Beziehung gesetzt werden.

Betrachtet man das geschilderte Beispiel hinsichtlich der erkennbaren Sitzungsformen, so wird deutlich, dass hier drei verschiedene Sitzungstypen identifizierbar sind:

Gerätesitzung : Als solche lässt sich die Zeitspanne definieren, während der ein Gerät kontinuierlich mit der Integrationsumgebung verbunden ist und während der das Gerät eindeutig identifiziert werden kann.

Benutzersitzung : Analog zur Gerätesitzung wird als Benutzersitzung die Zeitspanne bezeichnet, in der ein eindeutig identifizierbarer Benutzer die Ressourcen der Integrationsumgebung in Anspruch nimmt.

Anwendungssitzung : Als Anwendungssitzung wird schließlich der Zeitraum bezeichnet, während dem eine bestimmte Anwendung durch die Integrationsinfrastruktur hindurch verwendet wird. Die Anwendungssitzung ist einer bestimmten Anwendungsinstanz sowie einer bestimmten Benutzersitzung zugeordnet.

Stellt der Benutzer eine Kommunikationsverbindung zum ECA her, so wird zunächst eine Gerätesitzung instantiiert. Das Gerät wird identifiziert und der auf die spezifischen Fähigkeiten des Gerätes abgestimmte Nachrichtenkanal wird aufgebaut. Es wird sodann ein Anmeldebildschirm generiert und dieser über den zuvor eingerichteten Nachrichtenkanal zum Endgerät geleitet, wo er dem Benutzer präsentiert werden kann. Nach erfolgreicher Anmeldung gelangt der Benutzer nun in eine Benutzersitzung, in welcher er bis zu seiner Abmeldung verbleibt. Aus der Benutzersitzung heraus wird nun eine Auswahl verfügbarer Anwendungen gezeigt und nach Start der ersten Anwendung wird der Benutzersitzung eine Anwendungssitzung zugeordnet, welche die gestartete Anwendungsinstanz repräsentiert.

Durch temporäres Stoppen laufender Anwendungen wird der Wechsel zu einem anderen Endgerät eingeleitet. Hierbei wird durch Abmeldung vom System die Verbindung zwischen Benutzersitzung und Gerätesitzung gelöst und die zuvor aktuelle Gerätesitzung verworfen. Durch erneuten Aufbau einer Kommunikationsverbindung von einem anderen Gerät kann sodann eine neue Gerätesitzung hergestellt werden. Durch die erneute Anmeldung am System wird die neu generierte Gerätesitzung mit der zuvor entkoppelten Benutzersitzung verbunden, und die gestoppte Anwendungssitzung kann wieder aufgenommen werden. Durch den Start der zweiten Anwendung wird eine weitere Anwendungssitzung erstellt und mit der Benutzersitzung verkettet und in der Folge kann dann zwischen den Anwendungssitzungen hin und her gewechselt werden. Bei Beendigung der Anwendungen werden die entsprechenden Anwendungssitzungen verworfen, so dass schließlich nach dem Ende der letzten Sitzung die Sitzungssteuerung zum Start neuer Anwendungen auffordert. Abbildung 4.41 stellt dar, wie die Sitzungstypen sich den unterschiedlichen Bereichen des Integrationsvorgangs zuordnen lassen. Die Gerätesitzung existiert nur zwischen Endgerät und Integrationsumgebung. Sie markiert quasi den Kommunikationskanal zwischen diesen beiden Endpunkten. Die Anwendungssitzung wird hingegen zwischen Integrationsumgebung und Anwendung aufgespannt. Sie markiert die Lebenszeit einer Anwendungsinstanz im Kontext der Nutzung durch einen bestimmten Benutzer. Die Benutzersitzung schließlich markiert die Nutzung der Integrationsumgebung durch einen

bestimmten Benutzer. Sie existiert demgemäß zwischen Benutzer und Integrationsumgebung.

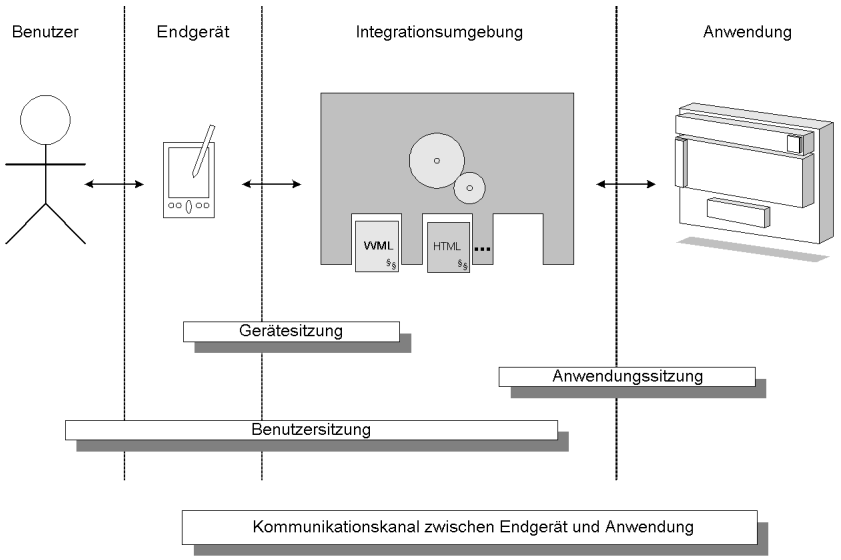


Abbildung 4.41: Die Anordnung der drei Sitzungstypen im Bezug auf die Integrationsumgebung

Wie aus der Abbildung des Weiteren deutlich wird, lassen sich die drei Sitzungstypen verschiedenen Bereichen des in vorigen Abschnitten beschriebenen logischen Kommunikationskanals zwischen Endgerät und Anwendung zuordnen. Die Benutzersitzung erstreckt sich über den dem Gerät zugewandten Teil des Kanals und reicht bis in den Kern der Integrationsumgebung hinein, wobei sie nicht mit der eigentlichen Anwendung zu tun hat. Die Gerätesitzung ist hingegen nur zwischen Externem Kommunikationsadapter und Umgebungskern definiert, während die Anwendungssitzung schließlich eine gewisse Überlappung mit der Benutzersitzung hat und den Abschluss des Kanals zur Anwendung hin bildet.

Über die Eigenschaften der drei vorgestellten Sitzungstypen lassen sich nun folgende Aussagen treffen:

- Es kann für einen Benutzer zu jeder Zeit nur eine Gerätesitzung aktiv sein. Einer jeden Benutzersitzung kann nur maximal eine Gerätesitzung zugeordnet sein. Andererseits ist es (zumindest kurzzeitig) möglich, dass eine Gerätesitzung ohne die Verbindung mit einer Benutzersitzung existiert. Dieses ist genau für die Zeitspanne der Fall, nachdem der Benutzer mit dem Gerät eine Kommunikationsverbindung zum ECA aufgebaut hat und bevor er sich am System erfolgreich angemeldet hat.

- Die Benutzersitzung kann nur im Kontext einer Gerätesitzung initialisiert werden. Nur wenn der Benutzer mit dem System über einen Kommunikationskanal verbunden ist, kann er sich anmelden und nur durch Anmeldung wird eine Benutzersitzung instantiiert. Andererseits ist es in der Folge durchaus möglich, dass die Benutzersitzung ohne eine korrespondierende

Gerätesitzung weiter existiert, beispielsweise wenn der Benutzer die Sitzung kurzzeitig unterbricht, um zu einem anderen Gerät überzuwechseln.

- Anwendungssitzungen sind ausschließlich im Kontext einer Benutzersitzung gültig. Ähnlich wie bei einem Mehrbenutzersystem ist auch in der hier entworfenen Integrationsumgebung nur ein angemeldeter Benutzer imstande, Anwendungen auszuwählen und zu starten. Übertragen auf ein Mehrbenutzersystem stellt die Menge der Anwendungssitzungen die „Taskliste“ der Integrationsumgebung für den Benutzer dar. Andererseits sind Anwendungssitzungen nicht mit Gerätesitzungen verknüpft. Während der Benutzer zwischen zwei Geräten wechselt und die Benutzersitzung entsprechend gestoppt ist, ruhen auch die Anwendungssitzungen, wobei sie jedoch innerhalb der Benutzersitzung weiter bestehen bleiben.

Die Aufgabe der Sitzungssteuerung ist es nun, den Auf– und Abbau von Instanzen der drei Sitzungstypen zu steuern und deren Verbindung miteinander zu koordinieren und zu überwachen. Als primäres Ziel ist hierbei anzusehen, eine möglichst robuste und flexible Ablaufumgebung für die Nutzung einer oder mehrerer Anwendungen durch die Integrationsumgebung hindurch zu schaffen. Die Gerätesitzung wird dabei vornehmlich durch den Externen Kommunikationsadapter verwaltet, da sie sehr stark von den Eigenschaften und Fähigkeiten des Gerätes bzw. des zugrunde liegenden Kommunikationskanals abhängt, während Benutzer– sowie Anwendungssitzung nur in geringem Maße von Plattformabhängigkeiten betroffen sind. Diese Sitzungstypen werden von einer zentralen Steuerungskomponente kontrolliert.

Welche konkreten Anforderungen sind nun hinsichtlich der Sitzungssteuerung an Steuerungskomponente und ECA zu stellen? Um die Identität von Benutzern und deren Berechtigungen zu erkennen (Authentisierung / Autorisierung), muss zunächst eine Benutzerverwaltung realisiert werden, die in der Lage ist, Kennungen und Zugangsschlüssel der Nutzer der Integrationsumgebung zu verwalten und so den Zugang zum System zu kontrollieren. Da nicht nur die Steuerungskomponente selbst, sondern auch der ECA auf die Benutzerdaten zugreifen können muss, ist es dabei sinnvoll die Benutzerverwaltung in einer eigenen Unterkomponente zusammenzufassen. Die Benutzerverwaltung muss demgemäß eine Schnittstelle anbieten, über welche sie mit der Steuerungskomponente und den ECAs interagieren kann, damit diese auf dort vorgehaltene Daten zugreifen können.

Der Zugriff auf die Integrationsumgebung erfolgt grundsätzlich über einen der im System bekannten ECAs. Eine wichtige Anforderung an die Sitzungsverwaltung besteht entsprechend darin, die von verschiedenen Endgeräten eingehenden Befehle aufzunehmen, sie den jeweils zuständigen Benutzersitzungen zuzuordnen und schließlich an die richtige Anwendungssitzung auszuliefern. Laufende Sitzungen müssen beendet bzw. vorläufig gestoppt und wieder aufgenommen werden können. Da die dafür notwendigen Menüpunkte nicht zur ursprünglichen Funktionalität der Anwendung gehören und diese somit nicht in der Benutzungsschnittstelle der Anwendung zu finden sind, müssen sie in den Strom der vom Application Shadow kommenden JSML Dokumente eingefügt werden können, bevor eine konkrete Repräsentation generiert wird. Die Sitzungssteuerung muss also auch mit der Analysekomponente und dem Transformator kom-

munizieren, um den richtigen Zeitpunkt wählen zu können, zu dem der Eingriff in den Analyseprozess möglich ist.

Nach einem Kommunikationsabriss muss die Sitzungssteuerung in der Lage sein, Benutzersitzung und nicht länger verfügbare Gerätesitzung voneinander zu lösen und der Benutzersitzung eine neue Gerätesitzung zuzuordnen. Hierzu muss die Steuerung erkennen können, ob der erneut angemeldete Benutzer eine noch bestehende Benutzersitzung besitzt und diese in einem solchen Fall wieder aktivieren. Diese Fähigkeit der Sitzungssteuerung setzt allerdings gleichzeitig voraus, dass der Austausch einer Benutzersitzung gegen eine andere auch zur Laufzeit möglich sein muss. Die Identifikation eines Benutzers kann nämlich nur nach dessen Anmeldung erfolgen. Diese wiederum erfolgt jedoch immer in einer für die Anmeldung erzeugten Benutzersitzung, die verworfen und ausgetauscht werden muss, sofern das System erkennt, dass für den angemeldeten Benutzer bereits eine Benutzersitzung existiert.

Wird eine unterbrochene Sitzung andererseits nicht wieder aufgenommen, sondern entschließt sich der Benutzer dazu, in der neu erzeugten Benutzersitzung weiterzuarbeiten, so muss es möglich sein, die vorige Benutzersitzung zu beenden. Da dieses nur unter der Bedingung möglich ist, dass keine weiteren Anwendungen in der Sitzung vorhanden sind, muss es der Sitzungssteuerung demgemäß auch möglich sein, Anwendungssitzungen nach einer bestimmten Zeit der Inaktivität zu beenden. Um einen konsistenten Systemzustand erhalten zu können, ist es notwendig, dass die Beendigung über die von der Anwendung bzw. dem GUI–System hierfür vorgesehenen Wege geschieht. Es ist entsprechend dafür zu sorgen, dass die Sitzungssteuerung aktiv auf den Application Shadow zugreifen und dessen Ereignissteuerung beeinflussen kann.

Wie aus der Aufzählung deutlich wird, muss die Sitzungssteuerung mit allen bisher besprochenen funktionalen Komponenten der Integrationsumgebung in Kontakt stehen und sie in ihrer Interaktion untereinander koordinieren bzw. „verzahnen“. Die Synchronisation erfolgt dabei anhand eines „5–Phasen–Modells“, das den Analyse–, Transformations– und Übertragungszyklus in einzelne Teilschritte aufspaltet und das nunmehr beschrieben werden soll.

Das Zusammenspiel zwischen Endgerät, Integrationsumgebung und Anwendung wurde bereits in vorangegangenen Abschnitten beschrieben. Zwischen Aufbau und Abbau der Kommunikationsverbindung ergibt sich ein zyklischer Prozess, in dem jeweils die Anfrage des Benutzers eine Folge von Teilschritten, die so genannten Phasen, auslöst, die im Rahmen eines jeden Bearbeitungszyklus sequentiell abgearbeitet werden (vgl. Abbildung 4.42):

Phase 1 – Eingang der Anfrage : Die erste Phase des Bearbeitungszyklus wird durch den Eingang der Anfrage über einen ECA eingeleitet. Aufgabe dieser Phase ist es, die in der Anfrage kodierten Daten entgegenzunehmen, diese gegebenenfalls in ein generisches Format zu überführen und sie so für die weitere Verarbeitung in der Integrationsumgebung vorzubereiten.

Phase 2 – Authentifizierung und Autorisierung : Im zweiten Teilschritt muss die Anfrage dann einer Benutzersitzung zugeordnet werden und festgestellt werden, ob es sich um eine berechtigte Interaktion mit der Integrationsumgebung handelt. Nur durch die Zuordnung zu einer Benutzersit-

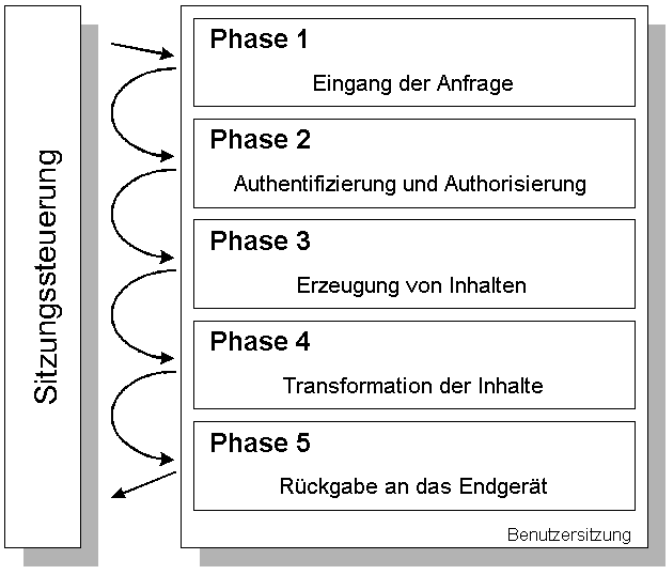


Abbildung 4.42: Aufteilung einer Anfrage in fünf Phasen

zung kann sichergestellt werden, dass die Anfrage der richtigen Anwendung zugestellt wird.

Die Authentifizierung bzw. Autorisierung muss in jedem Zyklus erfolgen, da auch solche Kommunikationskanäle unterstützt werden müssen, die über kein eigenes Sitzungskonzept verfügen und die entsprechend jeden Zugriff unabhängig von vorangegangenen Zugriffen behandeln. Um Zugriffe einem solchen Fall einer bestimmten Benutzersitzung zuordnen zu können, muss jedes Mal durch Analyse der übertragenen Daten festgestellt werden, von welchem Gerät bzw. über welchen Kanal die Anfrage eingegangen ist. Eine tatsächliche Anmeldung muss dabei nur beim ersten Durchlaufen der Phase erfolgen. In allen weiteren Durchläufen wird dann lediglich die bei der Anmeldung vergebene Sitzungsnummer verwendet, um die Zuordnung durchzuführen.

Phase 3 – Ermitteln der Ergebnisdaten : Nachdem die Anfrage erkannt und einer Benutzersitzung zugeordnet ist, kann eine Ermittlung der Ergebnisdaten stattfinden. Hierzu wird entweder die betroffene Anwendungssitzung nach einer Momentaufnahme der verbundenen Benutzungsschnittstelle befragt oder von der Sitzungssteuerung selbst eine solche synthetisiert. Letzteres ist beispielsweise notwendig, wenn die Anmeldung durch den Anwender fehlgeschlagen ist oder die Kontaktaufnahme mit der Anwendung unmöglich war. In beiden Fällen muss eine entsprechende Fehlermeldung an den Anwender zurückübertragen werden.

Phase 4 – Transformation der Ergebnisdaten : Die aus der vorigen Phase hervorgegangenen Daten werden nunmehr in ein Format transformiert, das über den Kommunikationskanal zum Endgerät übertragen werden und dort dargestellt werden kann. Bei der Transformation wird auf die

in der Gerätesitzung abgelegten Informationen über die gewünschte Zielrepräsentation zurückgegriffen und entsprechend des dort gespeicherten Geräteprofils gehandelt.

Phase 5 – Rückgabe an den Klienten : Zum Abschluss eines Bearbeitungszyklus werden die transformierten Daten zum Endgerät zurückübertragen. Die Sitzungssteuerung erkennt hierbei anhand der Gerätesitzung, über welchen Kommunikationskanal die Daten geleitet werden müssen. Analog zu der in der ersten Phase durchgeführten Übertragung der Anfrage in ein generisches Format findet nunmehr eine entgegengesetzte Übersetzung statt, bei der die bereits im Zielformat vorliegenden Ergebnisdaten in der für den Kommunikationskanal spezifischen Weise an das Endgerät übermittelt werden.

Diese Folge von Phasen wird über die Lebenszeit einer Benutzersitzung fortwährend durchlaufen, wobei die Sitzungssteuerung für jede laufende Sitzung darüber Buch führt, in welcher Phase sich diese befindet, und so den jeweils nachfolgenden Schritt bestimmen und einleiten kann. Die Sitzungssteuerung kann quasi als Motor verstanden werden, der die Benutzersitzungen kontrolliert und diese in ihrem Fortgang antreibt.

Die den einzelnen Phasen zuzuordnenden Handlungen und die Teilkomponenten, die für die Ausführung dieser Handlungen verantwortlich sind, wurden bereits beschrieben. An dieser Stelle soll daher nur die Beziehung zwischen beiden hergestellt werden. Die erste Phase wird durch die unterschiedlichen Externen Kommunikationsadapter bestritten. Diese dienen der Realisierung von Kommunikationskanälen für die zu unterstützenden Endgeräte und stellen demgemäß die Teilkomponente der Integrationsumgebung dar, welche zur Annahme von Anfragen verwendet wird. Die zweite Phase wird zum Teil durch die zentrale Steuerungskomponente selbst ausgefüllt. Die Tatsache, dass jede Anfrage durch diese überprüft und hinsichtlich ihrer Gültigkeit bewertet wird, hat letztlich zur Benennung der Steuerungskomponente als *Gateway* geführt: Nur wenn eine Anfrage das „Tor“ der Sitzungssteuerung passieren konnte, ist eine weitere Verarbeitung durch die Integrationsumgebung möglich.

Teile der Bearbeitung werden dabei vom Gateway auf unterstützende Teilkomponenten ausgelagert. So wird die eigentliche Überprüfung von Benutzerkennungen durch einen eigenständigen Benutzermanager durchgeführt, die Zuordnung von Benutzerkennungen zu Benutzersitzungen sowie die Speicherung und Verwaltung von Sitzungsdaten durch einen Sitzungsmanager geleistet. Die dritte Phase obliegt der Analysekomponente. Wie unter Abschnitt 4.3.3 beschrieben extrahiert diese die zur Generierung einer Momentaufnahme der Benutzungsschnittstelle notwendigen Daten und stellt sie in Form einer generischen Beschreibung über eine Schnittstelle zur Verfügung, an welcher das Gateway diese abrufen kann. Die Übersetzung der generischen Beschreibung in eine konkrete Repräsentation wird sodann in der vierten Phase vom Transformator durchgeführt, wobei sich der Transformator gegebenenfalls der im System gespeicherten Profildaten des Endgerätes oder der Anwendungsklasse bedient. Die fünfte und letzte Phase ist schließlich wieder in der Verantwortung des Externen Kommunikationsadapters, der die Daten zurück an das Endgerät übertägt.

Ein wesentliches Merkmal der Integrationsumgebung besteht in der Möglichkeit dazu, jederzeit zwischen den für den Zugriff auf die entfernte Anwendung verwendeten Endgeräten zu wechseln. Darüber hinaus ist der Anwender nicht an die Nutzung einer einzelnen Anwendung gebunden, sondern kann nacheinander mehrere Anwendungen starten und sodann zwischen diesen wechseln. Damit dieses möglich ist, muss neben dem zuvor beschriebenen internen Steuerungsmechanismus eine Möglichkeit existieren, aus der festen Anordnung der fünf Phasen auszubrechen und aktiv auf die Steuerung einzuwirken. Es muss die Möglichkeit geschaffen werden, die aktuelle Anwendungssitzung auszutauschen oder die Benutzersitzung vorübergehend zu stoppen und die verbundene Gerätesitzung zu beenden. Entsprechend müssen dem Benutzer Befehle zur Verfügung gestellt werden, mit deren Hilfe dieses möglich ist.

Zentraler Ansatz hierbei ist die nachträgliche Modifikation der von der Analysekomponente erfassten Momentaufnahme der Benutzungsschnittstelle. Die Sitzungssteuerung ist in der Lage, die erhaltene Beschreibung an bestimmten Stellen dahingehend zu erweitern, dass die notwendigen Dialogelemente zum Starten weiterer Anwendungen, zum Wechsel zwischen Anwendungen usw. in die generische Beschreibung integriert werden und gemeinsam mit der ursprünglichen Momentaufnahme transformiert und übertragen werden können. Den dafür generierten Knöpfen und Menüs werden Identitäten gegeben, die in der eigentlichen Schnittstelle nicht vorkommen, so dass diese GUI–Elemente auf dem Endgerät zwar wie die in der eigentlichen Schnittstelle enthaltenen GUI–Elemente dargestellt und verwendet werden können, deren Bedienung jedoch nicht an die Anwendung zurückgemeldet, sondern von der Sitzungssteuerung selbst verarbeitet werden kann.

Es entsteht auf diese Weise ein *plattformneutrales* Steuerungsprotokoll: Die Dialogelemente der Sitzungssteuerung werden in die generische Beschreibung der Schnittstelle eingeblendet und vom Transformator in ein konkretes Zielformat übertragen. Auf dem Endgerät kann der Benutzer sodann Befehle zum Wechseln der Anwendung, zum Starten oder Stoppen weiterer Anwendungen usw. erteilen, die durch den Externen Kommunikationsadapter in das generische Format übersetzt und an die Integrationsumgebung übertragen werden. Die Sitzungssteuerung kann hierauf dann entsprechend reagieren, indem sie beispielsweise die aktive Anwendungssitzung austauscht. Das verwendete Endgerät ist hierbei unerheblich, da die Steuerungselemente nicht in die konkrete Repräsentation eingeschleift oder gar von einer nativen Applikation auf dem Endgerät selbst realisiert werden müssen.

Über den Weg einer Modifikation der Zielrepräsentation wird schließlich auch ein weiteres wichtiges Konzept innerhalb der Sitzungssteuerung umgesetzt, nämlich die Etablierung einer Sitzungssemantik für Kommunikationskanäle, die von Hause aus nicht über die Möglichkeit verfügen, einzelne Interaktionen miteinander in einen logischen Zusammenhang zu bringen. Gerade im Umfeld des Internets, wo die Verwirklichung einer Sitzungssemantik einen vergleichsweise hohen und nicht immer wünschenswerten Aufwand bedeutet, wird selbst für höherschichtige Kommunikationsprotokolle hierauf verzichtet. Im Falle von HTTP bedeutet dieses beispielsweise, dass aufeinander folgende Anfragen bei einem HTTP–Server nicht als zu einer Sitzung gehörig zu erkennen sind. Während dieses den Vorteil besitzt, dass Verbindungsabbrüche für die Kommunikationspartner weitgehend unbemerkt bleiben, führt es im Zusammenhang mit einer anwendungsorientier-

ten Kommunikation zu unbefriedigenden Ergebnissen, da einzelne Zugriffe nicht bestimmten Anwendern bzw. Anwendungen zugeordnet werden können. Um nun möglichst beliebige Endgeräte in den Kontext eines Anwendungssystems integrieren zu können, ist es auch in solchen Fällen essentiell, eine Sitzungssemantik über das eigentliche Kommunikationsprotokoll zu legen.

Zu diesem Zweck bietet sich der Weg einer „In–Band“ Kommunikation an, der anhand des Anwendungsfalles HTTP erläutert werden soll. Erfolgt der Zugriff über den für HTTP zuständigen ECA, so wird bei der ersten Kontaktaufnahme im Zuge der Generierung einer Gerätesitzung eine für die Gerätesitzung eindeutige Kennung erstellt und diese zunächst vom ECA selbst als Teil der Sitzungsdaten vermerkt. Die vom ECA über die Anfrage informierte Sitzungssteuerung erzeugt eine neue Benutzersitzung und speichert nun ebenfalls die zuvor vom ECA generierte Sitzungskennung, dieses Mal als Teil der Daten zur Benutzersitzung. Sie erzeugt einen Anmeldeschirm und integriert die Kennung unsichtbar für den Anwender in die an das Endgerät zurückgegebene Zielrepräsentation. Die Zielrepräsentation ist dabei derart präpariert, dass nach Eingabe von Benutzerkennung und Passwort auf dem Endgerät neben diesen beiden Werten gleichzeitig die Sitzungskennung vom Endgerät an den ECA übergeben wird. Der ECA extrahiert die Kennung und kann durch den Vergleich mit den ihm bekannten Kennungen feststellen, dass die eingegangenen Daten zu der zuvor generierten Gerätesitzung gehören. Er gibt diese Information gemeinsam mit den vom Anwender eingegebenen Daten an die Sitzungssteuerung weiter, die ihrerseits eine Zugehörigkeit zur entsprechenden Benutzersitzung erkennen kann.

Da die Benutzersitzung die Information enthält, dass die Sitzung zuvor die zweite Phase (Authentisierung/Autorisierung) durchlaufen hat, sie jedoch noch über keine Verbindung zu einer Anwendungssitzung verfügt, kann die Sitzungssteuerung nunmehr den Start einer Anwendung veranlassen und hiernach die Erfassung einer Momentaufnahme der Benutzungsschnittstelle anfragen. In die erhaltene Beschreibung werden nun zunächst die oben beschriebenen GUI–Elemente zur Sitzungssteuerung eingefügt und es wird dann wie schon zuvor die nicht sichtbare Sitzungskennung eingebaut. Auch in diesem Falle ist wieder dafür gesorgt, dass eventuell vom Endgerät eintreffende Rückmeldungen unter anderem diese Kennung enthalten werden, so dass sowohl der Externe Kommunikationsadapter als auch die Sitzungssteuerung erkennen können, zu welcher Geräte– bzw. Benutzersitzung die Rückmeldung gehört.

Die Speicherung der Sitzungskennung sowohl in der Geräte– als auch in der Benutzersitzung erfolgt dabei nur scheinbar redundant: Während die Gerätesitzung vom ECA verwaltet wird und diese somit für den Kommunikationskanal spezifisch ist, zeichnet für die Benutzersitzung die Sitzungssteuerung verantwortlich; diese ist entsprechend nicht von einer bestimmten Plattform abhängig. Aus diesem Grunde muss auch bei der Etablierung einer Sitzungssemantik eine zweifache Speicherung der Sitzungskennung erfolgen, nämlich einerseits plattformspezifisch durch den Externen Kommunikationsadapter und andererseits plattformneutral durch die Sitzungssteuerung. Im Einzelfall können beide miteinander übereinstimmen, zur Wahrung der Allgemeingültigkeit des Konzepts ist jedoch die gewählte Trennung notwendig.

4.4 Einbindung von Diensterbringern – Integration durch Migration

Nachdem sich der erste Teil dieses Kapitels der Integration mobiler Dienstnehmer in ein verteiltes Anwendungsszenario widmete, soll nunmehr auf die Integration mobiler Diensterbringer eingegangen werden. Offensichtlich liegt in diesem Falle das Integrationsziel darin, das lokale Dienstangebot entfernten Kommunikationsknoten zur Verfügung zu stellen. Wie im Falle der zuvor betrachteten Dienstnehmerintegration kann diese grobe Zielstellung auf vielfältige Weise verfeinert und konkretisiert werden, wobei auf der Ebene der Anwendungsschicht, denn wiederum soll nur diese betrachtet werden, vornehmlich zwei Richtungen verfolgt werden:

- Zurverfügungstellung der vom Diensterbringer angebotenen Programmierschnittstelle (API) für andere Kommunikationsknoten
- Zurverfügungstellung der (eventuell grafischen) Anwendungsschnittstelle zur entfernten Nutzung durch menschliche Benutzer.

Grundsätzlich unterscheidet sich die Aufgabenstellung bei Betrachtung der Anwendungsebene also nicht wesentlich zwischen der Integration mobiler und der nicht mobiler Diensterbringer. Auch herkömmliche Diensterbringer in verteilten Systemen werden auf diese Art integriert: Dienste in CORBA Umgebungen beispielsweise zumeist nach der erstgenannten Methode, die im ersten Teil des Kapitels behandelten Applikationsserver eher nach der letztgenannten. Andererseits gibt es auch Aspekte, in denen sich die Integration mobiler Diensterbringer deutlich abhebt:

- Sie verfügen über wesentlich geringere Ressourcen, so dass herkömmliche Integrationsmethoden unter Umständen nicht realisierbar sind.
- Mobile Diensterbringer weisen eine erheblich höhere Verhaltensdynamik auf: Ihr Dienstangebot ist in stärkerem Maße durch Verbindungsabbrüche, Relokationen, schwankende Dienstgüte usw. gekennzeichnet, als dieses bei herkömmlichen Diensterbringern der Fall ist.
- Schießlich weisen mobile Diensterbringer eine wesentlich höhere Dynamik hinsichtlich ihres Dienstangebots auf. Aufgrund ihrer Exponiertheit sind sie eher defekt, sie werden häufiger gestohlen und die auch in diesem Bereich kurzen Innovationszyklen wirken sich aufgrund des vergleichsweise geringen Preises stärker aus. Daher werden mobile Diensterbringer weitaus öfter ausgetauscht, wobei das Dienstangebot des Nachfolgemodells unter Umständen stark vom vorherigen abweicht.

Hohe Dynamik und geringe Ressourcen sind entsprechend die wichtigsten Faktoren, die bei der Behandlung mobiler Diensterbringer zu den bisherigen Merkmalen von Diensten in verteilten Systemen treten. Entsprechend muss auch das eingeführte Konzept der Middleware an die geänderten Anforderungen angepasst werden. Bevor auf den im Rahmen dieser Arbeit gewählten Ansatz näher

eingegangen wird, sollen zunächst die Konzepte der Middleware kurz umrissen und dann anhand von Beispielen die wesentlichen Merkmale mobiler Middlewaresysteme skizziert werden.

4.4.1 Middleware im Überblick

Während frühe verteilte Systeme explizit im Hinblick auf den jeweiligen Anwendungsfall realisiert wurden, wird seit etwa Mitte der achtziger Jahre auf das Konzept der *Middleware* als wichtiges Hilfsmittel im Systementwurf zurückgegriffen. Mit Hilfe von Middleware Systemen werden die wiederkehrenden, nicht anwendungsspezifischen Realisierungsaspekte in Infrastrukturen zusammengefasst, die sodann als Basis für die eigentliche Anwendungsentwicklung dienen können.

Der maßgeblich an der Prägung des Middleware Begriffs beteiligte Philip A. Bernstein definiert in [Ber96] den Dienst einer Middleware als *„[...] a general purpose service that sits between platforms and applications.“* Er nennt weiter folgende typische Merkmale, die eine Middleware auszeichnen:

- *„they are generic across applications and industries“*
- *„they run on multiple platforms“*
- *„they are distributed“*
- *„they support standard interfaces and protocols“*

Middleware ist also vor allem als generisch zu bezeichnen. Sie ist weder hinsichtlich der unterstützten Anwendungen noch hinsichtlich der möglichen Anwendungsgebiete (bzw. Industriezweige) eingeschränkt, sondern in beliebigen Kontexten nutzbar. Auch hinsichtlich der berücksichtigten Computersysteme ist Middleware vielseitig; sie sollte auf verschiedenen Plattformen lauffähig sein. Des weiteren ist sie verteilt bzw. verteilbar, kann also im Netzwerk auf unterschiedlichen Knoten angeordnet werden. Und schließlich ist sie offen und unterstützt standardisierte Schnittstellen und Protokolle. Bei der Realisierung verteilter Anwendungssysteme wird der Anwendungsentwickler durch Middleware von der Behandlung von Problemen befreit, die in diesem Zusammenhang zu erwarten sind bzw. zwangsläufig behandelt werden müssen:

- Anpassung von Datenformaten zwischen beteiligten Plattformen,
- Transformation komplexer Datenstrukturen zu deren Übertragung über Netzwerkstrecken,
- Dienstregistrierung und Dienstlokation,
- Absicherung von Schnittstellen und Übertragungswegen et cetera.

Dabei werden Middleware Systeme zumeist nicht als bloße Programmbibliotheken realisiert, die bei der Anwendungsentwicklung verwendet werden können,

sondern stellen selbst Dienste zur Verfügung, die bei der Realisierung der verteilten Infrastruktur eine wesentliche Rolle spielen. Als mögliche Dienste, die als Teil einer Middleware angeboten werden können, lassen sich unter anderem folgende Beispiele nennen (vgl. [Ber96]:

Presentation Management zur Verwaltung von Eingabemasken oder Grafiksystemen.

Computation zur Konvertierung von Daten, zur Durchführung math. Berechnungen usw.

Information Management zur Verwaltung von Daten oder von Zugriffsreferenzen

Communications zur Versendung von Nachrichten oder entfernten Prozeduraufrufen

Control zur Verwaltung und Koordination von Aufträgen

System Management zur Zugriffssteuerung, Konfiguration oder Abrechnung von Diensten

Das Prinzip der Middleware wurde in einer Vielzahl von Systemen umgesetzt und zur Realisierung verteilter Anwendungssysteme eingesetzt. Dabei weichen die einzelnen Ansätze jeweils in ihrer besonderen Unterstützung bestimmter Aspekte im Entwurf verteilter Systeme voneinander ab. Sie unterscheiden sich in Art und Umfang der angebotenen Bibliotheken und in den Diensten, die zur sofortigen Nutzung zur Verfügung stehen. So wird bei CORBA [Sie96, HV99] beispielsweise stark auf die Berücksichtigung objektorientierter Konzepte Wert gelegt, während DCE [Sch93, RKF93] sich aufgrund des ausgereiften Sicherheitskonzepts und der angebotenen Dienste insbesondere für den Aufbau sehr großer verteilter Umgebungen eignet. Mit Voyager [Gla99] steht eine sehr effiziente Umgebung zur Realisierung von Systemen mit migrierenden Anwendungsobjekten zur Verfügung usw.

4.4.2 Nutzung von Middleware zur Geräteintegration

Insgesamt ist bestehenden Middleware Systemen die nur sehr eingeschränkte Unterstützung mobiler Anwendungsszenarien gemeinsam. Zu deren Unterstützung musste das Konzept der Middleware entsprechend erweitert werden, wie die im Folgenden skizzierten mobilen Middleware Systeme deutlich machen. Während die beschriebenen Ansätze dabei in ihrer Realisierung voneinander abweichen, ist ihnen doch die Behandlung der bereits oben für mobile Umgebungen dominierenden Probleme gemein, nämlich einer hohen Dynamik bei geringen Ressourcen.

4.4.2.1 Mobile CORBA

Die CORBA Middleware (Common Object Request Broker) [Sie96] stellt einen derzeit weit verbreiteten Ansatz im Bereich der Middleware Systeme dar. Auf-

bauend auf einem als ORB (Object Request Broker) bezeichneten Kommunikationsbus ermöglicht es einen plattformübergreifenden Zugriff auf Objektimplementationen, die innerhalb eines Netzwerksystems angeboten werden. Der ORB dient dabei der Entkopplung der Kommunikation zwischen Klienten (den aufrufenden Objekten) und Servern (den aufgerufenen Objekten). Er bietet jeweils lokal eine Programmierschnittstelle an, über die sich Server anmelden und ihre Objektschnittstelle dem System mitteilen können. Auf der anderen Seite kann über eine vergleichbare lokale Schnittstelle ein Klient beim System die Existenz eines Servers mit passender Schnittstelle erfragen und diesem dann, über den Umweg durch den ORB, Nachrichten bzw. Methodenaufrufe zusenden. Dem ORB obliegt dabei die Aufgabe der Anpassung und Serialisierung einfacher bzw. komplexer Datenstrukturen, der Realisierung entfernter Aufrufe, der Bereitstellung von Schnittstellen in einer für den Aufrufer passenden Programmiersprache usw.

Neben dem ORB als Kern einer jeden CORBA Umgebung können weitere Dienste existieren, die innerhalb der verteilten Umgebung genutzt werden können. Zu diesen als *CORBA Services* bezeichneten Diensten gehören Objektspeicher, Dienstvermittlungen, Transaktionsdienste oder Ereignisdienste als eher generische Dienste (sog. „horizontale“ Dienste) sowie vollständige branchenspezifische Anwendungen wie Personalverwaltungen, Abrechnungsdienste o.ä. (sog. „vertikale“ Dienste). Dienste – und hierbei insbesondere vertikale Dienste – sind optional und im Gegensatz zum ORB nicht als Teil einer CORBA Umgebung vorauszusetzen. Abbildung 4.43 skizziert den prinzipiellen Aufbau der CORBA Architektur.

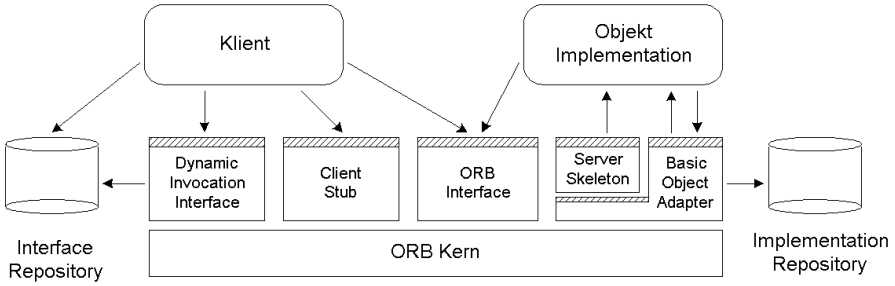


Abbildung 4.43: Die CORBA Architektur

Die CORBA Architektur mit ihren horizontalen sowie vertikalen CORBA Services ist in der Literatur hinlänglich beschrieben (so z.B. in [Sie96], [Web01] oder den offiziellen Standarddokumenten der Object Management Group). Daher soll an dieser Stelle lediglich ein im Zusammenhang mit der Unterstützung mobiler Systeme relevanter Aspekt angerissen werden, nämlich die ORB Interoperabilität. Während nämlich Abbildung 4.43 den ORB als einen nahtlosen Kommunikationsbus darstellt, auf dem Klient und Objektimplementation aufsetzen, handelt es sich hierbei tatsächlich um eine idealisierte Sicht. So ist der ORB in Wirklichkeit zumindest über mehrere Rechnerknoten verteilt, wobei auf jedem Knoten eine eigenständige ORB Instanz einen Teil des gemeinsamen Bussystems darstellt. Dieser Bus ist dabei wahrscheinlich auch diesbezüglich nicht homogen. Vielmehr sind verschiedene Knotenpunkte auf Basis unterschiedlicher Hardware bzw. unterschiedlichen Betriebssystemen realisiert und daher auch die

ORB Implementationen unterschiedlich. Um dennoch die Illusion eines einheitlichen Kommunikationsmechanismus zu schaffen, muss ein gemeinsames Kommunikationsprotokoll existieren, mit welchem die ORBs plattformübergreifend Objektinstanzen referenzieren sowie Methodenaufrufe, Parameter und Ergebnisse untereinander austauschen können. Der hierfür verwendete Mechanismus wird als *Inter–ORB Protocol* (IOP) bezeichnet.

Ein IOP definiert, wie auf der Basis eines bestimmten Kommunikationsmechanismus die von CORBA verwendeten Daten kodiert und zwischen ORB Instanzen übertragen werden können. Neben einer generischen Variante dieses Protokolls, welche als *Generic Inter–ORB Protocol* (GIOP) bezeichnet wird und die von allen CORBA konformen ORBs angeboten werden muss, ist in marktrelevanten CORBA ORB Implementationen vor allem das *Internet Inter–ORB Protocol* (IIOP) vorhanden. Dieses verfeinert das GIOP hinsichtlich der Verwendung in TCP/IP basierten Kommunikationsnetzen. Da die überwiegende Menge an CORBA Implementationen in derartigen Netzen zum Einsatz kommt, stellt IIOP das gängige Protokoll zur Kopplung von ORBs dar.

Warum ist IOP nun im Zusammenhang mit der Unterstützung mobiler Geräte durch CORBA nennenswert? Grundsätzlich kann man den Begriff *Mobile* CORBA auf dreierlei Art interpretieren:

1. Mobilität als **Mobilität der Objekte**: Eine Mobile CORBA Implementation muss hierbei die Objekte in ihrer Migration zwischen einzelnen Knotenpunkten der CORBA Umgebung unterstützen.

2. Mobilität als **Mobilität des Rechners**, auf dem die ORB Instanz abläuft (a): Abstrahiert man von der verminderten Qualität der zur Verfügung stehenden Kommunikationsverbindungen des mobilen Rechners (also Laptop, PDA, Mobiltelefon usw.), so muss der ORB hierbei insbesondere mit der Ressourcenknappheit des Geräts zurechtkommen. Und alternativ:

3. Mobiltät als **Mobilität des Rechners**, auf dem die ORB Instanz abläuft (b): Betrachtet man die besonderen Probleme von Kommunikationsverbindungen in mobilen Umgebungen, so muss der ORB die sich ändernden Merkmale des Kommunikationskanals, wechselnde Ursprungs– und Zieladressen et cetera behandeln.

Die Unterstützung von Objektmobilität setzt eine sehr weitgehende Plattformunabhängigkeit der Objektimplementationen voraus, wie sie derzeit nur bei Realisierungssprachen wie Java zu erlangen ist. Realisierungen dieser Art von Mobilität in CORBA (vgl. z.B. „Jumping Beans“ [Eng98]) sind mit Agentensystemen vergleichbar, wobei die Infrastruktur und die migrierenden Objekte über Schnittstellen verfügen, welche zu CORBA konform sind. Die strikte Forderung nach Plattformunabhängigkeit läuft in gewisser Weise jedoch dem Anspruch von CORBA zuwider, als gemeinsame Adaptionsplattform für stark heterogene Anwendungssysteme aufzutreten, da eine Mobilisierung bei in C++ oder gar Assembler realisierten Objektimplementationen nahezu undenkbar ist.

Die zweite Form der Unterstützung von Mobilität geht von der Annahme aus, dass CORBA oftmals auf mobilen Geräten zum Einsatz kommen soll, eine Kommunikation jedoch nicht während deren Relokation notwendig ist. Durch

Einführung dieser Einschränkung entfällt die ansonsten notwendige und überaus komplexe Unterstützung wechselnder Kommunikationskanäle, sich ändernder Ursprungs– und Zieladressen usw. Diese eingeschränkte Form der Mobilität wurde beispielsweise im PalmORB Projekt [RSC+99], im TAO Projekt [GS99], in der IONA IIOP Engine [Tec97] favorisiert und kommt ebenso bei MICO [PR00] zum Einsatz, welches im Stanford Digital Library Project [PBC+99] den mobilen Zugriff auf digitale Bibliotheken ermöglicht. In diesen Fällen wird eine stark reduzierte ORB Implementierung nach dem bibliotheks–basierten Ansatz verwendet (vgl. Abb. 4.44 links). Der ORB stellt nach diesem Verfahren keinen eigenständigen Prozess dar, der auf den beteiligten Rechnerknoten jeweils ständig verfügbar ist, sondern er wird als eine Sammlung von Routinen mit in die Server– oder Klientenimplementation integriert. Die Programminstanz übernimmt die notwendigen Aufgaben des ORB hierbei während ihrer Laufzeit quasi selbst und kommuniziert über ein minimales IIOP. Diese Kommunikation setzt dabei eine während der gesamten Kommunikation bestehende TCP/IP Verbindung voraus. Sie ist insbesondere nicht in der Lage, Relokationen des Rechners und die hiermit verbundenen Änderungen der Zugriffsadressen von Objektimplementationen zu behandeln.

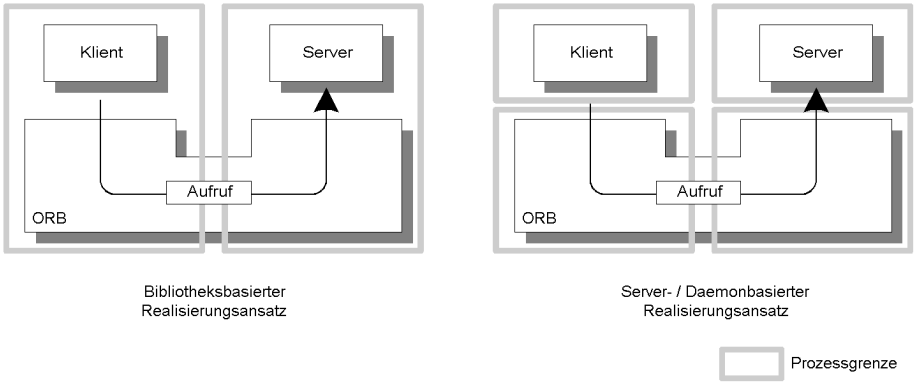


Abbildung 4.44: Bibliothek und Daemon als mögliche Realisierungsvarianten für Object Request Broker

Will man derartige Probleme behandeln, so wird üblicherweise auf der Ebene des Inter–ORB Protokolls angesetzt. Dabei wird dann das bestehende IIOP durch eine Unterstützungsumgebung ergänzt, welche als Stellvertreter (Proxy) zwischen den ORB Instanzen des Klienten sowie des Servers steht und die im mobilen Kontext auftretenden Probleme abfängt. Auf diese Art wird beispielsweise bei ALICE (Architecture for Location Independent CORBA Environments) [HCC99] verfahren (siehe Abb. 4.45). Hierbei werden auf der Seite des mobilen Geräts zwischen der IIOP Schicht und der TCP/IP Schicht zwei Adaptionsschichten eingezogen. Neben einer generischen Anpassung durch die so genannte „Mobility Layer“ steht dabei eine speziell auf das IIOP zugeschnittene S/IIOP Schicht. Erstere verdeckt Verbindungsabbrüche auf IP Ebene, Migration (Handoff) zwischen Netzzugangsknoten und andere allgemeine Kommunikationsprobleme vor höheren Schichten. Durch die S/IIOP Schicht werden hingegen spezifische Probleme des Internet Inter–ORB Protokolls in mobilen Umgebungen behandelt. Es werden Objektreferenzen (Interoperable Object References, IORs)

derart verändert, dass sie auf den im Festnetz angeordneten Stellvertreter des mobilen CORBA ORBs verweisen, anstelle auf das Objekt auf dem Mobilgerät selbst. Auf dem Stellvertreter ist es die Aufgabe der S/IIOP Schicht, eintreffende Aufrufe abzufangen und an das eigentliche Objekt auf dem Mobilgerät weiterzuleiten. Die Modifikation des IIOP ist dabei aus der Sicht des auf dem Endgeräts angeordneten ORBs ebenso unsichtbar wie aus der Sicht der CORBA Objekte.

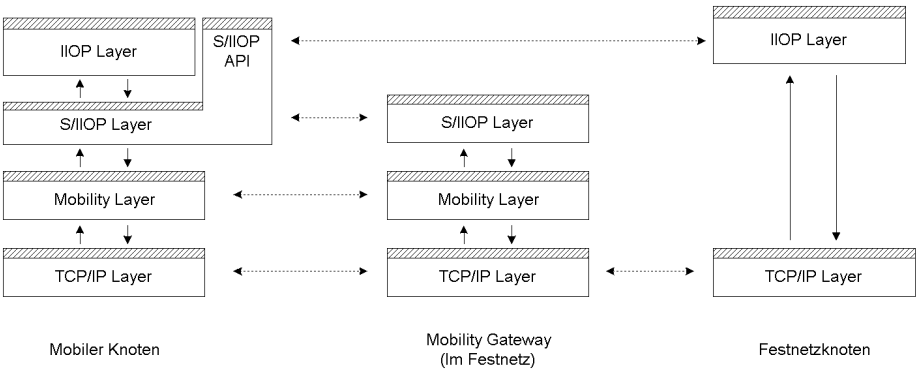


Abbildung 4.45: Architecture for Location Independent CORBA Environments (ALICE) (nach [HCC99])

Die Object Management Group (OMG) schlägt dagegen vor, nicht unterhalb des IIOP anzusetzen, sondern vielmehr anstelle des für TCP/IP als Transportprotokoll spezifischen IIOP das generische GIOP zu verwenden und unterhalb von GIOP dann die für das jeweils verwendete Kommunikationssystem notwendigen Anpassungen vorzunehmen (vgl. [RAA+98]). Allgemein ergibt sich dabei das in Abbildung 4.46 dargestellte Bild: An die Stelle von IIOP tritt GIOP, das über eine spezifische Anpassungsschicht in die jeweils verwendete Transportschicht abgebildet wird. Am Übergang zwischen den verschiedenen Protokolldomänen sorgen dabei so genannte „Access Gateways“ für die notwendige Protokollanpassung.

Ein konkreter Umsetzungsvorschlag nach [RAA+98] ist in Abbildung 4.47 zu sehen. Als Anpassungsschicht von GIOP wird auf dem mobilen Knoten das *Mobile Wireless Inter–ORB Protocol* (MWIOP) verwendet, das im Bereich des Mobilnetzes zwischen dem Mobilknoten selbst und dem Access Gateway eingesetzt wird. Festnetzknoten verwenden hingegen weiterhin das herkömmliche IIOP. Auf dem Gateway findet dabei eine Umsetzung zwischen MWIOP und IIOP statt.

Auf die besondere Relevanz des Gateways zur Nutzung von CORBA in mobilen Umgebungen wird auch in [KLP99] eingegangen. Wie dort ausgeführt wird, obliegt den Gateways, hier auch als *Bridges* bezeichnet, neben der Umsetzung der verschiedenen IOP Varianten zueinander auch deren effiziente Auffindung (z.B. durch die Dienstvermittlungsfunktion eines *Traders*). Bridges sollen hiernach über Verzeichnisse der ihnen bekannten Objektinstanzen verfügen und diese nach der Relokation eines Gerätes und der damit verbundenen Migration von Objekten in den Verantwortungsbereich einer anderen Bridge miteinander abgleichen können. Auf diese Weise soll dann neben dem Zugriff von auf mobilen

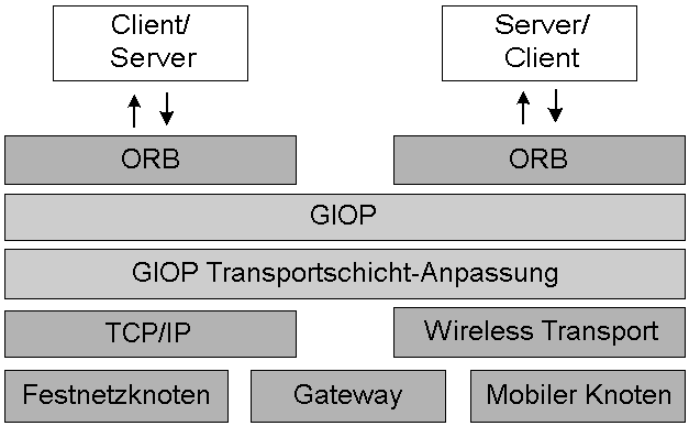


Abbildung 4.46: CORBA Reference Networking Architecture (RNE) der Object Management Group (nach [RAA+98])

Geräten befindlichen Klienten hin zu Objektinstanzen im Festnetz jederzeit auch der entgegengesetzte Fall möglich sein, bei dem Klienten aus dem Festnetz auf CORBA Objektinstanzen auf mobilen Geräten zugreifen können.

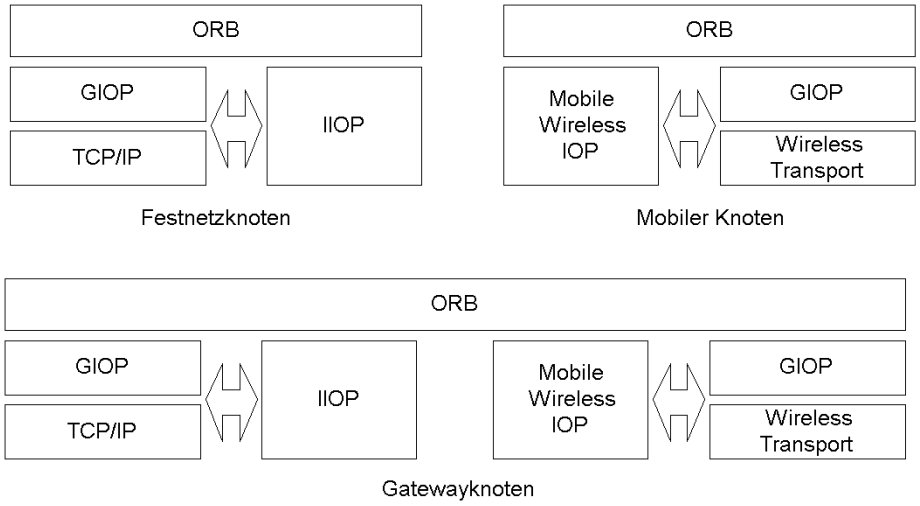


Abbildung 4.47: Architekturvorschlag zur Realisierung der RNE in mobilen Umgebungen (nach [RAA+98])

4.4.2.2 OnTheMove

Während die zuvor geschilderten Ansätze zur Integration mobiler Diensterbringer von der Erweiterung eines bestehenden Middleware Konzepts in die Domäne der mobilen Verarbeitung hinein ausgingen, wurde in dem im ACTS Projekt OnTheMove (OTM) [Har96b] entworfenen Mobile Application Support Environment (MASE) [Har96a, MP97, KRR98b, KRR98a] ein entgegengesetzter Ansatz

verfolgt: Hier stand am Ausgangspunkt der Entwurf einer explizit auf die Anforderungen mobiler Umgebungen ausgelegten Middleware, die im weiteren Verlauf dann um Schnittstellen zu herkömmlichen Middleware Systemen (unter anderem auch zu CORBA) und um die Konzepte eines Agentensystems erweitert wurde.

Das primäre Ziel von MASE ist es, die Benutzer mobiler Systeme im Kontext von UMTS in ihrem Zugriff auf personalisierte, multimediale Daten zu unterstützen, wobei neben speziell für den mobilen Einsatz gedachte Applikationen (sog. „mobile–aware applications“) gleichfalls auch solche Applikationen Berücksichtigung finden sollen, die nicht für derartige Anwendungsfälle entwickelt wurden (sog. „legacy applications“). MASE teilt die hierfür notwendigen Funktionen einer Middleware in drei funktionale Gruppen ein (Siehe Abb. 4.48):

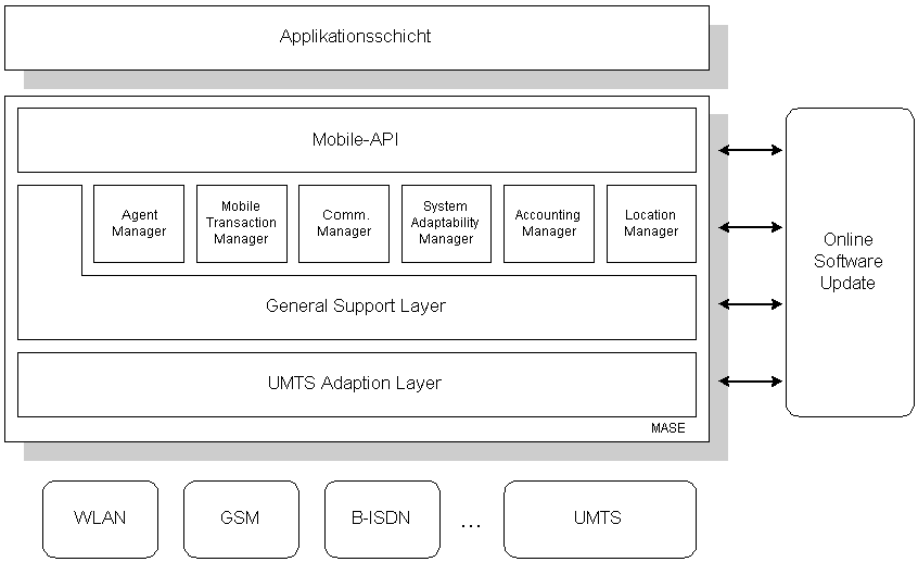


Abbildung 4.48: Das OnTheMove Mobile Application Support Environment (MASE)

UMTS Adaption Layer (UAL) : Die UAL bildet das Fundament der MASE Middleware. Ihre Aufgabe ist es, von den Eigenheiten der verschiedenen Kommunikationsnetzwerke zu abstrahieren und den höheren Diensten eine einheitliche Schnittstelle hierzu bereitzustellen. Dabei werden unter anderem auch Informationen über Status oder Dienstgüte der Verbindung erfasst und weitergeleitet.

General Support Layer (GSL) : Die GSL dient der Bereitstellung von allgemeinen Diensten der Infrastruktur, wie sie als integraler Bestandteil einer jeden Middleware zur Verfügung stehen sollten. Hierzu zählt ein Verzeichnisdienst ebenso wie eine Sicherheitsüberwachung, ein Objektspeicher usw. Daneben kann es dann eine Reihe weiterer (z.T. vertikaler) Dienste geben, die unmittelbar zur Unterstützung spezieller Anwendungsfälle in die Middleware eingebunden werden können.

Mobile–API : Als Abschluss in Richtung zur Anwendungsebene wird von der MASE Middleware die so genannte Mobile–API angeboten. Diese dient der weiteren Vereinheitlichung des Angebots der Infrastruktur und wird von allen höherschichtigen Applikationen genutzt.

Auf der Basis dieser Architektur bietet MASE die notwendige Unterstützung zur Realisierung von Anwendungen, die auf den Kontext mobiler Szenarien zugeschnitten sind. So werden in [Har96b] als mögliche Bereiche unter anderem lokationsabhängige Informationssysteme, Nachrichtensysteme, mobile E–Commerce Systeme, Gruppenkalender genannt. Die MASE sorgt dabei auf der Basis von Benutzer– und Geräteprofilen (ähnlich den unter 4.3.5 genannten Ansätzen) für die optimale Anpassung der Dienstgüte an die zur Verfügung stehende Kommunikationsverbindung, den nahtlosen Übergang zwischen verschiedenen Kommunikationssystemen und die Bereitstellung von Informationen zu Verbindungsgüte, Standort o.ä. zur weiteren Verwendung durch das Anwendungssystem.

Neben den Merkmalen einer Middleware herkömmlicher Ausprägung verfügt die MASE Middleware über die Fähigkeiten eines Agentensystems. So können kommunikationsintensive Aufgaben an Softwareagenten delegiert werden, welche diese dann auch während Unterbrechungen der Kommunikationsverbindung autonom bearbeiten und nach der Wiederherstellung der Kommunikationsverbindung und danach erfolgter Migration auf das Endgerät dem Anwender präsentieren können (siehe [KRR98b, KRR98a]). Die mobilen Agenten können dabei von Applikationen über die Mobile–API angesteuert werden, sie bedienen sich jedoch auch selbst der über die Mobile–API angebotenen Funktionen der MASE Infrastruktur.

4.4.2.3 JINI

Betrachtet man die Unterstützung von Mobilität in CORBA, AnsaWare oder DCE, so ist jenen Middleware Systemen gemeinsam, dass die speziellen Merkmale mobiler Geräte oftmals nur unzureichend unterstützt werden und diese Unterstützung oftmals nur mit logischen Brüchen in das Gesamtkonzept eingefügt werden konnte. Dieses stellt insbesondere bei älteren Systemen wie DCE ein ernsthaftes Problem für deren zukünftigen Einsatz dar. Im Gegensatz dazu stand beim Entwurf der Middleware JINI (*Java Intelligent Network Infrastructure*) wie bei OnTheMove/MASE von Beginn an ausdrücklich der Aufbau von *Ad–hoc* Netzwerken zwischen Geräten aus dem Segment der Unterhaltungs–, Haushalts– und Mobilgeräte im Vordergrund. Bereits in der Konzeptionsphase wurde also auf die spezifischen Eigenschaften mobiler Geräte Rücksicht genommen.

Im Hinblick auf die in der Dokumentation vorgestellten Einsatzszenarien fällt auf, dass eine besondere Betonung auf der Integration solcher Geräte in *dynamische* verteilte Anwendungsumgebungen liegt. Im Gegensatz zu CORBA oder DCE wird nicht die Integration von Dienstnehmern in statische, verteilte Umgebung in den Vordergrund gestellt, sondern die spontane Vernetzung gleichberechtigter Mobilgeräte oder der kurzfristige Datenaustausch eines Mobilgerätes

mit einer Übergabestation (von Goodman [Goo97] beispielsweise als „Datentankstelle“ bezeichnet). In diesem Zusammenhang oft genannte Beispiele sind ([Sun99b, Sun99c]):

Der vernetzte Präsentationsraum : Ein Laptop wird in einen Präsentationsraum gebracht und kann nach Einbindung in das lokale Netzwerk auf die vorhandenen Ressourcen wie Projektoren, Lautsprecher, Drucker usw. zugreifen, ohne dass hierfür eine explizite Konfiguration vorgenommen werden müsste.

Der Fotodrucker : Eine Digitalkamera wird in die Nähe eines stationären Farbdruckers gehalten, und der Benutzer kann ohne vorherige Konfiguration die gespeicherten Bilder auf einem mit dem Drucker verbundenen Bildschirm anschauen und diese sodann auf dem Drucker selbst ausdrucken.

Der vernetzte Videorecorder : Ein Videorecorder wird im Wohnzimmer an das Stromnetz angeschlossen und kann ohne vorherige Anpassung im Zusammenhang mit der Stereoanlage genutzt werden. Die Bilddaten werden auf dem Fernseher angezeigt und die Audiodaten vom Verstärker auf die Lautsprecher übertragen. Videos können aus dem im Haus befindlichen Personalcomputer übernommen werden.

All diesen Beispielen ist gemeinsam, dass die beteiligten Komponenten ohne weitere Anpassungen miteinander Kontakt aufnehmen und dann interagieren können. Es müssen weder Treiber installiert werden, noch ist es notwendig, Konfigurationsparameter der entfernten Knoten auf dem eigenen Gerät einzutragen oder sich explizit bei einer zentralen Verwaltungsinstanz zu registrieren. Außerdem ist es nicht notwendig, nach erfolgter Dienstnutzung die Kommunikationsumgebung aktiv zu verlassen, d.h. Registrierungen aufzuheben, erneut die Konfiguration beteiligter Geräte zu modifizieren usw. Dabei sind Dienstnehmer genauso wie Diensterbringer als aktive Komponenten vertreten und können das Ein– bzw. Austreten aus einer Interaktionsbeziehung selbst initiieren.

Primäres Entwicklungsziel für die JINI Middleware war es also, die Realisierung dynamischer verteilter Anwendungsszenarien gegenüber bisherigen Ansätzen deutlich zu vereinfachen. Eine zunächst komplexe Installation und der im weiteren Verlauf stetig zunehmende Wartungsaufwand machen den Einsatz von verteilten Systemen im Bereich der Konsumelektronik derzeit nahezu unmöglich. Werden die bisherigen, eher statischen Systeme nun jedoch noch (wie etwa in der oben anhand konkreter Beispiele geschilderten Weise) dynamisiert, geht man also zu Ansätzen der Ad hoc–Vernetzung über, so sind die entstehenden Probleme mit herkömmlichen Middleware Systemen kaum noch beherrschbar.

Grundsätzlich beruht der Lösungsvorschlag durch die JINI Middleware auf fünf Prinzipien (vgl. [Sun99b, Edw99], die im Folgenden vorgestellt und beschrieben werden sollen:

Discovery : Auffinden von Basisdiensten der JINI Infrastruktur

Lookup : Registrieren und Vermitteln von Diensterbringern der verteilten Umgebung

Leasing : Zeitliche Begrenzung von Beziehungen zwischen Entitäten innerhalb der verteilten Umgebung

Remote Events : Ereignismechanismus für den Einsatz in verteilten Umgebungen

Transactions : Unterstützung für die verlässliche Transaktionsverarbeitung im Kontext verteilter Anwendungsszenarien

Was verbirgt sich hinter diesen Konzepten? Die folgenden Absätze sollen eine erste Klärung bringen.

Ein wesentliches Problem aller Middleware Systeme besteht in der Integration neuer Kommunikationsknoten (d.h. Diensterbringer und –nehmer) in eine bestehende Anwendungsumgebung. Damit jedoch eine Registrierung oder gar die Suche nach passenden Kommunikationspartnern stattfinden kann, müssen die dafür notwendigen zentralen Dienste der Infrastruktur dem zu integrierenden Knoten mitgeteilt werden. Im Gegensatz zur manuellen Konfiguration, wie sie bisher üblich ist, wird in JINI ein verbindliches Protokoll definiert, mit dem Teilnehmer eines verteilten Anwendungsszenarios in JINI diesen initialen Schritt der Erkundung („Discovery“) ihrer Umgebung durchführen können. In diesem Schritt können sich die Dienste der Infrastruktur zu erkennen geben und dann im weiteren Verlauf direkt angesprochen werden.

Eine wichtige Aufgabe, welche eine Middleware in der Unterstützung von Teilnehmern eines verteilten Anwendungsszenarios erfüllen muss, ist die Vermittlung von Diensten. Hierzu muss es den Diensterbringern ermöglicht werden, ihre Dienstleistung dem System mitzuteilen und Dienstnehmern, ihren Bedarf genau zu spezifizieren und hierauf die Referenz auf einen passenden Dienst zu erfragen. Die elementare Komponente einer jeden Middleware, die für diesen Schritt verantwortlich zeichnet, wird im allgemeinen als *Dienstvermittlungsinstanz* oder als *Trader* bezeichnet [MJ96]. Auch in JINI stellt die Dienstvermittlungskomponente den konzeptionellen Kern der Infrastruktur dar. Entsprechend ihrer Aufgabe zur Unterstützung bei der Suche („Lookup“) von Diensten wird diese Komponente als *Lookup Service* bezeichnet [Sun99f]. Während der initiale Schritt der Discovery dem Auffinden von infrastrukturellen Basisdiensten – eben den vorhandenen Lookup Services – dient, so ermöglicht erst die Interaktion mit einem Lookup Service die Erschließung weiterer Dienstangebote, wie sie für das Erreichen des Anwendungsziels relevant sein dürften. Da die Schritte der Erkundung der JINI Umgebung und der Registrierung eines Dienstes miteinander eng verbunden sind, werden sie oftmals zusammengefasst zum Prozess des so genannten „Discovery & Join“(vgl. [Sun99d, Sun99e]).

Das wahrscheinlich größte Problem herkömmlicher Middleware Systeme besteht darin, dass die Beziehungen zwischen Dienstnehmern und Diensterbringern zwar meistens durch Dienstvermittlungskomponenten (und somit dynamisch) hergestellt werden können, dass sie in der Folge jedoch als statisch zu bezeichnen sind: Verfügt ein Klient erst einmal über die Referenz zu einem Dienst, so behält er diese im Prinzip ohne zeitliche Begrenzung. Verlässt der Diensterbringer im weiteren Verlauf die verteilte Umgebung (z.B. bedingt durch einen Anwendungsfehler oder – gerade im mobilen Umfeld – durch Kommunikationsabbrüche bzw. Relokationen), so ist die Referenz nicht länger gültig, ihre Verwendung ist im

besseren Falle ohne Resultat, führt wahrscheinlich jedoch zum Absturz des aufrufenden Klienten. Dieses Problem, im Bereich der Programmiersprachen als „dangling pointer problem“ bekannt, verschärft sich mit der Lebenszeit des Systems zunehmend und kann schließlich dessen Verlässlichkeit empfindlich beeinträchtigen. JINI setzt diesem Problem das Konzept des *Resource Leasing* entgegen. Hierbei ist die Nutzung von Ressourcen grundsätzlich an zeitliche Limits gebunden. Wird für eine Beziehung ein solches Limit überschritten und die Beziehung in diesem Augenblick nicht aktualisiert, so verfällt sie und wird aus dem System entfernt. Während auch hierbei nach dem Absturz eines Diensterbringers für eine gewisse Zeit dessen (nunmehr ungültige) Referenz im Dienstnehmer vorhanden ist, wird sie doch spätestens nach Ablauf der Mietdauer als ungültig erkannt, da der Diensterbringer sein Angebot bei der Dienstvermittlung nicht mehr erneuern kann. Sie kann dann aus dem System entfernt werden.

Das im Bereich grafischer Schnittstellen weit verbreitete Konzept der Ereignisse (Events) hat spätestens mit der Einführung der Objektorientierung in das Middleware–Design durch CORBA auch in die verteilten Systeme Einzug gehalten. Verteilte Objekte können sich in Nachrichten über Zustandsänderungen informieren oder Dienste in Anspruch nehmen. Wie bei CORBA werden auch in JINI die entfernten Objekte hinter lokalen Stellvertretern „versteckt“ und Methodenaufrufe an diese lokalen Objekte umgeformt und über das Netzwerk weitergeleitet. Bemerkenswert an JINI ist hierbei, dass das Konzept auf eine Weise mit der Middleware verwoben wurde, die zu bisher unbekannten Möglichkeiten geführt hat. So ist es in JINI dank *Remote Events* möglich, den für die Dienstvermittlung typischen Verlauf „Dienstregistrierung“ – „Anfrage “ – „Dienstvermittlung“ – „Dienstnutzung“ (vgl. sog. „Trading Dreieck“in [MMJLT96, Mül96]) aufzubrechen: In seiner Anfrage an den Lookup Service kann ein Diensterbringer festlegen, dass die Antwort auch zu späterem Zeitpunkt noch erfolgen darf. Kann der Lookup Service die Anfrage nun nicht direkt beantworten, so meldet er dieses zwar an den Interessenten zurück. Gleichzeitig speichert er die Anfrage jedoch intern ab, so dass auch noch später das Eintreffen eines Dienstes des zuvor gesuchten Typs erkannt und das Angebot sodann in Form einer Nachricht an den Dienstnehmer zurückgesandt werden kann. JINI regelt alle in diesem Zusammenhang notwendigen Belange und stellt dem Anwendungsentwickler eine sehr stark an die aus der Programmiersprache Java bekannten Mechanismen angelehnte Sammlung von Mechanismen zur Verfügung.

Von ähnlicher Bedeutung für den Aufbau robuster verteilter Umgebungen wie die zeitlich begrenzte Nutzung von Ressourcen mittels Leasing sowie die Möglichkeit zur asynchronen Versendung von Nachrichten mittels Events ist die transaktionale Sicherung der Kommunikation anzusehen. Nur auf der Basis verlässlicher Nachrichtenübertragung ist der Aufbau missionskritischer Anwendungen möglich. Andererseits stellen die von anderen Middleware Systemen bekannten Transaktionsmonitore (z.B. DCE Encina) mit die komplexesten Dienste dar, die im Bereich verteilter Systeme denkbar sind. Eine Transaktionsbehandlung im herkömmlichen Sinne schließt sich also im Zusammenhang mit Ad–Hoc Netzwerken, wie sie von JINI unterstützt werden sollen, nahezu aus. Um dennoch die Möglichkeit zur Definition von Transaktionen zu geben, wurde bei JINI ein anderer Weg gewählt. Nicht der Transaktionsmonitor selbst wird definiert, sondern lediglich die Schnittstelle, die ein Diensterbringer im Rahmen einer transaktional gesicherten Kommunikation zu erfüllen hat. Bei JINI bedeutet dieses

konkret, dass dem transaktionalen Dienst lediglich vorgeschrieben wird, dass er über die Transaktionsschnittstelle Nachrichten wie „prepare“, „commit“ oder „abort“ entgegennehmen können muss[16]. Es wird jedoch nicht vorgeschrieben, in welcher Weise diese sodann intern realisiert werden. Auf diese Weise ist es dem Entwickler freigestellt, von den Anforderungen des Anwendungsszenarios ausgehend zu entscheiden, welcher Aufwand für die Realisierung der Transaktionen getroffen werden soll. In einem Protokoll, das zwischen PDA und Bankautomat bei der Übermittlung von Geldbeträgen in das elektronische Portemonnaie benutzt wird, würde bei der Realisierung der Methoden ein sehr hoher Aufwand getrieben werden und die transaktionale Sicherung entsprechend komplex ausfallen, während ein transaktional gesicherter Austausch der Visitenkarte zwischen zwei PDAs vergleichsweise einfach gestaltet würde. Die Stärke von JINI liegt hierbei in der Flexibilität, in welcher diese Aspekte gehandhabt werden können.

Ein weiteres wichtiges Konzept in JINI ist dasjenige einer Verwaltungseinheit, eines so genannten „Djinns“[17]. In einem Djinn können Dienstanbieter und Dienstnehmer in einer JINI Umgebung zusammengefasst werden, z.B. im Sinne einer logischen Gruppierung von Diensten. Djinns dienen somit einer Partitionierung des Namensraumes der JINI Umgebung. Jeder Djinn wird von mindestens einem Lookup Service betreut, wobei ein Lookup Service für mehrere Djinns zuständig sein kann. Im Beispiel des vernetzten Videorecorders könnte das Hausnetzwerk für jeden Raum einen Djinn vorsehen, im Beispiel des Fotodruckers könnte der Einzugsbereich des Fotodruckers als Djinn definiert sein et cetera. Oftmals orientieren sich die Grenzen eines Djinns an technischen oder administrativen Organisationsbereichen.

Neben den genannten Grundbausteinen ist es die von der Verwendung von Java als Realisierungssprache herrührende Plattformunabhängigkeit auf Seiten der Dienstnehmer und –erbringer, die JINI aus der Menge herkömmlicher Middleware Systeme heraushebt und für den Aufbau hochdynamischer verteilter Anwendungen prädestiniert. Es ist insbesondere das Prinzip der Code–Migration, das die Durchsetzung eines flexiblen Treiberkonzepts ermöglicht, wie es zur Realisierung der geschilderten Beispielszenarien notwendig ist und wie es auch im Zusammenhang mit der Integration von Diensterbringern in mobile Anwendungskontexte insgesamt wünschenswert ist.

Da JINI in seiner Verwendung mobilen Codes zur Realisierung verteilter Anwendungsszenarien ein erhebliches Potential für den Einsatz zur Integration von Diensterbringern in mobile Kontexte aufweist, soll der Beschreibung des verwendeten Prinzips im Folgenden ein eigener Abschnitt gewidmet werden.

4.4.3 Integration durch Migration

In Analogie zum Prinzip der „Integration durch Abstraktion“, wie sie als Basis für die Einbeziehung mobiler Geräte in verteilte Systemumgebungen in ihrer Rolle als Dienstnehmer bereits im ersten Teil dieses Kapitels eingehend beschrieben

[16] Für eine detaillierte Beschreibung des „Two–Phase Commit Protocols“ (2PC) siehe [CDK01]

[17] Der gewählte Name rührt von einem Wortspiel her: Das Akronym JINI entspricht in der Aussprache dem englischen Wort „genie“, dem dienstbaren Geist also, den die JINI Entwickler in der von ihnen entworfenen Middleware sehen. Das Wort „djinn“ ist ein Synonym hierzu.

wurde, soll das in JINI verwendete Prinzip zur Integration von Diensterbringern als „Integration durch Migration“ bezeichnet werden.

Das Prinzip beruht auf einer Vorgehensweise, die im Bereich der Betriebssysteme bereits hinlänglich bekannt ist: Wird ein neues Gerät in ein Computersystem integriert, so muss vor dessen Benutzung zunächst ein Treiberprogramm installiert werden. Das Treiberprogramm wird in das Betriebssystem eingebunden und bietet fortan eine Schnittstelle, über welche das Betriebssystem selbst, aber auch beliebige Anwendungen, mit dem Gerät kommunizieren können. Zwischen äußerer Schnittstelle und eigentlicher Hardware stellt das Treiberprogramm dabei eine Art Adaptionsschicht dar, die den Anwendungsprogrammierer von der Aufgabe enthebt, direkt mit der Hardware kommunizieren zu müssen. Insbesondere muss bei Änderung der Hardware lediglich die interne Realisierung des Treibers angepasst werden. Bleiben dessen Schnittstellen gleich, so werden Anwendungen von Hardwareänderungen nicht beeinträchtigt.

So, wie Treiber für direkt in das Computersystem integrierte Geräte notwendig und bekannt sind, werden sie zunehmend auch für verteilte Systeme wichtig. Immer mehr Geräte werden nicht über eine Schnittstelle direkt mit einem Computersystem verbunden, sondern verfügen über Netzwerkanschlüsse, mit denen sie an das lokale Netzwerk angeschlossen werden können. Viele Drucker verfügen heute schon über derartige Optionen und beherrschen dabei gängige Netzwerkprotokolle wie TCP/IP, IPX oder Appletalk. Auch durch das zunehmende Interesse an Funksystemen mit kurzer bis mittlerer Reichweite, die mittelfristig als Ersatz sowohl für LAN– als auch für Geräteverkabelungen dienen sollen, wird dieser Trend verstärkt: Rechensysteme kommunizieren nicht länger direkt über Kabelverbindungen mit ihrer Peripherie, sondern verwenden hierfür (Funk–)Netzwerkschnittstellen. Auch die oben skizzierten Beispiele für JINI Szenarien (Fotodrucker, vernetzter Präsentationsraum usw.) fallen letztlich in diese Kategorie: Die Digitalkamera (Computersystem) überträgt Bilder an den Fotodrucker (Peripherie) mittels IrDA, der Laptop nutzt die Peripherie des Präsentationsraumes via Bluetooth et cetera.

Wenngleich das mit einer Netzwerkschnittstelle ausgerüstete Gerät bereits ein oder mehrere Netzwerkprotokolle beherrscht und somit mit anderen Geräten interagieren kann, muss dennoch in der hergebrachten Weise ein Treiber verwendet werden, um auf dessen spezifischen Dienst zugreifen zu können. So muss der Kamera beispielsweise mitgeteilt werden, wie sie den Fotodrucker anschaltet, wie und in welchem Format sie die Fotodaten übertragen kann oder wie sie dem Drucker signalisiert, dass er die übertragenen Fotos in einer bestimmten Auflösung ausdrucken soll. Auch in verteilten Umgebungen existiert also weiterhin die Notwendigkeit zur Installation und ggf. auch Konfiguration von Treiberprogrammen.

Ein herkömmliches verteiltes Betriebssystem oder eine herkömmliche Middleware setzt nun für die Integration eines Diensterbringers (denn als solchen kann man die oben genannten Geräte allgemein ja bezeichnen) zunächst voraus, dass dessen Treiber manuell installiert wird. Das für verteilte Systeme mächtige Konzept der Dienstvermittlung wird so für mobile, dynamische Systeme stark geschwächt, da die Vermittlung eines Dienstes allein für einen potentiellen Dienstnehmer nicht ausreichend ist, weil hiernach zusätzlich die Installation eines passenden Treibers notwendig wird. JINI geht hier einen neuartigen Weg, da der Diensters-

bringer *selbst* den Treiber beisteuert. Dieser kann von der Vermittlungsinstanz (dem so genannten Lookup Service [Sun99f]) verwaltet werden und im Zuge der Dienstvermittlung dann an den Dienstnehmer weitergeleitet werden. Dank der von JINI darüber hinaus verwendeten Prinzipien des Leasing und der Remote Events (s.o.) ist des weiteren in einfacher Weise auch eine Evolution des Systems möglich: Soll der vom Diensterbringer an das System übergebene Treiber erneuert werden, so muss der Diensterbringer lediglich den aktualisierten Treiber im System anmelden und fortan die Erneuerung von Registrierungen für ältere Versionen verweigern. Auf diese Weise werden die Nutzer nach und nach gezwungen sein, den alten Treiber zu verwerfen und ihn durch einen aktualisierten Treiber zu ersetzen.

Zur Verdeutlichung des Prinzips soll der Lebenszyklus eines Diensterbringers in einem JINI Szenario skizziert werden (vgl. auch [Edw99, OW00, MWL00a]). Wird ein Diensterbringer an das Netzwerk angeschlossen (z.B. indem er in die Reichweite eines Funkempfängers kommt), so startet zunächst der Initialisierungsprozess. Wurde der für den Dienst verantwortliche Lookup Service nicht explizit vorgegeben (auf die direkte Kommunikation via Unicast–Protokoll soll hier nicht weiter eingegangen werden), so fordert der Diensterbringer alle im Netzwerk vorhandenen Lookup Services dazu auf, sich ihm gegenüber zu identifizieren. Hört ein Lookup Service diesen Aufruf, so präpariert er eine Rückmeldung, in der er neben Daten über die von ihm verwaltete Organisationsdomäne einen Codeblock – seinen Treiber – übergibt. Der Diensterbringer wählt unter den Rückmeldungen den passenden Lookup Service aus und nutzt sodann dessen Treiber zur Registrierung beim ausgewählten Lookup Service. Er ruft dazu die vom Treiber angebotene Registrierungsmethode auf, woraufhin der Treiber den Aufruf über das Netzwerk an den Lookup Service überträgt und den Dienst in der geeigneten Weise dort anmeldet. Dieser erste Schritt ist in Abbildung 4.49 links oben skizziert.

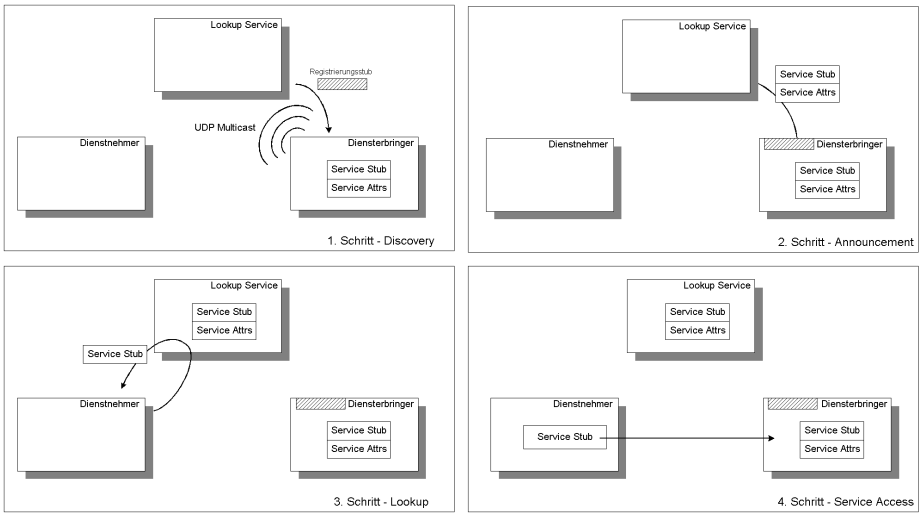


Abbildung 4.49: Discovery, Join, Lookup, Access: Die vier Schritte der Dienstnutzung in JINI

In Java werden Aufrufe von Methoden über das Netzwerk üblicherweise mittels

RMI (Remote Method Invocation) (siehe z.B. [Har00]) durchgeführt. Ähnlich wie bei anderen Verfahren zum entfernten Aufruf von Prozeduren bzw. von Methoden basiert auch Java RMI auf der Verwendung von so genannten Stubs, die als lokale Stellvertreter des entfernten Programmcodes fungieren. RMI Stubs bieten eine zum entfernten Objekt identische Aufrufschnittstelle, wobei der Aufruf einer Methode des Stubs zur Kodierung der Aufrufparameter und zu deren Übertragung zur eigentlichen Objektinstanz führt. Da es sich bei RMI Stubs um reinen Java Code handelt, können solche Stubs auf jeder Plattform zum Einsatz kommen, die über eine Virtuelle Java Maschine (Virtual Java Machine bzw. JVM) verfügt. Neuere Versionen der JVM sind in der Lage, Stubs selbständig zu serialisieren (d.h. für die Übertragung über Netzwerkstrecken vorzubereiten) und am Zielort in eine herkömmliche Objektinstanz zurückzuwandeln. Mit RMI bettet sich das Konzept entfernter Methodenaufrufe entsprechend nahtlos in den Sprachkern ein. Da JINI ausschließlich auf der Verwendung von Java Technologien beruht, basiert auch das skizzierte Treiberkonzept auf RMI. So handelt es sich bei dem vom Lookup Service übertragenen Treiber um einen RMI Stub, dessen Methoden der Diensterbringer nutzen kann.

Aber auch in seiner eigenen Anmeldung beim Lookup Service übergibt der Diensterbringer im einfachsten Fall einen RMI Stub an die Infrastruktur (auf weitere Varianten wird zu späterem Zeitpunkt in diesem Abschnitt noch näher eingegangen). Dieser Stub stellt den Kern der Registrierungsnachricht dar, welche der Diensterbringer an den Lookup Service übergibt. Darüber hinaus kann er außerdem weitere Informationen an die Vermittlungsinstanz übergeben, welche das Dienstangebot näher beschreiben helfen und die es so im weiteren Verlauf einem Dienstnehmer einfacher machen, das am besten geeignete Dienstangebot zu erkennen. Bei diesen Informationen, den so genannten Dienstattributen, handelt es sich einerseits um unveränderliche Eigenschaften des Dienstes, andererseits um dynamische, d.h. über die Zeit veränderliche Merkmale. Zur ersten Gruppe würden im Beispiel des Fotodruckers die maximale Auflösung oder die Anzahl an Druckfarben gehören. Zu den dynamischen Dienstattributen kann man hingegen die Anzahl an Druckaufträgen zum Zeitpunkt der Nachfrage zählen oder die Information über den Papiertyp im Papierfach. Auch Dienstattribute können in JINI als entfernte Objekte realisiert und als RMI Stubs an den Lookup Service übergeben werden. Dieser als „Announcement“ (Anmeldung) bezeichnete Schritt ist in Abbildung 4.49 rechts oben skizziert.

Will ein JINI Dienstnehmer auf einen Dienst zugreifen, so wendet er sich an den Lookup Service. Wie in der Grundform des Tradings formuliert er eine Anfrage, in der er den Typ des gewünschten Dienstes vorgibt und gegebenenfalls weitere Merkmale formuliert, welche der Dienst aufweisen soll. Als Merkmale kommen dabei die statischen und dynamischen Dienstattribute in Frage, welche der Diensterbringer bei seiner Registrierung an den Lookup Service übergeben hatte. Der Lookup Service überprüft anhand des spezifizierten Diensttyps und der angegebenen Attribute, ob eines der bei ihm registrierten Angebote zur Anfrage passt. Im Falle dynamischer Attribute kann er dabei zunächst den zum Zeitpunkt der Anfrage aktuellen Wert bestimmen. Da es sich bei Attributen um RMI Objekte handelt, ruft er dazu einfach eine vorgegebene Methode des Attributs auf, woraufhin das Attribut selbst in geeigneter Weise mit dem Diensterbringer Kontakt aufnimmt und den Wert erfragt. Konnte ein passendes Dienstangebot ermittelt werden, so wird der dazugehörige Stub an den Dienstnehmer übermit-

telt und kann daraufhin zur Kommunikation mit dem Diensterbringer verwendet werden. Der Schritt der Dienstermittlung wird als „Lookup“bezeichnet. Er ist in der Abbildung unten links dargestellt.

Da es sich bei dem vom Lookup Service übergebenen Dienstangebot um ein einfaches (entferntes) Java Objekt handelt, kann der Dienstnehmer hierauf unmittelbar zugreifen. Er stellt die notwendigen Parameter zusammen und ruft sodann die gewünschte Methode des Stubs auf. Vom Stub entgegengenommen werden die Parameter durch Java RMI serialisiert und dann an die im Diensterbringer residierende Objektinstanz übergeben. Dieser vierte und letzte Schritt ist in Abbildung 4.49 unten rechts dargestellt.

Wo liegen nun die Vorteile des beschriebenen Vorgehens gegenüber anderen Ansätzen? Bei der Verwendung einfacher RMI Stubs besitzt JINI gegenüber einigen anderen Ansätzen den Vorteil, dass das Konzept sich nahezu nahtlos in die Programmiersprache einpasst. Ist ein Objekt als „entfernt“ markiert, so kann es unmittelbar als Dienstschnittstelle fungieren, als solche an den Lookup Service übergeben und schließlich von einem Dienstnehmer genutzt werden. JINI erweitert in diesem Falle lediglich den ohnehin fest in den Sprachumfang von Java integrierten Verteilungsmechanismus RMI um die typischen Komponenten einer Middleware (Naming, Trading usw.). Andererseits ist Ähnliches auch über die zuvor beschriebene CORBA Middleware zu sagen. Auch sie ist – sofern auf die Nutzung in Java begrenzt – zumeist sehr gut in die Sprache integriert. Einen Vorteil bietet JINI in diesem Zusammenhang vor allem insofern, als Konzepte wie Trading und Leasing für den Entwurf verteilter Anwendungsszenarien *vorgeschrieben* werden, während diese bei CORBA und anderen Middleware Systemen allenfalls fakultativ sind. Auf diese Weise wird der Aufbau robuster und flexibler verteilter Anwendungen in JINI wesentlich einfacher, da alle beteiligten Komponenten gewisse Voraussetzungen erfüllen.

Seine besondere Stärke gegenüber anderen Middleware Ansätzen zeigt JINI jedoch beim Übergang von einfachen RMI Stubs zu komplexeren Treibern. Um diese Behauptung zu belegen, soll zunächst darauf eingegangen werden, welche Möglichkeiten bei der Realisierung von JINI Treibern zur Verfügung stehen. Die Aufgabe eines Treibers wurde oben bereits beschrieben. Er soll eine einheitliche Schnittstelle realisieren, über welche auf ein Gerät (oder allgemeiner: einen Diensterbringer) zugegriffen werden kann. Der einfachste Fall, wie dieses in JINI vonstatten gehen kann, wurde oben bereits beschrieben. Hierbei wird der Treiber durch einen RMI Stub repräsentiert, der an den Dienstnehmer übertragen wird und dessen Aufrufschnittstelle das Dienstangebot vor Ort darstellt. Als Protokoll zwischen Dienstnehmer und Diensterbringer wird in diesem Falle RMI eingesetzt, die lokale Schnittstelle des Diensterbringers entspricht der Schnittstelle des Treibers. Welche Möglichkeiten bestehen nun generell zur Realisierung von Treibern in JINI? Im wesentlichen sind folgende vier Fälle denkbar (vgl. Abbildung 4.50; für Realisierungsdetails siehe [LAB+00]):

Lokaler aktiver Fall : Der vom Diensterbringer an den Lookup Server übertragene Treiber besitzt keine externen Bezüge, sondern er wird ausschließlich in der Laufzeitumgebung des Dienstnehmers selbst verwendet. Dieser Sonderfall findet beispielsweise Verwendung bei der Realisierung bestimmter Sicherheitsverfahren. Der Treiber dient hierbei lediglich zur Berechnung von Signaturen, ohne dass dabei nach anfänglicher Übertragung des

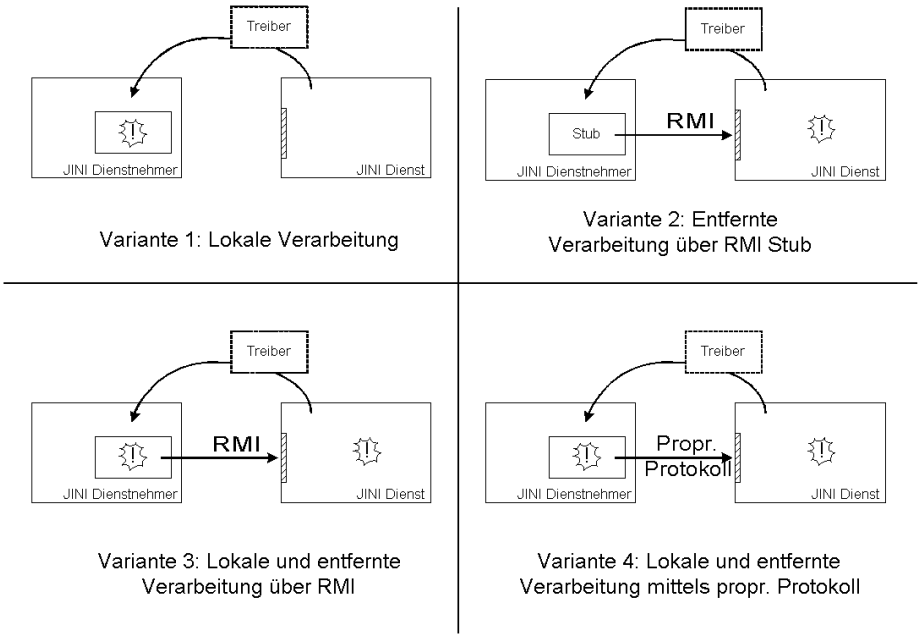


Abbildung 4.50: Die vier möglichen Treibervarianten von JINI

Codes in den Dienstnehmer eine weitere Interaktion mit dem Diensterbringer anfällt.

Passiver entfernter Fall via RMI : Dieser Ansatz entspricht dem oben geschilderten Fall. Beim Treiber handelt es sich hierbei um einen einfachen RMI Stub.

Aktiver entfernter Fall via RMI : Der Diensterbringer registriert nicht nur eine einfache Aufrufschnittstelle, deren alleinige Aufgabe in der Übertragung von Methodenaufrufen an die entfernte Objektinstanz besteht, sondern ein komplexes Objekt. Aufrufe bei diesem Objekt werden zunächst lokal in der Laufzeitumgebung des Dienstnehmers vorverarbeitet, bevor der eigentliche entfernte Aufruf über das Netzwerk zum Diensterbringer stattfindet. Die Aufrufschnittstelle des Treiberobjektes kann dabei von derjenigen des eigentlichen Dienstes abweichen und der Schritt der Vorverarbeitung einen tatsächlichen Mehrwert darstellen (z.B. eine Datenkompression oder die Verschlüsselung der Aufrufparameter). Die Netzwerkkommunikation findet in diesem Fall weiterhin mittels RMI statt.

Aktiver entfernter Fall ohne RMI : Wie im vorigen Fall handelt es sich bei dem vom Diensterbringer übertragenen Treiber um ein komplexes Objekt, das dem Dienstnehmer eine Aufrufschnittstelle präsentiert, über welche jener den Dienst in Anspruch nehmen kann. Anders als vorher basiert die interne Kommunikation mit dem Diensterbringer nunmehr jedoch nicht auf RMI sondern kann in einem beliebigen Protokoll erfolgen.

Insbesondere im dritten und vierten Fall birgt JINI eine erheblich größere Flexibilität als andere Middleware Systeme. So ist der Treiber nicht darauf be-

schränkt, die Daten an den Diensterbringer zu übertragen, sondern kann selbst aktiv werden. Dieses kann wie oben angegeben bedeuten, dass die Daten vor ihrer Übertragung modifiziert werden. Darüber hinaus ist es jedoch auch denkbar, dass der Treiber mit dem Benutzer selbst interagiert. Er könnte eine grafische Benutzungsschnittstelle darstellen, die dem Benutzer präsentiert wird. Über eine solche Schnittstelle könnte der Dienst dann z.B. konfiguriert oder sein Zustand bildlich dargestellt werden.

Gerade in Verbindung mit den oben geschilderten Eigenschaften der JINI Middleware (Leasing, Remote Events usw.) zeigt sich dabei der Nutzen im Kontext hochgradig dynamischer Anwendungsszenarien: Wird ein zuvor unbekannter Diensterbringer in eine JINI Umgebung eingebracht, so kann er seine Schnittstelle der Umgebung in Form eines aktiven Treibers übergeben. Handelt es sich dabei um eine grafische Schnittstelle (z.B. die Konfigurationsschnittstelle eines Fotodruckers), so ist es dem menschlichen Benutzer sofort möglich, den neuen Dienst anzusprechen. Auch völlig neuartige Dienste können so problemlos in das System integriert werden.

Aber auch im Falle von Diensten, die über Programmierschnittstellen gesteuert werden, ist die Integration neuartiger Schnittstellen möglich, so lange diese auf bekannten Schnittstellentypen basieren. Da JINI sich auf das von RMI bekannte Typsystem stützt, gelten die von RMI bekannten Regeln zu abgeleiteten Objekten. Sofern der neu registrierte Dienst also in seiner Schnittstelle auf einem zuvor bekannten Dienst basiert, kann ein Lookup Service ihn auch dort vermitteln, wo eine Instanz des bekannten Dienstes verlangt war. Finden dabei zunächst nur übernommene Methoden des neuen Dienstes Verwendung, können nach dessen Etablierung auch die übrigen Anteile der Schnittstelle von angepassten Klienten angesprochen werden.

Die eigentliche Interaktion mit dem Diensterbringer wird durch den aktiven Treiber verdeckt. Auf welche Weise Daten zwischen Treiber und entferntem Dienst ausgetauscht werden, ist vom Klienten nicht erkennbar. Solange die Schnittstelle des Treibers gleich bleibt, ist somit jederzeit eine Änderung oder Optimierung des Kommunikationsprotokolls möglich, ohne dass dieses auf der Seite des Klienten zu Problemen führen würde. Die hierbei notwendige Erneuerung des Treibers wird durch JINI interne Prinzipien vereinfacht. Wird der Dienst aus der Umgebung entfernt, so verfällt nach Ende der Leasing–Dauer dessen Treiber und dieser wird automatisch aus dem System entfernt. Auf Änderungen des Dienstangebots bzw. der Abstimmung zwischen Treiber und Dienst kann in diesem Augenblick durch Aktualisierung des Treibers reagiert werden.

4.4.4 Unterstützung mobiler Diensterbringer durch JINI

Von besonderem Interesse für die Nutzung von JINI bei der Integration mobiler Diensterbringer in verteilte Anwendungsszenarien ist nun die Tatsache, dass die eigentliche Kommunikation des Dienstnehmers mit dem Diensterbringer in oben skizzierter Weise hinter der äußeren Schnittstelle des Treibers versteckt wird. Letztlich ist es so ausschließlich Sache des Diensterbringers selbst zu entscheiden, auf welche Weise der Dienstnehmer mit ihm Kontakt aufnehmen soll. Wie bereits oben beschrieben, kann dieses per RMI geschehen. Während dies für (hinsichtlich ihres Kommunikationsumfelds) einfache Szenarien ausreichen sollte, ist es

im Zusammenhang mobiler Anwendungsfälle vermutlich notwendig — und dank des Treiberkonzepts von JINI darüber hinaus recht einfach möglich —, von einer einfachen RMI Kommunikation abzuweichen und effizientere Kommunikationsmechanismen einzusetzen[18]. So könnte bei Einsatz im Funk–LAN anstelle des herkömmlichen RMI das speziell auf den Einsatz in drahtlosen Netzen hin optimierte *Monads RMI* [CKRH00] zum Einsatz kommen, ohne dass der Dienstnehmer hiervon Kenntnis erhielte. In diesem Falle würde der vom Diensterbringer angebotene Treiber die notwendige Vorverarbeitung (Kompression, Reduktion der Protokollschritte usw.) übernehmen und mit der entsprechend angepassten Schnittstelle der Gegenstelle kommunizieren.

Doch auch in anderer Hinsicht eignet sich JINI sehr gut zur Integration mobiler Diensterbringer. In einer Vielzahl von Fällen handelt es sich bei mobilen Diensterbringern um Geräte zur Erfassung von Daten oder Überwachung bzw. Steuerung von Prozessen. Die hierbei eingesetzten Sensoren und Aktoren zeichnen sich durch ihre geringe Größe aus, die wenig Raum für die Integration übermäßig komplexer Rechensysteme lässt. In solchen Anwendungsfällen sind es tatsächlich eher die Klienten, in denen die größere Rechenleistung zu finden ist. So könnte zur Verarbeitung der durch mobile Sensoren erfassten Daten ein leistungsstarker Prozessrechner zum Einsatz kommen. JINI bietet hier den Vorteil, dass der Treiber nicht an die Nutzung von RMI und letztlich nicht einmal an die Verwendung herkömmlicher Netzwerkprotokolle gebunden ist. So könnte der in den Prozessrechner geladene Treiber über ein einfaches Protokoll, basierend auf TCP/IP Sockets (vgl. [Com98] bzw. [Ste90, Ste92, Ste94]), oder einen leichtgewichtigen Aufsatz hierzu, wie z.B. das *Simple Network Management Protocol* (SNMP) [Sta99], kommunizieren oder hierfür eine proprietäre Funkschnittstelle verwenden. Über den Umweg des Treibers können die angeschlossenen Sensoren dabei wie herkömmliche Java Objekte angesprochen werden.

Es lässt sich feststellen, dass sich JINI mit seinem auf Code–Migration basierenden Konzept zur Realisierung von Interaktionsbeziehungen zwischen Diensterbringern und Dienstnehmern offensichtlich gut für den Einsatz zur Integration mobiler Diensterbringer in verteilte Anwendungsumgebungen eignet. Es stellt somit scheinbar eine ideale Ergänzung zu dem im ersten Teil des Kapitels beschriebenen Ansatz zur Integration mobiler Dienstnehmer durch Einführung einer Abstraktionsschicht zwischen Applikationslogik und Präsentationsschicht dar.

Andererseits zeigt sich bei näherer Betrachtung ein elementares Problem beim Einsatz von JINI für ressourcenschwache Geräte: Während JINI in der oben gezeigten Weise durchaus die Verwendung effizienter, ressourcenschonender Protokolle zwischen Treiber und Diensterbringer ermöglicht und sogar die Abfrage dynamischer Attribute mittels proprietärer Basisprotokolle erlaubt, schreibt die Registrierung zwingend den Einsatz von RMI vor. Wäre es also grundsätzlich möglich, JINI Dienste *ohne* Verwendung von Java zu konstruieren (wie es bis auf weiteres für mobile Diensterbringer vorteilhaft sein dürfte), scheitert ein solches Unterfangen letztlich doch bereits am Initialisierungsschritt.

[18] Für eine Diskussion der Tauglichkeit von RMI in Funknetzen siehe z.B. [CKRH00].

4.4.5 Ein generischer Ansatz zur Migrations–basierten Integration

Soll JINI zu einer adäquaten Integrationsbasis für mobile Diensterbringer aufgebaut werden, so ist es notwendig, den Widerspruch zwischen seiner besonderen Eignung für die Realisierung ressourcenschonender Protokolle einerseits und der Notwendigkeit zur Nutzung von Java andererseits aufzulösen und JINI so auch für nicht Java–fähige Geräte nutzbar zu machen.

Sun Microsystems hat bereits bei der Entwicklung von JINI die Notwendigkeit vorausgesehen, nicht Java–fähige Diensterbringer in verteilte JINI Umgebungen einzubeziehen. In [Sun99b] werden eine Reihe von Merkmalen formuliert, die ein zu JINI konformer Dienst aufweisen muss, und es werden sodann Realisierungsalternativen aufgezeigt, wie dessen Integration geschehen kann.

Zunächst einmal wird festgestellt, dass jeder Diensterbringer in einer JINI Umgebung grundsätzlich eine zu Java kompatible Aufrufschnittstelle anbieten muss („*... be defined in terms of a datatype for the Java programming language...* “). Der Diensterbringer muss also innerhalb der Anwendungsumgebung als Java Objekt wahrgenommen und von anderen Java Objekten über eine herkömmliche Java API angesprochen werden können. Weiterhin wird festgestellt, dass es jedem JINI Dienst möglich sein muss, am JINI Discovery Protokoll teilzunehmen. Dienste müssen demgemäß Registrierungsobjekte erstellen können und diese über die hierfür angebotenen Schnittstellen an die Vermittlungseinheit übergeben können. Schließlich muss ein Dienst in der Lage sein, seine Registrierung zu verwalten. Er muss diese bei Bedarf erneuern können (Leasing) und Zugriffe auf dynamische Attribute erlauben.

Sun sieht hinsichtlich dieser Anforderungen drei Ansätze, wie ein nicht Java–fähiger Diensterbringer in eine JINI Umgebung eingebunden werden kann :

- **Verwendung einer speziellen virtuellen Maschine** : Ist ein Gerät nicht dazu in der Lage, eine vollwertige Java VM zu beherbergen, so könnte stattdessen im Prinzip eine reduzierte Variante zum Einsatz kommen. Während sich hierbei bezüglich der Treiberentwicklung bzw. Einbettung in den Java/JINI Kontext die geringsten Anpassungsprobleme ergeben, erfordert dieser Weg gleichzeitig einen enormen Realisierungsaufwand, da für jeden Gerätetyp eine eigene VM entworfen werden muss. Außerdem stellt gerade das benötigte RMI Protokoll sehr hohe Anforderungen an das Design der VM, so dass bisherige Anpassungen der Virtuellen Maschine an das mobile Umfeld auf RMI verzichteten – so z.B. die von Sun als offizielle VM für Mobilgeräte vorgesehene „K Virtual Machine“(KVM), die als Teil einer ganzen Java Entwicklungsumgebung für mobile Umgebungen („Java 2 Micro Edition“, J2ME) zum Einsatz kommt [Sun00].
- **Gruppierung von Geräten mittels einer gemeinsamen JVM** : Nach dieser Variante können sich mehrere Geräte die Kosten für die Teilnahme an einer JINI Umgebung teilen. Sie würden hiernach in ein leistungsstarkes Basisgerät (sog. „device bay“) eingesteckt und mit diesem über ein privates Protokoll kommunizieren. Das Basisgerät würde nach diesem Ansatz anstelle der eingesteckten Geräte als JINI Diensterbringer auftreten und als solcher die oben genannten Pflichten eines JINI Dienstes

wahrnehmen. Während dieser Ansatz im Falle von Geräten sinnvoll ist, die üblicherweise in Gruppen eingesetzt werden und dabei statisch sind (z.B. die Platten eines RAID Systems), scheint er nicht praktikabel im Falle von Geräten, die eher einzeln oder gar mobil zum Einsatz kommen (z.B. Drucker in einem Bürogebäude oder PDAs).

- **Gruppierung von Geräten unter Verwendung einer gemeinsamen Java VM über ein Netzwerk** : Dieser Ansatz stellt eine Variation des zuvor genannten Vorgehens dar. Auch in diesem Fall nutzen mehrere Geräte eine gemeinsame Java VM. Während die Geräte zuvor jedoch direkt mit dem Basisgerät verbunden sein mussten, ist das nach diesem Ansatz nicht notwendig: Die Kommunikation mit dem Basisgerät findet über Netzwerkstrecken und unter Verwendung des TCP/IP Protokolls statt. Während dieser Ansatz den Vorteil größerer Flexibilität hinsichtlich der Positionierung des Basisgeräts bietet, erscheint er als nachteilig im Einsatz in Ad–hoc Netzwerken. So ist die Verwendung kleiner Geräte nach diesem Ansatz nur dann möglich, wenn die Netzwerk–VM als zusätzlicher Dienst der Infrastruktur vorhanden ist. Gerade in kleinen oder dynamischen Netzwerken ist dieses von Nachteil.

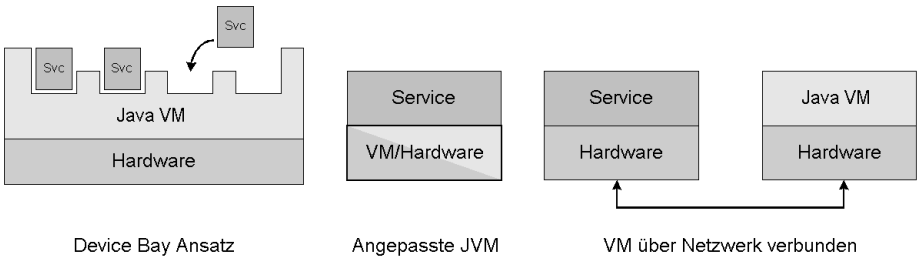


Abbildung 4.51: Mögliche Ansätze zur Einbindung nicht Java–fähiger Geräte in JINI

In Abbildung 4.51 sind die drei Alternativen nochmals skizziert. Im Sinne einer vereinfachten Darstellung wurde in der Abbildung auf eine Trennung zwischen VM und JINI Adapter–Code verzichtet. Es sei also angenommen, dass auf der eigentlichen Virtuellen Maschine jeweils noch der für eine Teilnahme an der JINI Umgebung notwendige Java Code vorhanden ist.

4.4.5.1 Integration mittels Surrogaten

Sun setzt in der Unterstützung ressourcenschwacher bzw. mobiler Geräte auf eine Kombination aus der Nutzung proprietärer Protokolle zwischen einem Stellvertreter (wie im Falle der Device Bay) und der Möglichkeit zur Nutzung von Netzwerkstrecken zur Kopplung der Geräte mit ihrem Basisdienst (wie im Falle der Gruppierung von Geräten über das Netzwerk). Das hieraus resultierende Konzept der Nutzung von *Surrogaten* [Tho01] soll im Folgenden kurz beschrieben werden, da es den derzeit vorgeschriebenen Ansatz zur Integration nicht java–fähiger Geräte in eine JINI Umgebung darstellt.

Kern des Ansatzes ist die Etablierung eines so genannten *Surrogate Host* , der als Teil der JINI Infrastruktur neben den vorhandenen Lookup Service tritt. Aufgabe des Surrogate Host ist es, als Plattform für die eigentlichen Surrogate zu dienen, die von den unterstützten Geräten an den Surrogate Host übergeben werden. Als Surrogat wird der Treiber verstanden, der zur Kommunikation mit dem Gerät verwendet wird. Der eigentliche Surrogate Host besitzt dabei kein explizites Wissen über die unterstützten Gerätetypen und deren Kommunikationsprotokolle. Diese Aufgabe übernehmen vielmehr spezielle Softwareadapter (sog. *Interconnect Adapter*), die für jede Geräteklasse in den Surrogate Host eingebunden werden und als Mittler zwischen Host und Gerät fungieren. Während das Protokoll zwischen Host und Adapter durch die JINI Spezifikation vorgegeben ist, hängt die Interaktion zwischen Adapter und Gerät von den Fähigkeiten des Geräts ab. Es wird einzig vorausgesetzt, dass der Adapter folgende Fähigkeiten aufweisen muss:

- Er muss in der Lage sein, Geräte zu erkennen und zu identifizieren, die über den von ihm betreuten Kanal angebunden sind.
- Er muss erkennen können, ob ein zuvor registriertes Gerät sich aus dem Einzugsbereich des unterstützten Kanals entfernt und den Surrogate Host hiervon informieren.
- Er muss das vom Gerät übertragene Surrogat annehmen und dieses an den Surrogate Host weiterleiten können.

Leider ist weder der Aufbau des Interconnect Adapters noch seine Schnittstelle zum Surrogate Host bzw. zum Gerät derzeit vollständig spezifiziert, so dass bestenfalls Vermutungen angestellt werden können, wie die Kopplung zwischen den beteiligten Entitäten vonstatten gehen soll (vgl. [LAB+00], S. 460). Der Aufbau des Surrogate Host ist in Abbildung 4.52 skizziert.

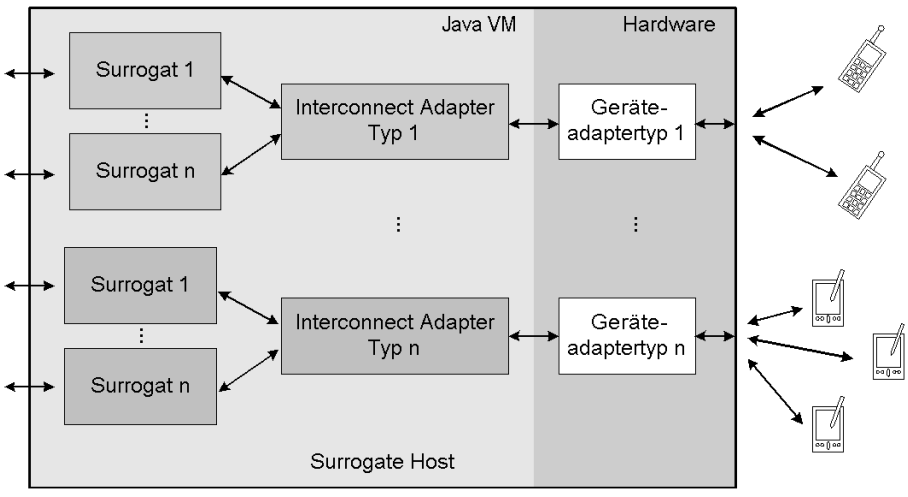


Abbildung 4.52: Anatomie des Sun Surrogate Host

In einem einzigen Surrogate Host könnten prinzipiell beliebig viele Adapter zusammengefasst sein, z.B. für SCSI, USB, WaveLAN et cetera. Über den jeweils

passenden Interconnect Adapter kann ein Gerät sich an den Surrogate Host wenden und ihm ein Surrogat (genauer: Surrogat–Paket bzw. „surrogate bundle“) übergeben, das als Stellvertreter des Gerätes innerhalb des Surrogate Host zum Einsatz kommen soll. Surrogat–Pakete stellen eine Kombination aus eigentlichem Surrogat und ergänzenden Daten wie Icons, Hilfstexten et cetera dar. Da das Gerät über keine eigene Java VM verfügt, ist der Inhalt des Surrogat–Pakets nicht von ihm interpretierbar, sondern stellt einen opaken Datenblock dar, der im Zuge der Herstellung bzw. Initialisierung des Gerätes eingespeist wird.

Das Surrogat stellt schließlich den eigentlichen Stellvertreter des Geräts in der JINI Umgebung dar. Das Surrogat wird nach Übernahme durch den Interconnect Adapter vom Surrogate Host instantiiert und bietet fortan eine Schnittstelle, über welche der Surrogate Host mit dem Gerät interagieren kann. Auch diese Schnittstelle ist derzeit noch nicht vollständig spezifiziert. Es ist lediglich vorgeschrieben, dass der Surrogate Host in der Lage sein muss, das Gerät über seinen Stellvertreter zu aktivieren und zu deaktivieren. Nicht klar ist jedoch, wie das Gerät durch den Surrogate Host hindurch mit der JINI Umgebung kommunizieren können soll und wie es sich letztlich in die Umgebung einbetten soll. Auch in diesem Bereich ist die Entwicklung des Surrogate Konzepts noch nicht abgeschlossen (vgl. [LAB+00], S. 460ff).

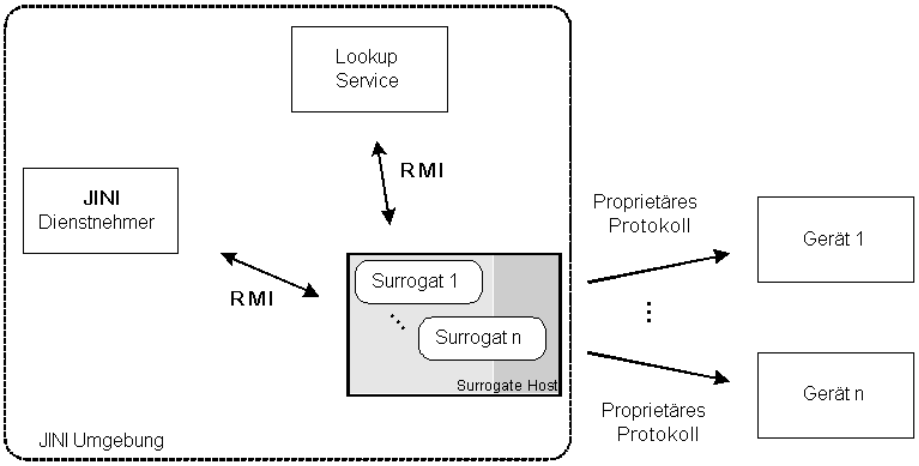


Abbildung 4.53: Der Surrogate Host im Kontext einer JINI Umgebung

Suns Surrogate Konzept ist in Abbildung 4.53 nochmals verbildlicht. Der Ansatz lässt sich am ehesten als ein *„One Face to the Customer“*–Verfahren bezeichnen. Danach bildet der Surrogate Host den Abschluss der JINI Umgebung in Richtung der nicht Java–fähigen Geräte. In Richtung der JINI Umgebung kommuniziert der Surrogate Host ausschließlich mittels JINI konformer Protokolle, während er in Richtung der unterstützten Geräte auf proprietäre Protokolle zurückgreifen kann. Aus der Sicht eines JINI Dienstnehmers bzw. des Lookup Service erscheint das Gerät dabei wie ein herkömmlicher JINI Dienst.

Die Einführung von Surrogate Hosts ermöglicht es auf universelle und flexible Art, nicht Java–fähige Geräte in JINI Anwendungsszenarien einzubinden. Die Assoziation von Surrogaten und Interconnect Objekten ermöglicht es, alle geräte-

spezifischen Aspekte vor dem Surrogate Host selbst zu verbergen, so dass sich durch die dabei entstehende zweifache Abstraktion (JINI Umgebung/Surrogate Host und Surrogate Host/Interconnect Adapter) ein insgesamt sehr robustes Integrationsmodell ergibt. Leider weist der Ansatz einen vorläufigen und einen konzeptionellen Mangel auf:

1. Derzeit ist der Ansatz noch nicht vollständig ausgearbeitet und daher nicht für den Einsatz in konkreten Anwendungsszenarien verfügbar – ein temporärer Mangel, der kurz– bis mittelfristig überwunden werden dürfte.
2. Die Einbindung von Geräten setzt die Existenz eines Surrogate Hosts und dessen Initialisierung mit den entsprechenden Interconnect Adaptern voraus. Die Verwendung von Surrogate Hosts eignet sich demgemäß nur bedingt für den Einsatz in mobilen Netzwerken bzw. Ad–Hoc Netzwerken.

Primär mit dem Ziel, dem letztgenannten Mangel zu begegnen, wurde im Rahmen dieser Arbeit ein alternativer Ansatz entwickelt, um nicht Java–fähige Geräte in den Kontext von JINI Anwendungsumgebungen einbinden zu können. Der Schwerpunkt lag also insbesondere auf der Unterstützung mobiler ressourcenschwacher Diensterbringer.

Im Übrigen begann die Konzeption des im Rahmen dieser Arbeit entwickelten Ansatzes der opaken Treiber zu einem Zeitpunkt, als Sun das Konzept der Surrogate Hosts noch nicht vorgestellt hatte[19]. Es ergab sich also auch im Sinne des erstgenannten Punktes ein unmittelbarer Handlungsbedarf.

4.4.5.2 Nutzung opaker Treiber als Integrationsmechanismus

Bei der Beschreibung des Surrogat–Konzepts wurde von Surrogat–Paketen gesprochen, die im Gerät zwischengespeichert und nach Aufbau einer Kommunikationsverbindung mit dem Surrogate Host an diesen übergeben werden. Als Surrogat–Paket wurde die Kombination aus dem eigentlichen Surrogat–Objekt und weiteren Daten bezeichnet, die in geeigneter Form miteinander verbunden sind. Das eigentliche Surrogat ist dabei ein Java Objekt, das als Stellvertreter des Geräts im Surrogate Host fungiert.

Ein ähnlicher Ansatz hinsichtlich der Nutzung von Stellvertretern wurde auch in dieser Arbeit gewählt. Wie im Surrogate Konzept wird auch in diesem Falle ein Stellvertreterobjekt vom Gerät uninterpretiert als Datenblock oder Binärdatei vorgehalten. Dieses wird bei Bedarf an die JINI Umgebung übergeben, dort aktiviert und tritt dann als Stellvertreter des Geräts mit anderen Teilnehmern der Umgebung in Kontakt. Aufgrund der Tatsache, dass das Stellvertreterobjekt die Rolle eines gewöhnlichen JINI Treibers übernimmt, als Java–Objekt jedoch vom Gerät selbst lediglich als ein einfacher (opaker) Datenblock angesehen, gespeichert und weitergeleitet werden kann, wird das Objekt als *opaker Treiber* bezeichnet.

Welche Entwurfsziele standen bei der Konzeption der opaken Treiber im Vordergrund?

[19] Die Arbeiten hieran wurden im weiteren Verlauf dann unabhängig und parallel zu dem von Sun favorisierten Ansatz fortgeführt.

- Das Konzept sollte bei der Integration von Endgeräten auf die Verwendung von Java auf der Seite der integrierten Diensterbringer verzichten können – sowohl bei der Registrierung als auch bei der darauf folgenden Nutzung durch Teilnehmer der JINI Umgebung.
- Das Konzept sollte ohne die Einführung eines weiteren Dienstes in die JINI Infrastruktur auskommen. Die notwendige Unterstützung sollte ausschließlich vom Lookup Service selbst und gegebenenfalls vom Dienstnehmer erbracht werden.
- Der nicht Java–fähige Diensterbringer sollte als vollwertiger JINI Dienstleister in Erscheinung treten können und hierbei neben der eigentlichen Registrierung eines JINI Treibers auch Mechanismen des Leasings und der entfernten Ereignisse beherrschen.
- Aus der Sicht der übrigen Dienstleister soll kein Unterschied zwischen herkömmlichen JINI Diensten und den mittels opaker Treiber realisierten nicht Java–fähigen Diensten erkennbar sein. Herkömmliche Diensterbringer sollen ohne Modifikation mit dem Lookup Service in Kontakt treten können.
- Aus der Sicht der übrigen Diensterbringer soll kein Unterschied zu einer herkömmlichen JINI Infrastruktur erkennbar sein. Insbesondere soll die Kommunikation mit dem Lookup Service in gewohnter Weise erfolgen.

Dabei wird die Nutzung von TCP/IP als Basisprotokoll *aller* beteiligten Kommunikationspartner vorausgesetzt und kein Versuch unternommen, eine direkte Einbeziehung von Protokollen wie SCSI, USB, Bluetooth usw. zu machen. Da TCP/IP (bzw. RMI) derzeit ohnehin als Vorbedingung für den Aufbau von JINI Netzwerken angesehen werden muss und ein Treiber nach seiner Integration in den Dienstnehmer zumeist auf TCP/IP als Basisprotokoll zum Datenaustausch mit dem Diensterbringer zurückgreift, ist diese Einschränkung jedoch als zweitrangig für den Einsatz in Ad–hoc JINI Netzwerken anzusehen.

Bei der Unterstützung nicht Java–fähiger Diensterbringer müssen folgende Phasen betrachtet werden:

Initialisierungsphase : Auffinden von Vermittlungsinstanzen durch den Diensterbringer

Registrierungsphase : Anmeldung des Dienstangebots sowie der statischen bzw. dynamischen Dienstattribute

Attributanfrage : Aktualisierung der dynamischen Dienstattribute im Zuge einer Anfrage beim Lookup Service

Nutzungsphase : Kontaktaufnahme und hierauf folgende Nutzung des Dienstes durch einen Dienstnehmer

Über diese vier Phasen hinaus muss der Diensterbringer weiterhin in der Lage sein, die Registrierung seines Dienstangebots im Bedarfsfall zu erneuern (d.h. er muss das hiermit verbundene Leasing–Protokoll unterstützen) und das Angebot

schließlich vor seinem kontrollierten Ausscheiden aus dem Netzwerk zurückzuziehen.

An mehreren Stellen wird dabei beim Durchlaufen der vier Phasen bisher auf die Nutzung von RMI zurückgegriffen, so dass der Lösungsansatz vor allem den Übergang zu einem einfachen Protokoll zum Ziel haben muss. Durch Erweiterung des Lookup Service um ein socket–basiertes Registrierungsprotokoll ist dieser Schritt zunächst recht einfach durchzuführen.

Die Stärke von JINI liegt in der Fähigkeit, bei der Definition von Dienstangeboten neben herkömmlichen passiven Schnittstellen zu entfernten Diensten auch aktive Treiber zu erlauben. Bei derartigen Treibern handelt es sich, wie bereits gesagt, um reguläre Java Objekte, die vom Diensterbringer zum Lookup Service und von dort zum Dienstnehmer übertragen werden. Sie werden in der Laufzeitumgebung des jeweiligen Kommunikationsknotens instantiiert und können dort eine Vorverarbeitung vornehmen. Wie entstehen nun opake Treiber, die von Diensterbringern an den Lookup Service übergeben werden? Bei Nutzung von RMI werden diese Objekte im Augenblick der Kontaktaufnahme auf die Übertragung über das Netzwerk vorbereitet (d.h. in einen Datenstrom serialisiert), zur Gegenstelle übertragen und dort von der VM aus dem empfangenen Datenstrom wieder in Objekte zurückübersetzt. Das nicht Java–fähige Gerät muss dagegen auf vorbereitete Datenpakete zurückgreifen, die dem Datenstrom eines Treiberobjekts entsprechen und die im Vorweg von einem Javaprogramm erzeugt und in den Festspeicher des Geräts eingebracht wurden. Dieses Programm nutzt zur Serialisierung die vorhandenen Routinen der VM, die auch im Rahmen der RMI Kommunikation eingesetzt werden.

Ein Problem stellen bei diesem Vorgehen die veränderlichen Daten dar, die erst zum Zeitpunkt der Serialisierung fixiert werden und die insbesondere bei der Kodierung von Dienstattributen eine wichtige Rolle spielen. Für diese Objekte können keine vorgefertigten Datenpakete erstellt werden, da sich ihre Werte ja üblicherweise erst zur Laufzeit des Diensterbringers ergeben. Es ist somit nicht ausreichend, den Lookup Service um das eigentliche Übertragungsprotokoll zu erweitern, mit dem er Daten empfangen und in Java Objekte umsetzen kann, sondern es muss gleichzeitig ein Mechanismus geschaffen werden, der nach der Übertragung für eine Aktualisierung ungültiger Attributwerte sorgt.

Schließlich muss dafür gesorgt werden, dass im weiteren Verlauf eine kontinuierliche Kommunikation zwischen Diensterbringer, Dienstnutzer, Dienstangebot und Lookup Service möglich ist. Hierbei kommen wiederum aktive Treiber zum Einsatz: Alle Aspekte, die die Verwendung von RMI voraussetzen (wie z.B. das Leasing–Protokoll), werden nicht durch den Diensterbringer selbst, sondern durch das ihn stellvertretende Dienstangebot ausgeführt. Im Bedarfsfall kann das Dienstangebot dabei mit dem Diensterbringer über ein einfaches, privates Protokoll Kontakt aufnehmen.

4.4.5.3 Realisierung opaker Treiber

Die Aufgabe eines JINI Treibers besteht darin, als Stellvertreter eines Dienstes in der Laufzeitumgebung des Dienstnehmers aufzutreten. Er wird durch den Dienstnehmer wie ein herkömmliches Java Objekt instantiiert und kann dann über die angebotene Aufrufschnittstelle angesprochen werden. Während er dem

Dienstnehmer gegenüber als ein einfaches, lokales Objekt erscheint, obliegt es seiner Verantwortung, bei seiner Instantiierung eine Verbindung zum Dienst herzustellen und sodann die Aufrufe durch den Dienstnehmer an den Dienst weiterzuleiten.

Üblicherweise findet die Kommunikation zwischen JINI Treiber und Dienst mittels RMI statt. Der Dienst bietet hierzu eine Aufrufschnittstelle an, über welche der Treiber Daten übergeben oder abfordern kann. Auf der Basis des RMI Protokolls wird hierdurch ein anwendungsspezifisches Protokoll realisiert. Im Falle des opaken Treibers muss RMI als Basisprotokoll durch ein einfaches Socketprotokoll ersetzt werden. Während der Treiber dem Dienstnehmer weiterhin eine vollwertige Java Schnittstelle anbieten muss, wird für die Kommunikation mit dem Dienst also auf den Austausch einfachster Protokollnachrichten zurückgegriffen. Für den Datenaustausch bestehen dabei keine prinzipiellen Einschränkungen, solange sie vor dem Hintergrund knapper Ressourcen des Dienstes als sinnvoll erscheinen. Als mögliche Basisprotokolle sind daher HTTP, XML–RPC [ABK+00] oder auch SOAP (Simple Object Access Protocol) [BEK+00] denkbar. Jedes dieser Protokolle basiert auf einfachen Zeichenketten (bzw. XML Tags), in denen Protokollnachrichten kodiert werden. Während selbst einfache Microcontroller zur Verarbeitung solcher Zeichenketten in der Lage sind, zeigt sich unter anderem an SOAP und dem XML–RPC, dass auf ihrer Basis ein vollwertiger RPC Mechanismus realisiert werden kann.

Wie weiter oben dargestellt wurde, stellt die Verwendung alternativer Protokolle eine allgemein anerkannte Konstruktionsvariante eines JINI Treibers dar. Die Entwicklung eines opaken Treibers unterscheidet sich also zunächst nicht von der Entwicklung eines herkömmlichen JINI Treibers. Lediglich der letzte Entwicklungsschritt, also die Vorbereitung des Treibers für seine Übertragung zum Lookup Service muss angepasst werden. Erfolgt im Falle eines JINI Treibers die Serialisierung zur Laufzeit durch die Java Umgebung, so wird sie beim opaken Treiber vorgezogen. Mit Hilfe eines in Java geschriebenen Hilfsprogrammes wird der Treiber für die vermeintliche Weiterleitung via RMI in einen Datenblock überführt und dann abgespeichert. Danach kann er dann, beispielsweise als Binärdatei, in den Diensterbringer übertragen werden.

Und noch in einem weiteren Detail weicht der opake Treiber von einem herkömmlichen JINI Treiber ab: Da die vorgezogene Serialisierung keine Berücksichtigung dynamischer Attribute erlaubt, kann bei der Übergabe des Treibers an den Lookup Service nicht auf den Zustand des Diensterbringers eingegangen und das Dienstangebot entsprechend parametrisiert werden. Der opake Treiber muss diesen Schritt selbst übernehmen, nachdem er in der Laufzeitumgebung des Lookup Service aktiviert worden ist. Er verwendet dafür eine abgeänderte Deserialisierungsroutine, die eine Verbindung mit dem Diensterbringer aufnimmt und die aktuellen Werte von diesem erfragt, bevor das Dienstangebot als vollständig wiederhergestellt an den Lookup Service gemeldet wird. Die dafür notwendige Routine wird in den Treiber fest integriert und als ein Teil von ihm vorab serialisiert, so dass sie nach Übertragung an den Lookup Service sofort zur Verfügung steht.

Schließlich muss der Treiber seine eigene Registrierung beim Lookup Service übernehmen und diese im weiteren Verlauf aktualisieren. Er tut auch dieses mittels Hilfsroutinen, die, ähnlich wie die Deserialisierungsroutine, fest in jeden

Treiber integriert und nach seiner Instantiierung sofort verfügbar sind.

Im Übrigen handelt es sich beim opaken Treiber aber ab dem Zeitpunkt der Registrierung um ein vollwertiges JINI Dienstangebot. Dienstnehmer können seine Attribute erfragen, können es vom Lookup Service laden und lokal aktivieren. Sie können auf seine Methoden zugreifen und den Dienst in der erwarteten Weise in Anspruch nehmen.

Auf der Seite des Lookup Service ist hauptsächlich eine Anpassung der Discovery und Join Protokolle notwendig. Erfolgt im herkömmlichen Fall die Registrierung über ein RMI Objekt, das vom Lookup Service zum Diensterbringer übertragen wird und das dann die Aufrufe des Diensterbringers entgegennimmt, so muss an dessen Stelle nunmehr ein Socket–Protokoll treten. Ähnlich wie im unmodifizierten Protokoll wird zunächst per Multicast die Existenz passender Lookup Service Instanzen ermittelt. Die eigentliche Anmeldung erfolgt jedoch nicht mehr per RMI, sondern über das HTTP Protokoll: Der Lookup Service übernimmt die Rolle eines Klienten, der Diensterbringer die eines Web–Servers. Über die zwischen beiden aufgebaute HTTP–Verbindung wird der opake Treiber in den Lookup Service geladen, er wird instantiiert und schließlich in der oben skizzierten Weise aktiv.

Hierbei wird im Kommunikationsaufbau auf die Protokollnummer zurückgegriffen, die am Anfang jeder Kommunikation zwischen den Kommunikationspartnern ausgetauscht wird. Anhand dieser Nummer können die Partner erkennen, ob sie dieselbe Protokollversion beherrschen. Im modifizierten Lookup Service tritt neben die bislang bekannte Protokollversion eine zweite, die nicht auf RMI basiert. Will ein nicht Java–fähiger Dienst sich bei einem Lookup Service registrieren, so übergibt er die neue Protokollnummer. Beherrscht der angesprochene Lookup Service das Registrierungsprotokoll für ressourcenschwache Diensterbringer, so antwortet er mit Informationen über den Zugriffspunkt, den er hierfür anbietet. Mit diesem Zugriffspunkt kann der Diensterbringer dann eine Socket–Verbindung herstellen und seinen opaken Treiber übergeben. Verfügt ein Lookup Service andererseits nicht über die notwendige Erweiterung zur Kommunikation mit ressourcenschwachen Geräten, so sendet er einfach eine negative Rückantwort. Der Diensterbringer muss in diesem Fall den Schritt der Discovery wiederholen, um so schließlich einen passenden Lookup Service zu finden.

4.4.5.4 Vorteile des Konzepts

Wie bereits weiter oben beschrieben, besitzt JINI ein großes Potential für die Realisierung dynamischer Anwendungsszenarien. Zeigen Middleware Systeme wie CORBA, DCE oder AnsaWare [May94, May96] insbesondere in statischen Konstellationen ihre Stärke hinsichtlich Sicherheit, Robustheit und Verlässlichkeit, so wird ihr Einsatz in Ad–Hoc Netzwerken dadurch erschwert, dass sie oftmals komplizierte Konfigurationen der beteiligten Kommunikationsknoten voraussetzen und sie selten adäquate Verfahren für die Verwaltung dynamischer Beziehungen vorsehen. JINI setzt dem mit seinem Ansatz der Verwendung migrierender Treiber und zeitgebundener Referenzen ein neuartiges, leichtgewichtiges Konzept entgegen. Gerade für spontane Interaktionen, bei denen der aufwendige Austausch von Treiberprogrammen nicht akzeptabel ist, wird ein Basismechanismus geschaffen, mit dem zuvor inkompatible Geräte einfach miteinander in

Kontakt treten können. Leasing und Remote Events sorgen schließlich dafür, dass auch hohe Dynamik und Fluktuation nicht die Konsistenz des Gesamtsystems schädigen können.

Während sich JINI somit besonders für den Einsatz im Kontext mobiler Anwendungsfälle zu eignen scheint, erweist sich die stringente Nutzung von Java Technologien als hemmend. Die Flexiblität, die der migrierende Treiber in die Kommunikation zwischen Dienstnehmer und Diensterbringer einführt, wird dadurch zunichte gemacht, dass an entscheidender Stelle im Protokoll der Einsatz von RMI (und somit Java) verbindlich vorgeschrieben wird. JINI ist damit für den überwiegenden Teil an kleinen Geräten mit beschränkten Ressourcen nicht einsetzbar, da für diese Geräte die Verwendung von Java ausgeschlossen wird.

Sun selbst hat mit der Einführung von Surrogaten versucht, diesem Mangel zu begegnen. Auf der Basis von Surrogate Hosts, Interzeptionsadaptern und Surrogaten können auch nicht Java–fähige Diensterbringer an einer JINI Umgebung teilnehmen. Der Ansatz bildet einen flexiblen Integrationsrahmen, der die Einsatzmöglichkeiten von JINI wesentlich erweitert. Gerade im Zusammenhang mit Ad–hoc Netzwerken weist er jedoch einen gravierenden Nachteil auf: Er fußt auf der Einführung eines weiteren Dienstes in die JINI Infrastruktur, was im Rahmen der spontanen Vernetzung von Geräten schwer wiegt.

Das hier vorgestellte Konzept der opaken Treiber ähnelt in gewisser Weise der von Sun favorisierten Nutzung von Surrogaten. Auch hierbei werden Treiber verwendet, die vom Diensterbringer selbst nicht interpretiert werden können und die erst in der Laufzeitumgebung eines Java–fähigen Kommunikationspartners aktiviert werden können. Im Gegensatz zu jenem Ansatz wird hierbei aber auf die Einführung eines eigenen Dienstes verzichtet und vielmehr durch minimale Modifikation des Lookup Service ein ähnliches Resultat erzielt, nämlich die Möglichkeit, nicht Java–fähige Diensterbringer in einer JINI Umgebung verwenden zu können. Der Ansatz nutzt dabei selbst intensiv die Möglichkeiten von JINI. Wo dieses notwendig ist, werden Java spezifische Aufgaben durch den zuvor instantiierten opaken Treiber erledigt und zwischen Diensterbringer und Dienstnehmer ein effizientes Protokoll eingesetzt, das den Dienst selbst nur minimal belastet.

4.5 Diskussion: Surrogate vs. opake Treiber

In Abschnitt 4.4.5.1 wurde die Verwendung von Surrogaten als ein anderer Ansatz zur Einbindung nicht Java–fähiger Diensterbringer in eine JINI Umgebung vorgestellt. Nun, nachdem dem Surrogat das Konzept opaker Treiber gegenübergestellt wurde, soll ein bewertender Vergleich beider Ansätze erfolgen.

In ihrem Ziel stimmen die beiden Integrationsansätze zunächst überein. Sie dienen der Erschließung nicht Java–fähiger Geräte für die Nutzung im Kontext einer JINI Umgebung. Das Surrogat–Konzept setzt dabei auf die Vermittlung zwischen Diensterbringer und JINI Umgebung durch einen Stellvertreter, den Surrogate Host, während im Fall der opaken Treiber die Interaktion direkt zwischen Diensterbringer und Lookup Service erfolgt. Insofern kann zunächst festgestellt werden, dass die hauptsächliche Last bei der Verwendung von Surrogaten

auf dem Surrogate Host liegt, während diese bei der Verwendung opaker Treiber dem Lookup Service aufgebürdet wird.

In beiden Fällen liegt die Kernidee in der Verwendung von Treibern, die vom Diensterbringer selbst nicht interpretierbar sind. Diese sind im Festspeicher des Diensterbringers abgelegt und werden von diesem im Laufe der Registrierungsphase an den Kommunikationspartner übergeben. Erst in der Java VM eines Stellvertreters kann das Treiberobjekt aktiviert, bei der JINI Laufzeitumgebung registriert und letztlich an den Dienstnehmer übertragen werden. Bei der Verwendung von Surrogaten fällt diese Aufgabe dem Surrogate Host zu. Er muss die Kommunikation mit dem Diensterbringer aufbauen und von diesem den Treibercode nebst Attributen übernehmen. Im Anschluss ist es dann seine Pflicht, einen passenden Lookup Service zu ermitteln und das Angebot bei diesem zu registrieren. Im Fall der opaken Treiber wird dieses durch den Lookup Service erledigt. Er aktiviert den übernommenen Treiber und registriert ihn dann mitsamt der beschreibenden Attribute in seiner eigenen Datenbank.

In beiden Fällen wird auf ein zusätzliches Protokoll zwischen dem nicht java–fähigen Gerät und dessen Stellvertreter in der JINI Umgebung gesetzt. Für die opaken Treiber wurde dazu der Lookup Service dahingehend erweitert, dass er zusätzlich zur Verwendung von RMI ein ausschließlich auf TCP/IP basierendes Registrierungsprotokoll beherrscht, mit dessen Hilfe der Diensterbringer sein Dienstangebot an den Lookup Service übermitteln kann. Der Austausch von Dienstattributen erfolgt hingegen in einem privaten Protokoll, das lediglich zwischen Treiber und Diensterbringer bekannt sein muss. Beim Einsatz von Surrogaten wird hingegen ein Vorgehen favorisiert, das ausschließlich auf der Verwendung privater, d.h. dem Surrogate Host nicht offen gelegter, Protokolle basiert. Hierzu sind zu jedem Diensttyp Interconnect Adapter bereitzustellen, welche in den Surrogate Host eingebunden werden und welche die eigentliche Kommunikation mit dem Diensterbringer regeln. Nach außen bieten diese Adapter eine einheitliche Schnittstelle, so dass der Surrogate Host jederzeit modular um weitere Protokolle erweiterbar ist. Im Gegensatz zur Protokollerweiterung des Lookup Service ist die Verwendung von Interconnect Adaptern konzeptionell nicht an die Nutzung von TCP/IP als Basisprotokoll gebunden. Denkbar sind hier beliebige andere Protokolle wie Firewire, USB, das PCI Protokoll et cetera.

Gerade in diesem Zusammenhang ist ein wichtiger Aspekt, dass die Kommunikation zwischen Diensterbringer und Dienstnehmer im Surrogat–Ansatz vornehmlich über den Umweg des Surrogate Host läuft. Er tritt in der JINI Umgebung als echter Stellvertreter des eigentlichen Diensterbringers auf und leitet Befehle und Daten an das jeweilige Gegenüber weiter. Bei Nutzung von nicht auf TCP/IP basierenden Kommunikationsprotokollen ist dieses essentiell, bei Nutzung von TCP/IP basierten Protokollen stellt es lediglich eine Vereinfachung dar, da auch der Treiber hierbei direkt auf RMI zurückgreifen kann. Opake Treiber kommunizieren hingegen ausschließlich direkt mit dem Diensterbringer, was den Ansatz auf die Nutzung in TCP/IP Umgebungen einschränkt. Der Lookup Service fällt nach Registrierung des Dienstangebots als Stellvertreter aus der Kommunikationsbeziehung heraus und ist fortan nurmehr als Dienstvermittler aktiv.

Die durchgehende Verwendung einer dritten Instanz als Stellvertreter des nicht java–fähigen Geräts eröffnet im Surrogat Ansatz schließlich die Möglichkeit, neben Diensterbringern auch Dienstnehmer ohne eigene Java VM einzubeziehen.

Einen entsprechenden Interconnect Adapter vorausgesetzt ist es ihnen möglich, den Surrogate Host als ihren Stellvertreter im Anwendungsszenario zu nutzen und fortan durch ihn hindurch aktiv zu sein. Der Surrogate Host verbindet sich an ihrer Stelle mit dem Lookup Service, überträgt ihre Suchanfrage und übernimmt die gefundenen Dienstangebote. Er sorgt für die Einbindung des ausgewählten Treibers und vermittelt schließlich zwischen dem Treiber und dem nicht java–fähigen Dienstnehmer. Die Nutzung eventuell vom Treiber angebotener grafischer Schnittstellen ist dabei natürlich nur eingeschränkt möglich. Opake Treiber sind andererseits ausschließlich für den Einsatz in Diensterbringern zu verwenden. Da es in diesem Fall keinen permanenten Stellvertreter zwischen Diensterbringer und Dienstnehmer gibt, ist die Unterstützung nicht java–fähiger Dienstnehmer nicht möglich.

Während die Vorteile bei der Nutzung von Surrogaten vornehmlich in der Generizität des Ansatzes liegen, stehen bei der Verwendung von opaken Treibern die hohe Flexibilität sowie die effiziente Ausnutzung von Ressourcen im Vordergrund. Surrogate basieren zwingend auf der Existenz des Surrogate Host. Dieser muss neben dem Lookup Service in jeder Anwendungsumgebung vorhanden sein, in der nicht java–fähige Diensterbringer zum Einsatz kommen sollen. Für die spontane Vernetzung von Mobilgeräten bedeutet dieses, dass eine Kommunikation nur dann zu garantieren ist, wenn java–fähige Diensterbringer neben dem Lookup Service gleichzeitig auch einen Surrogate Host beherbergen. Die so quasi zustande kommende Verschmelzung beider Funktionalitäten führt jedoch zu einem dem Ansatz der opaken Treiber identischen Aufbau, wobei lediglich die stellvertretende Funktion des Lookup Service weiter ausgebaut ist. Auch führt die Verwendung von Surrogate Hosts zu einem stark erhöhten Administrationsaufwand. Da die Verbindung nicht java–fähiger Diensterbringer und Dienstnehmer nur unter Verwendung passender Interconnect Adapter möglich ist, muss vor erstmaliger Einbindung eines bestimmten Treibertyps zunächst ein korrespondierender Adapter installiert werden. Dieses steht dem Anspruch von JINI zur Schaffung administrationsfreier Anwendungsnetzwerke diametral entgegen. Andererseits stellt die Verwendung von Surrogaten einen sehr generischen Ansatz dar. Eine Einschränkung auf bestimmte Kommunikationsprotokolle zwischen Treiber und Diensterbringer besteht genauso wenig wie die Festlegung auf bestimmte Hardware–Voraussetzungen. Durch die Verwendung der Interconnect Adapter ist der Surrogate Host auch nachträglich noch zu erweitern, ohne dass hierfür seine Basisarchitektur zu ändern wäre. Dieses trägt letztlich zur Robustheit des Gesamtkonzepts bei. Der Ansatz opaker Treiber verzichtet hingegen auf die Indirektionsstufe eines eigenständigen Stellvertreters. Treiber werden vom Diensterbringer direkt beim Lookup Service angemeldet, und auch die Kommunikation zwischen Diensterbringer und Dienstnehmer erfolgt ohne Umwege direkt. Dieses bedeutet einerseits eine Festlegung auf TCP/IP als Basisprotokoll, befreit die JINI Umgebung jedoch andererseits von der Verpflichtung, einen zusätzlichen Stellvertreter bereitzustellen. Es wird lediglich ein um das angepasste Protokoll erweiterter Lookup Service vorausgesetzt. Zusätzliche spezielle Adapter müssen nicht in den Lookup Service integriert werden. Während das alternative Registrierungsprotokoll in sich vorgegeben ist, kann der Austausch von Attributen oder anderen Informationen zwischen Gerät und Treiber beliebig ausgearbeitet werden. Ein zusätzlicher Administrationsaufwand entsteht demgemäß nach Austausch des herkömmlichen durch den angepassten

Lookup Service nicht.

4.6 Zusammenfassung

Nachdem in den vorangegangenen Kapiteln die Grundlagen der verteilten und insbesondere der mobilen Datenverarbeitung erläutert und prinzipielle Strategien zur Integration mobiler Geräte in verteilte Systemumgebungen vorgestellt wurden, stellte dieses Kapitel den eigentlichen Kern der vorliegenden Arbeit dar. Es diente der Einführung von Konzepten, die zur Integration mobiler Diensterbringer sowie mobiler Dienstnehmer auf Ebene der Anwendungsschicht Verwendung finden können. Die Behandlung von Mobilität erfordert die Berücksichtigung von Aspekten wie Ressourcenknappheit, geringer Rechenleistung, variabler Verbindungsqualität, Geschlossenheit der Systeme hinsichtlich Anpassung oder Erweiterung usw. Diese Aspekte müssen ungeachtet der Funktion des zu integrierenden Geräts als Anbieter oder Nutzer von Daten beachtet werden. Beschränkt man sich in seiner Betrachtung auf die Anwendungsebene, so können gewisse Probleme wie Verbindungsabbrüche oder die Behandlung der Relokation von Kommunikationsknoten aus der Betrachtung ausgegrenzt werden. Bei den in diesem Kapitel entwickelten Konzepten stand demgemäß deren Anwendbarkeit auf der Anwendungsebene im Vordergrund.

Wie sich zeigte, unterscheiden sich im Detail die Anforderungen an die Integration mobiler Dienstnehmer deutlich von denen der Integration mobiler Diensterbringer. Während in beiden Fällen dieselben Kommunikationskanäle verwendet werden und dieselben Einschränkungen gelten, sind vollkommen eigenständige Konzepte zu entwerfen. Dieses Kapitel zerfiel dementsprechend in zwei Teile, die sich den spezifischen Problemen im Detail widmeten.

Zunächst wurde auf die Integration mobiler Dienstnehmer eingegangen. Es wurden wichtige Integrationsansätze vorgestellt, die auf Anwendungsebene anzuordnen sind. Ausgehend von einer Unterteilung in betriebssystembasierte Integration, applikationsserverbasierte Integration sowie Entwurfszeit–Transformation wurden die bestehenden Konzepte durchleuchtet und hinsichtlich ihrer Vor– bzw. Nachteile bewertet. Dabei wurde deutlich, dass existierende Ansätze zwar jeweils eine Teilmenge der zur Lösung der Fragestellung zu behandelnden Probleme angehen, in keinem der besprochenen Ansätze jedoch eine vollständige Lösung zu erkennen ist. So ist mit dem X11 Window System, mit RDP oder gar Puppeteer zwar eine entfernte Nutzung von Anwendungen über ein Netzwerk möglich, jedoch eine umfassende Modifikation des Endgerätes zur Unterstützung des jeweiligen Ansatzes vorauszusetzen. DataX, MDDS oder andere Ansätze aus dem Bereich der applikationsserverbasierten Integration ermöglichen durch den Einsatz abstrakter Zwischenrepräsentationen zwar die Transformation in eine gewünschte Zieldarstellung, sie bedingen jedoch eine entsprechende Vorbereitung des aus der Ferne zu nutzenden Anwendungssystems. Gleiches gilt schließlich auch für die Spielarten der Entwurfszeit–Transformation, die zwar die unkomplizierte Anpassung der Zielrepräsentation an die Bedürfnisse des zu unterstützenden Endgerätes ermöglichen, dieses jedoch nur zum Preis einer vorherigen Modifikation des Applikationskerns zulassen. Vor dem Hintergrund dieser Analyse entstand die Etablierung einer aktiven Abstraktionsschicht zwischen

Anwendungslogik und Präsentationslogik als zentrale Idee des vorgestellten Integrationsansatzes. Durch die Einführung eines modularen Unterstützungssystems zwischen Endgerät und Anwendungssystem lassen sich die vorteilhaften Eigenschaften der zuvor betrachteten Ansätze miteinander kombinieren und zu einer flexiblen Adaptionsschicht vereinen.

Als wesentliches Konzept wurde dabei zunächst die Nutzung eines als Introspektion bezeichneten Verfahrens vorgestellt, bei dem eine Analysekomponente in die laufende Anwendung eingreift und durch eine fortlaufende Analyse ihrer Datenstrukturen eine jederzeit aktuelle abstrakte Beschreibung ihrer Benutzungsschnittstelle erschließt. Die Analysekomponente beschränkt sich dabei auf eine rein strukturelle Analyse, während sie auf eine semantische Untersuchung verzichtet. Von grundlegender Wichtigkeit für den Introspektionsprozess ist es dabei, dass die resultierende Beschreibung weitestgehend von plattformspezifischen Aspekten der Schnittstelle bereinigt ist. Auf diese Weise ergibt sich eine erste Abstraktion somit darin, dass eine Unterstützung verschiedenster Ausgangsplattformen möglich ist, sofern die technische Realisierung der Schnittstellenanalyse den speziellen Eigenschaften des behandelten GUI–Systems angepasst wird.

Im Zuge der Beschreibung des Introspektionsansatzes wurde erläutert, dass GUI Systeme üblicherweise nach dem *Model–View–Controller* Prinzip aufgebaut sind, bei dem ein zugrunde liegendes Datenmodell (Model) durch eine oder mehrere Sichten (Views) repräsentiert wird und durch eine oder mehrere Kontrollinstanzen (Controller) gesteuert wird. Es wurde deutlich, dass die Analyse nur zu einer generischen Sicht, d.h. einer Art von View führt. Für eine vollständige Fernsteuerung ist jedoch nicht nur die entfernte Darstellung der Schnittstelle notwendig, sondern auch die Erfassung und Einspeisung von Befehlen, mit denen die Anwendung gesteuert wird. Als quasi spiegelbildliche Funktionalität zur Analyse wurde demgemäß die entfernte Steuerung als zweite wichtige Teilkomponente des Introspektionskonzepts identifiziert. Es wurde dargestellt, dass die wichtige Forderung nach weitgehender Transparenz des Ansatzes, d.h. die entfernte Nutzung der Anwendung, ohne dass diese für die Anwendung selbst erkennbar wäre oder gar von der Anwendung aktiv unterstützt werden müsste, sich durch einen Ansatz erreichen lässt, bei dem eine Steuerungsfunktion auf die Ereignisbehandlung der Anwendung insgesamt, bzw. auf die Ereignisbehandlung der einzelnen GUI–Elemente, abzielt. Es wurde verdeutlicht, dass sich im Zusammenhang mit dem hier vorgestellten Konzept insbesondere letztere Herangehensweise als vorteilhaft erwiesen hat. Entsprechend den vom Endgerät stammenden Befehlen können GUI–Elemente direkt aktiviert und benutzt werden, ohne dass aus der Sicht der Anwendung ein Unterschied zur direkten Nutzung über Maus und Tastatur erkennbar wäre. Gleichzeitig wird die Benutzungsschnittstelle der Anwendung durch die entfernte Nutzung nicht isoliert, sie steht also auch weiterhin für die lokale Bedienung zur Verfügung. Wie die Analysekomponente ist dabei auch die Steuerungskomponente derart angelegt, dass sie die systemspezifischen Realisierungsdetails vor der Integrationsumgebung verbirgt, wodurch das Konzept flexibel auf die Unterstützung nahezu beliebiger GUI–Systeme erweiterbar ist.

Betrachtet man den Gesamtkomplex der zur Einführung einer Abstraktionsschicht notwendigen Teilfunktionen, so stellt die Generierung einer allgemeinen Schnittstellenbeschreibung nur den ersten Schritt dar. Nicht minder wichtig ist

die weitere Übertragung dieser allgemeinen Beschreibung in ein konkretes Format, wie es für die Nutzung von einem entfernten Endgerät notwendig ist. Gerade diese Übertragung ist es, die den eigentlich sichtbaren Teil der Abstraktionsschicht darstellt, indem hier eine tatsächlich im Weiteren verwertbare Darstellung der Schnittstelle erstellt wird. In einem weiteren Teil dieses Kapitels wurde entsprechend beschrieben, wie die aus der Analyse hervorgegangene Momentaufnahme einen Transformationsprozess durchläuft, in welchem auf der Basis eines für die jeweilige Zielrepräsentation spezifischen Regelwerks eine Umsetzung in eine konkrete Repräsentation erfolgt. Es wurde gezeigt, dass sich mit Hilfe von XML zur Formulierung der allgemeinen Zwischenrepräsentation und XSL zur Realisierung der Regelwerke eine universelle Transformationskomponente erstellen lässt, die eine effiziente Umsetzung in eine Vielzahl möglicher Zielrepräsentationen möglich macht und dabei einen Wechsel zwischen den Darstellungsformen auch noch zur Laufzeit des gesteuerten Anwendungssystems machbar ist. Wie bei der Analyse erfolgt auch bei der Transformation keine Behandlung semantischer Aspekte der Schnittstelle. Entsprechend sind die verwendeten Regelwerke auch nicht applikationsspezifisch, sondern basieren ausschließlich auf den Fähigkeiten unterstützter Endgeräte.

Als letzte elementare Funktionseinheit innerhalb der Integrationsumgebung ist schließlich die Komponente zur direkten Interaktion mit dem Endgerät anzusehen. In ähnlicher Weise, wie die Analyse– und Steuerungskomponente eine Abstraktion in Richtung der zu unterstützenden Anwendungssysteme darstellt, bildet die Menge der Externen Kommunikationsadapter eine entsprechende Abstraktion in Richtung der zu unterstützenden Endgeräte. In einem weiteren Abschnitt dieses Kapitels wurde beschrieben, wie die Kommunikationsstrecke zwischen Endgerät und Anwendung als ein Nachrichtenkanal dargestellt werden kann. Stellt die Analyse– und Steuerungskomponente das eine Ende dieses Kanals dar, so lässt sich der Kommunikationsadapter als das andere Ende des Kanals ansehen. Er bildet die speziellen Anforderungen der zu unterstützenden Geräte hinsichtlich ihrer Interaktion mit entfernten Knoten auf der innerhalb der Integrationsumgebung verwendeten Nachrichtenvermittlung ab und sorgt dafür, dass in der einen Richtung die vom Transformator erzeugte Zielrepräsentation zum Endgerät übertragen werden kann und in entgegengesetzter Richtung die Befehle vom Endgerät aufgenommen und zur Steuerungskomponente weitergeleitet werden können. Wie sich in den Ausführungen dieses Kapitels zeigte, ist dabei eine gewisse Anpassung des Kanals möglich: Existiert beispielsweise für den vom Endgerät verwendeten Kommunikationspfad kein Sitzungskonzept (wie es für die Integration vorauszusetzen ist), so kann der verwendete Externe Kommunikationsadapter dieses nachbilden, ohne dass dieses für die Integrationsumgebung bzw. das Endgerät störend sichtbar würde. Es wurde weiterhin deutlich, dass gleichzeitig eine Reihe unterschiedlicher Kommunikationsadapter aktiv sein können, wobei jeder Kommunikationsart ein Adaptertyp zugeordnet ist und einzelne Instanzen eines Adapters zur Unterstützung verschiedener Geräte Verwendung finden. Schließlich wurde erläutert, dass das Konzept der Externen Kommunikationsadapter zu einer ähnlichen Flexibilisierung der Integrationsumgebung führt wie das Konzept, welches der Analysekomponente zugrunde liegt. So können zur Laufzeit weitere Adapter in das System integriert werden, um ursprünglich nicht bekannte Endgerätetypen zu unterstützen, ohne dass dieses einen Einfluss auf das Gesamtsystem besitzt.

Die optimale Unterstützung der Vielzahl möglicher Mobilgeräte ist nur dann möglich, wenn deren Eigenschaften akkurat erfasst und verwaltet werden können. Insbesondere für die Auswahl des am besten geeigneten Transformationsregelsatzes ist eine Verarbeitung solcher Eigenschaften, im Kontext dieser Arbeit als Profile bezeichnet, essentiell. Aus diesem Grund befasste sich ein weiterer Abschnitt dieses Kapitels eingehend mit der Generierung und Verarbeitung solcher Profile. Es wurden Ansätze zur Erfassung von Profilen beschrieben und Konzepte zur Speicherung und Verarbeitung von Profilen vorgestellt und es wurde anschließend beschrieben, wie Profile im Zusammenhang mit der in dieser Arbeit entworfenen Integrationsumgebung zum Einsatz kommen können. Schließlich wurden verschiedene in diesem Zusammenhang relevante Realisierungstechniken vorgestellt und bewertet.

Zum Abschluss des ersten Teils dieses Kapitels wurde ein zentraler Koordinationsmechanismus vorgestellt, der die zuvor beschriebenen Teilkomponenten miteinander verzahnt und in ihrer Handlung aufeinander abstimmt. Hierzu wurde zunächst das Sitzungsmodell beschrieben, das der entfernten Nutzung zugrunde liegt. Es wurde verdeutlicht, wie sich der entfernte Zugriff in drei unterschiedliche Sitzungstypen zerlegen lässt, die sich verschiedenen Bereichen des Kommunikationskanals zuordnen lassen. Es wurde verdeutlicht, dass erst auf der Grundlage des entworfenen Sitzungskonzepts eine Hauptzielsetzung dieser Arbeit erreicht werden kann, nämlich der Wechsel der im Zuge der Nutzung einer Anwendung verwendeten Endgeräte zur Laufzeit der Anwendung. Es wurde beschrieben, wie sich laufende Anwendungen einer Anwendungssitzung zuordnen lassen, die Verwendung verschiedener Anwendungen einer Benutzersitzung zuordnen lässt und schließlich der Zugriff von verschiedenen Endgeräten aus über verschiedene Gerätesitzungen repräsentiert wird. Gleichzeitig wurde verdeutlicht, dass durch das verwendete Sitzungskonzept eine Entkopplung zwischen Anwendungslauf und Anwendungsnutzung möglich ist, indem einzelne Sitzungen persistiert werden und die Assoziation zwischen einzelnen Geräte– bzw. Benutzersitzungen gelöst und neu verbunden werden kann. Schließlich wurde auf die Steuerungsfunktion eingegangen, welche der Sitzungssteuerung im Rahmen der entfernten Anwendungsnutzung zukommt. Es wurde beschrieben, wie sich die Anwendungsnutzung in fünf Phasen zerlegen lässt, welche im Verlauf der Nutzung immer wieder durchlaufen werden, und in welcher Weise jede der zuvor beschriebenen Teilkomponenten der Integrationsumgebung (Analyse, Steuerung, Transformation usw.) sich einer dieser Phasen zuordnen lässt. Es wurde hierbei deutlich gemacht, dass die Sitzungssteuerung anhand dieser Phasen eine Aktivierung der jeweiligen Komponente kontrolliert und auf diese Weise deren Verzahnung realisiert.

Im zweiten Teil des Kapitels wurde sodann auf die Integration mobiler Diensterbringer eingegangen. Hierbei wurde zunächst das Konzept der Middleware beschrieben, das seit längerem als Basis zur Realisierung von Diensten in verteilten Systemen Anwendung findet. Es wurden die wesentlichen Eigenschaften von Middleware erläutert und sodann beschrieben, wie sich die Unterstützung von Mobilität in Middleware integrieren lässt. Am Beispiel von Mobile CORBA als der Erweiterung einer herkömmlichen Middleware hinsichtlich einer Unterstützung mobiler Diensterbringer wurde veranschaulicht, wie eine solche nachträglich integrierte Unterstützung aussehen kann. Im Vergleich dazu wurde daraufhin mit OnTheMove ein Ansatz beschrieben, bei dem Mobilität von

Beginn an im Mittelpunkt stand. Schließlich wurde im Detail auf die JINI Umgebung eingegangen, bei welcher mehr als bei zuvor beschriebenen Ansätzen auf die Unterstützung dynamischer verteilter Anwendungssysteme Gewicht gelegt wurde. Aufbauend auf Konzepten wie Leasing, fest integrierten Dienstvermittlungsmechanismen sowie einer verteilten Ereignisbehandlung ermöglicht JINI den Aufbau von Ad–Hoc Netzwerken, in denen Diensterbringer ohne weitere Konfiguration in bestehende Anwendungsszenarien integriert werden können.

Als wichtiges Konzept von JINI kann die Verwendung migrierenden Codes zur Realisierung von Treibern angesehen werden, die zwischen Diensterbringern, Dienstvermittlern sowie Dienstnehmern ausgetauscht werden und welche die Kommunikation zwischen den beteiligten Komponenten regeln. Wie sich zeigte, besteht ein inhärentes Problem von JINI in diesem Zusammenhang darin, dass die verwendeten Protokolle Java als Programmiersprache nicht nur für Treiber, sondern auch für deren Registrierung festschreiben. Der Einsatz von JINI in mobilen Anwendungsszenarien ist somit an Ressourcenanforderungen gekoppelt, die nur durch die wenigsten Geräte erfüllbar sind. In diesem Sinne bestand die Hauptaufgabe des zweiten Teils dieses Kapitels in der Vorstellung des Konzeptes der opaken Treiber, welche dieses Hemmnis zu überwinden helfen. Es wurde beschrieben, wie sich durch die Modifikation des Registrierungsprotokolls und die gezielte Nutzung des von JINI verwendeten Prinzips der Code–Migration auch nicht Java–fähige Geräte in Anwendungsszenarien auf der Basis von JINI einbinden lassen. Es wurde gezeigt, wie opake Treiber aufgebaut sind und wie sie sich in das JINI–Gesamtkonzept einbinden lassen.

In Analogie zu dem im ersten Teil des Kapitels beschriebenen Integrationskonzept zur Unterstützung mobiler Dienstnehmer wurde die im zweiten Teil vorgeschlagene Erweiterung von JINI um die Verwendung opaker Treiber als Integration durch Migration bezeichnet. Zur Abrundung des Kapitels wurde das Konzept der Surrogate beschrieben, einem weiteren Ansatz zur Integration nicht java–fähiger Diensterbringer. Es wurde in seinen wesentlichen Eigenschaften skizziert und erläutert, inwiefern es sich von der Integration durch Migration unterscheidet.

Die Erarbeitung der verschiedenen Konzepte und deren Aggregation zu einer flexiblen Integrationsumgebung für mobile Dienstnehmer bzw. –erbringer stellt den Schwerpunkt dieser Arbeit dar. Der hiermit abgeschlossene erste Teil diente deren Herleitung, während im folgenden zweiten Teil darauf eingegangen wird, wie die Ergebnisse der konzeptionellen Arbeit in einem Implementationsschritt in eine konkrete Realisierung umgesetzt werden können. Im folgenden Kapitel wird dabei zunächst eine Gesamtsicht entworfen, welche die auf der Basis der konzeptionellen Vorarbeit entstandene Integrationsumgebung in ihrer Vollständigkeit betrachtet. Im weiteren Verlauf wird dann auf wesentliche Implementationsdetails eingegangen, die zur Verdeutlichung der hier beschriebenen Konzepte herangezogen werden sollen.

Teil II

Eine Integrationsumgebung auf Anwendungsebene

Kapitel 5

Eine Integrationsumgebung für mobile Geräte

Die Tragfähigkeit der in den vorigen Kapiteln eingeführten und im Detail beschriebenen Konzepte wurde im Rahmen einer prototypischen Umsetzung nachgewiesen. In diesem Kapitel wird ein Überblick über die entstandene Hydeparc Umgebung[1] gegeben. Es wird dabei mit einer Beschreibung von Entwurfskriterien begonnen, deren Umsetzung am Anfang der Entwicklung standen. Danach werden die grundsätzlichen Eigenschaften der Umgebung beschrieben. Die Umgebung wird in ihren wesentlichen Komponenten vorgestellt und diese werden sodann in ihren Aufgaben kurz eingeführt. Schließlich wird diskutiert, welche Bezeichnung der entstandenen Integrationsumgebung am besten gerecht wird.

5.1 Entwurfskriterien

Die Entwurfskriterien der prototypischen Umsetzung lassen sich in direkter Weise aus den Ausführungen der vorigen Kapitel ableiten. Im Wesentlichen sind diese auf drei grundlegende Anforderungen zu verdichten, die sich aus dem dieser Arbeit zentralen Abstraktionsgedanken ergeben, nämlich:

- Entkopplung beteiligter Entitäten,
- Generizität der Umgebung,
- Unterstützung geschlossener Systeme.

Betrachtet man die drei Punkte genauer, so lassen diese sich in folgender Weise zu Forderungen an den Entwurf konkretisieren:

[1] *Hyper Distributed Environment for Personal Appliances using Remote Control*

Forderungen zur Entkopplung beteiligter Entitäten

- **Entkopplung zwischen Endgerät und Middleware**: Es soll keine direkte Beziehung zwischen dem Endgerät und der verwendeten Middleware bestehen. Die Middleware soll auch ohne Anpassung auf dem Endgerät ausgewechselt werden können.

- **Entkopplung zwischen Endgerät und Anwendung bzw. Anwendungskontext**: So, wie keine Abhängigkeiten zwischen der verwendeten Middleware und dem verwendeten Endgerät bestehen sollen, muss auch die Anwendung in ihrem zugehörigen Anwendungskontext vom Endgerät unabhängig sein. Änderungen der Anwendungen sollten keiner Nachbesserung auf dem Endgerät bedürfen und andererseits müssen neue Endgeräte eingeführt werden können, ohne dass die Anwendung hierfür zu modifizieren wäre.

- **Entkopplung von Zielrepräsentation und Endgerät**: Es hat sich gezeigt, dass verschiedene Endgeräte oftmals eine gemeinsame Zielrepräsentation aufweisen (z.B. WML), sich darüber hinaus jedoch hinsichtlich des Umfangs ihrer Unterstützung voneinander unterscheiden (z.B. bezüglich der maximalen Größe eines WML Dokumentes). Solche Abweichungen sollten innerhalb der Integrationsumgebung behandelt werden können.

Forderungen zur Generizität der Umgebung

- **Keine Einschränkung auf bestimmte Anwendungskontexte**: Der Ansatz sollte insofern generisch sein, als er nicht auf den Einsatz in einer bestimmten Anwendungsumgebung abgestimmt sein sollte. Auch sollte er nicht den Kontext einer bestimmten Middleware voraussetzen.

- **Keine Einschränkung bezüglich der Nutzbarkeit der Anwendung**: Anwendungen sollten nicht ausschließlich als Teil der Integrationsumgebung nutzbar sein. Es sollten keine besonderen Programmierschnittstellen zu deren Entwurf zu verwenden sein und die Einbeziehung in die Integrationsumgebung sollte letztlich auch zur Laufzeit der Anwendung möglich sein.

- **Keine besonderen Anforderungen an das Endgerät**: Die Integrationsumgebung sollte bezüglich Betriebssystem, Kommunikationsplattform, verwendbarer Programmiersprache usw. nur minimale Anforderungen an das mobile Endgerät stellen. Im Zusammenhang hiermit ist ein wichtiges Kriterium:

- **Keine Einschränkung auf bestimmte Endgeräte bzw. Zielrepräsentationen**: Der Ansatz sollte genügende Allgemeingültigkeit aufweisen, um die inhärente Einschränkung auf bestimmte Endgeräte vermeiden zu können. Insbesondere sollte keine Festlegung auf einen bestimmten Gerätetyp existieren. Das schnurlose Telefon sollte als Beispiel eines sehr eingeschränkten mobilen Endgerätes genauso seine Berücksichtigung finden wie der leistungsfähige Laptop.

- **Keine Einschränkungen bezüglich Programmiersprache, Betriebssystem usw.**: Die Integrationsumgebung sollte keine Anforderungen hinsichtlich Programmiersprache, Betriebssystem usw. festschreiben, die im Rahmen der Entwicklung von Anwendungen benutzt werden können, die durch die Integrationsumgebung hindurch von Mobilgeräten verwendet werden können.

Forderungen zur Unterstützung geschlossener Systeme

- **Unterstützung „geschlossener" Endgeräte**: die Umgebung sollte keine Modifikationen am Endgerät selbst notwendig machen. Es sollte möglich sein, den Zugriff auf entfernte Anwendungssysteme möglich zu machen, ohne dass hierfür ein Aufspielen besonderer Anwendungsprogramme auf dem Endgerät notwendig wäre.

- **Unterstützung „geschlossener" Anwendungsprogramme**: Der Zugriff auf nicht modifizierbare Anwendungsprogramme sollte ebenso unterstützt werden wie auf geschlossene Endgeräte.

Es ist offensichtlich, dass die angeführten Entwurfsforderungen jeweils in ihrer maximalen Ausprägung genannt worden sind, wie sie aus den konzeptionellen Vorarbeiten abgeleitet werden kann und die sich oftmals allenfalls annähern lässt. Insbesondere ist es nahezu unmöglich alle Anforderungen gleichzeitig optimal zu unterstützen. Allen Punkten gemeinsam ist die zugrunde liegende Forderung nach Generizität, was sich in der zweiten genannten Gruppe explizit darstellt, was aber auch für die Entkopplung und die Unterstützung geschlossener Systeme fundamental ist. Abhängigkeiten sollen zwischen beteiligten Einheiten vermieden werden, um so den universellen Einsatz sicherzustellen. Beliebige Endgeräte sollen genauso möglich sein, wie beliebige Zielrepräsentationen. Bezüglich der verwendbaren Anwendungsprogramme sollen genauso wenig Beschränkungen festgeschrieben werden, wie bezüglich der zu verwendenden Programmiersprache, der nutzbaren Middleware, der zugrunde liegenden Hardwareplattform.

In Anbetracht der Menge möglicher Kombinationen, wie sie sich allein aufgrund der existierenden Vielfalt an Endgeräten ergeben, ist eine solche Forderung nach universeller, auch zukünftiger Einsetzbarkeit die wünschenswerte Grundlage aller Entwurfskriterien einer Integrationgsumgebung. Andererseits sind diese Kriterien in ihrer Gesamtheit kaum zu erreichen und teilweise sogar orthogonal zueinander. So lässt sich die optimale Unterstützung eines bestimmten Anwendungskontextes oder eines bestimmten Endgerätes nur durch die ausdrückliche Berücksichtigung der jeweiligen Eigenschaften des Kontextes oder des Endgerätes erreichen. Ebenso sind sicherlich der möglichen Unterstützung von geschlossenen Geräten bzw. Anwendungen genauso Grenzen gesetzt wie der Unterstützung beliebiger Zielrepräsentationen. Bestimmte Bedingungen lassen sich immer identifizieren, die von einem Endgerät erfüllt sein müssen, damit seine Einbeziehung überhaupt möglich ist.

5.2 Die funktionalen Schwerpunkte der Integrationsumgebung

Nachdem im vorigen Abschnitt die Anforderungen dargestellt wurden, die der prototypischen Umsetzung des Integrationskonzepts zugrunde liegen, soll nun näher auf die funktionalen Schwerpunkte eingegangen werden, die sich hieraus für die Realisierung ergeben.

Entsprechend der Forderung nach Unterstützung sowohl von Diensterbringern als auch von Dienstnehmern ist die Integrationsumgebung in zwei funktionale Schwerpunkte aufgegliedert, nämlich die Java Border Service Architecture (JBSA) auf der einen Seite (vgl. [MWL00b] bzw. [MWWL00]) sowie die Jini Small Device Adapter (JiSDA) auf der anderen (siehe [MWHL00]). Dabei ergibt sich folgende Zuordnung:

- **Integration von Dienstnehmern**: In der Java Border Service Architecture sind diejenigen Module der Infrastruktur zusammengefasst, welche für die Einbindung mobiler Dienstnehmer in eine Anwendungsumgebung notwendig sind. Die JBSA realisiert somit die Adaptionsschicht, welche die Eigenheiten des Mobilgerätes vor der Anwendungsumgebung versteckt. Die JBSA stellt die Mechanismen zur Generierung abstrakter Beschreibungen von Benutzungsschnittstellen laufender Anwendungen, zur Umformung dieser Beschreibungen in konkrete Repräsentationen sowie zu deren Übertragung auf das mobile Endgerät zur Verfügung. Die JBSA beinhaltet darüber hinaus die notwendigen Mechanismen zur Verwaltung von Sitzungskontexten, in denen die Anwendungen organisiert sind und laufen. Schließlich realisiert sie Mechanismen zur Zugriffssteuerung und Benutzerverwaltung.
- **Integration von Diensterbringern**: Die Jini Small Device Adapters umfassen diejenigen Module, welche innerhalb der Hydeparc Umgebung die Integration mobiler Diensterbringer realisieren. Wie gesagt, wurde die Integration von mobilen Diensterbringern in Hydeparc am Beispiel der JINI Middleware behandelt. Entsprechend sind in der JiSDA diejenigen Module enthalten, die zur Verwendung ressourcenschwacher Diensterbringer in JINI notwendig sind.

Abbildung 5.1 illustriert, wie sich die einzelnen Module den beiden Hauptgruppen zuordnen lassen.

Anhand von Beispielen sollen die Aufgaben der beiden Teilbereiche verdeutlicht werden. Die beteiligten Module und deren Handlungen werden dabei zunächst nur oberflächlich gestreift und nur so weit beschrieben, wie es für deren grobe Einordnung notwendig ist. Eine detaillierte Beschreibung erfolgt für die Behandlung von Dienstnehmern im folgenden Kapitel 6 und für die Behandlung von Diensterbringern im Kapitel 7.

Zunächst soll die Einbindung von Diensterbringern näher erläutert werden. Es sei als Anwendungsszenario die Überwachung von Temperaturen in einem Gebäudekomplex angenommen. An verschiedenen Stellen sollen Sensoren die Temperatur messen und diese an einen zentralen Verwaltungsknoten weitergeben, wo die

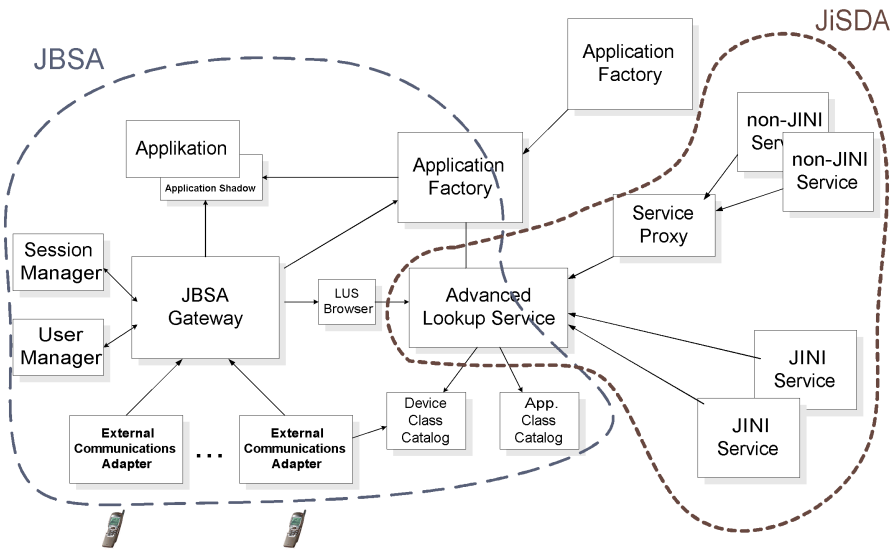


Abbildung 5.1: Die Aufteilung der Gesamtfunktionalität von Hydeparc in JBSA und JiSDA

eingehenden Messwerte registriert und für die weitere Auswertung gespeichert werden. Sensoren sollen nach Bedarf und ohne weitere Anpassung des Verwaltungsknotens in das Messsystem integriert bzw. aus dem Messsystem entfernt werden können. Mit seiner besonderen Eignung für den Aufbau selbstkonfigurierender Netzwerke bietet sich JINI als Entwicklungsplattform für diesen Anwendungsfall an. Werden die Sensoren und der zentrale Verwaltungsknoten als JINI Dienste realisiert, so kann die Registrierung und Abmeldung von Sensoren im Messsystem zunächst vollständig der verwendeten Middleware übertragen werden. Auch die Bereitstellung einer für den Verwaltungsknoten einheitlichen Zugriffsschnittstelle auf die Sensoren und letztlich auch die ansprechende Darstellung der Sensoren in einem Administrationsprogramm können mit JINI realisiert werden. Der Sensor selbst bietet hierbei eine Aufrufschnittstelle an, über welche die Temperatur erfragt werden kann. Nachdem der Sensor (per Kabel oder sogar per Funk) an das Computernetzwerk angeschlossen wurde, kann über die Schnittstelle die Temperatur abgerufen werden. Der für das JINI System realisierte Stellvertreter „kennt“ die Schnittstelle und kann hierüber die Temperatur erfragen, die er dann auf ansprechende Weise darstellt. Der zentrale Verwaltungsknoten kann die im JINI System registrierten Stellvertreter in einem Programmfenster grafisch darstellen bzw. in einer Datenbank ablegen.

Das Problem der Verwendung von JINI liegt in diesem Beispiel in seinen Anforderungen an die zu verwendenden Sensoren. Da JINI die Nutzung von Java als Realisierungssprache vollwertiger Dienste voraussetzt, kommt für die Entwicklung der Sensoren nur PC–Elektronik in Frage, die neben der Sensorik auch genügenden Speicher für den Dienst selbst, eine Java VM und die Implementation des Remote Method Invocation Protokolls besitzt. Miniaturisierte Hardware mit dem geforderten Profil ist derzeit nicht so kostengünstig verfügbar, dass ihr Einsatz in einem einfachen Messsystem unter ökonomischen Gesichtspunkten vertretbar wäre. Die Verwendung kostengünstiger herkömmlicher PC–Elektronik

scheidet andererseits aufgrund ihrer Größe und aufgrund weiterer Faktoren wie Stromverbrauch, Geräuschentwicklung usw. aus.

Bei der Verwendung der JiSDA kann auf die Verwendung von Java auf dem Sensor selbst verzichtet werden. Wie für JINI typisch, wird zwar der Stellvertreter in Java programmiert und auf dem Gerät abgelegt. Die Kontaktaufnahme mit dem Verzeichnisdienst der Infrastruktur, die Anmeldung des Stellvertreters und die weitere Interaktion mit den Diensten der JINI Umgebung werden hingegen über neuartige Protokolle durchgeführt, bei denen auf den Einsatz von Java verzichtet werden kann. Insbesondere ist weder die Serialisierung von Daten im Zuge der Registrierung eines Dienstes notwendig, noch muss bei dieser Registrierung mittels RMI kommuniziert werden. Der Stellvertreter stellt aus der Sicht des Sensors einen unstrukturierten Datenblock dar, einen Treiber, der im internen Speicher abgelegt ist und der im Zuge des Anmeldungsprotokolls uninterpretiert an den Verzeichnisdienst der JINI Umgebung übertragen wird. Der Sensor muss im Zusammenhang mit JiSDA also nur über die Fähigkeit verfügen, über das Netzwerkprotokoll TCP/IP mit einem entfernten Dienst Kontakt aufnehmen und mit entfernten Netzwerkknoten kommunizieren zu können. Des Weiteren muss genügende Speicherkapazität vorhanden sein, um den Treiber lokal speichern zu können. Anforderungen an die verwendete CPU, das zugrunde liegende Betriebssystem, die eingesetzte Programmiersprache usw. werden hingegen nicht gestellt. Hier kann die für den jeweiligen Einsatzzweck optimale Auswahl getroffen werden.

Im Beispiel des Messsystems ist die Realisierung in C auf der Basis eines Single–Chip PCs mit wenigen Kilobytes RAM sinnvoll. Auf dem Sensor ist ein JINI Stellvertreter gespeichert, der die entfernte Schnittstelle zur Abfrage der Temperaturmesswerte enthält. Außerdem enthält der Sensor die Anwendungslogik, um mit dem entfernten JINI Verzeichnisdienst in Kontakt zu treten, um den Stellvertreter zu übertragen. Schließlich ist auf dem Sensor die notwendige Anwendungslogik enthalten, um das Leasing Protokoll von JINI unterstützen zu können, mit dem in gewissen Abständen die Registrierung beim Verzeichnisdienst erneuert wird.

Wird ein Sensor an das Netzwerk angeschlossen, so beginnt dieser, über das Netzwerk nach verfügbaren Instanzen des JINI Verzeichnisdienstes zu suchen, die das angepasste Registrierungsprotokoll verstehen. Ist eine solche Instanz verfügbar, so meldet sich diese bei dem Sensor und fordert ihn zur Übergabe des Stellvertreters auf. Der Sensor überträgt den lokal gespeicherten Code an den Verzeichnisdienst, welcher diesen in seiner Datenbank ablegt. Entsprechend der in JINI üblichen Vorgehensweise ist die Speicherung des Stellvertreters an ein Zeitintervall gebunden, an dessen Ende die Registrierung erneuert werden muss. Nach Anmeldung des Stellvertreters wartet der Sensor nun auf Aufrufe, entweder zwecks Abfrage des Temperaturwertes über seine Anwendungsschnittstelle oder zwecks Aufforderung zur erneuten Anmeldung des Stellvertreters durch den Verzeichnisdienst über seine Dienstschnittstelle. Bis zur Entfernung des Sensors vom Netzwerk verbleibt er in diesem Zustand. Wird der Sensor vom Netzwerk entfernt, so kann er nicht mehr auf die Aufforderungen durch den Verzeichnisdienst reagieren, woraufhin jener den registrierten Stellvertreter aus seiner Datenbank entfernt.

Die Nutzung des Dienstes durch Dienstnehmer in der JINI Umgebung verläuft

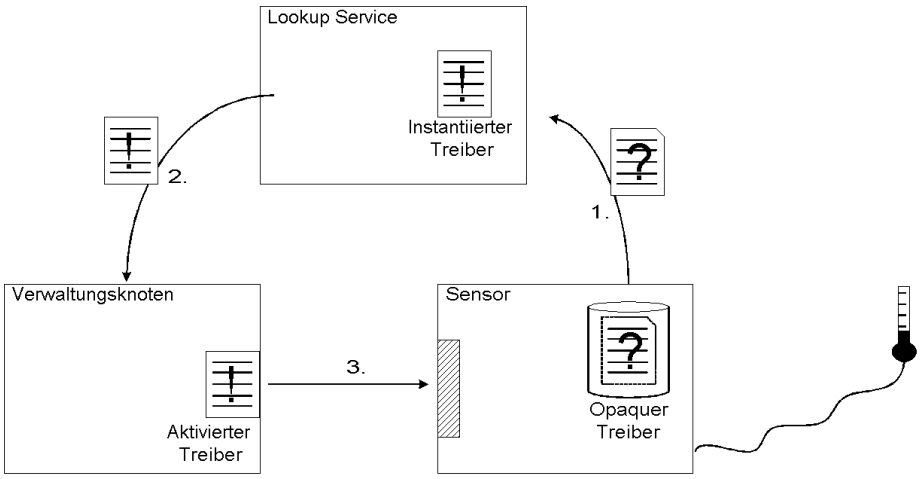


Abbildung 5.2: Zusammenspiel der Komponenten im beispielhaften Temperaturmesssystem

in Hydeparc bzw. in JiSDA analog zu herkömmlichem JINI. Wird dem zentralen Verwaltungsknoten als JINI Dienstnehmer die Anmeldung eines neuen Sensors gemeldet, so überträgt dieser den registrierten Stellvertreter in seinen lokalen Speicher und integriert die enthaltenen Objekte in seine Benutzungsschnittstelle. Soll der Temperaturwert eines Sensors abgefragt werden, so wird über die vom Stellvertreter angebotene Aufrufschnittstelle die entsprechende Anfrage gestellt. Der Stellvertreter leitet die Anfrage an den Sensor weiter, der den aktuellen Messwert zurückgibt. Der erfragte Wert wird dann auf geeignete Weise im zentralen Verwaltungsknoten angezeigt. Für die Interaktion zwischen Stellvertreter und Sensor kann dabei das in den Stellvertreter einkodierte, für den Sensor optimale Kommunikationsprotokoll verwendet werden.

Anhand des beschriebenen Beispiels lassen sich die Komponenten der Hydeparc Umgebung identifizieren, die im Kontext der Dienstintegration durch JiSDA relevant sind. Dieses ist zunächst einmal der Dienst selbst. Er enthält die notwendige Applikationslogik, um im Sinne eines Anwendungsszenarios einen Mehrwert zu erbringen, der über das Netzwerk in Anspruch genommen werden soll. Der Dienst enthält darüber hinaus den Stellvertreter, der in der JINI Umgebung registriert wird und über den der Zugriff erfolgt, sobald er in einen Dienstnehmer integriert wurde. Schließlich enthält der Dienst die Anwendungslogik, um gegenüber dem modifizierten JINI Verzeichnisdienst als JINI Dienst auftreten zu können. Den Kernteil von JiSDA stellt der modifizierte Verzeichnisdienst dar. Dieser bietet neben der vollen Funktionalität eines herkömmlichen Verzeichnisdienstes in JINI die eben skizzierten Protokolle zur Behandlung ressourcenschwacher Dienste. Im Einklang mit der JINI Terminologie, die den Verzeichnisdienst als Lookup Service (LUS) bezeichnet, wird der JiSDA Verzeichnisdienst *Advanced Lookup Service* oder ALUS genannt. Weitere Teilkomponenten von JiSDA stellen neben den Diensten natürlich die Dienstnehmer dar, die in Hydeparc durch die anderen Dienste der Integrationsinfrastruktur vertreten sind. Dieser Gruppe können die so genannte Anwendungsfabrik zum Start von Anwendungen und der Browser zum Suchen von Anwendungen zugeschlagen werden, welche beide im nächsten

Kapitel eingehend beschrieben werden.

Der Integration von Diensterbringern durch JiSDA steht in Hydeparc mit der JBSA ein Unterstützungssystem zur Integration von Dienstnehmern gegenüber. Wie JiSDA soll auch die JBSA anhand von Beispielen kurz vorgestellt werden. Die JBSA bietet zwei Ausgangspunkte für die Integration von Dienstnehmern. Zum einen kann dieses auf der Basis gewöhnlicher Java Anwendungen geschehen, die eine grafische Benutzungsschnittstelle besitzen, welche überwacht und nach geeigneter Transformation an ein mobiles Endgerät weitergeleitet wird. Eine solche Anwendung ist nicht speziell für die Verwendung im Kontext der Hydeparc Umgebung entworfen, sondern nachträglich über die Anwendungsfabrik eingebunden und so für die Verwendung vom mobilen Gerät verfügbar gemacht worden. Dieser Art von Anwendung steht diejenige gegenüber, die explizit für den Einsatz in der Hydeparc Umgebung entworfen wurde und die zum Aufbau ihrer Benutzungsschnittstelle direkt auf die Programmierschnittstelle der JBSA zugreift. Dieses können einerseits herkömmliche Anwendungen sein, die nur im Kontext der Hydeparc Umgebung benutzt werden und die daher keine grafische Benutzungsschnittstelle im üblichen Sinne benötigen. Konkrete Beispiele hierfür sind der Browser für beim ALUS registrierte Dienstangebote oder die Sitzungssteuerung. Andererseits fallen in diese Kategorie auch Dienste aus JiSDA, deren Stellvertreter keine grafische Benutzungsschnittstelle im üblichen Sinne enthalten, sondern die ausschließlich für die Verwendung durch die Hydeparc Umgebung hindurch entworfen wurden. Denkbar sind selbstverständlich auch Mischformen, die neben einer normalen grafischen Benutzungsschnittstelle außerdem noch eine zweite, auf die JBSA abgestimmte Schnittstelle implementieren. Solches wäre etwa im Beispiel des Temperaturmesssystems denkbar, wo die Stellvertreter sowohl durch den zentralen Verwaltungsknoten nutzbar sein sollen, die Abfrage aber ebenfalls vom Mobilgerät aus zu ermöglichen ist.

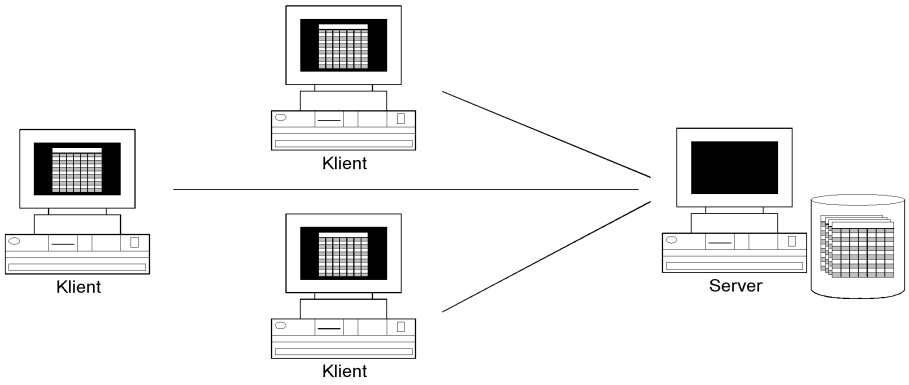


Abbildung 5.3: Schematischer Aufbau des verteilten Terminplaners

Als Beispiel für die Dienstnehmerintegration sei die Verwaltung von Terminen in einem einfachen verteilten Kalender angenommen. In einem zentralen Anwendungsdienst werden hierbei die Terminkalender der Angestellten eines Unternehmens verwaltet. Mit einer entsprechenden Zugriffsapplikation können sich die Benutzer über das Netzwerk mit dem Anwendungsdienst verbinden und auf die Termindaten zugreifen. Sie können die zurzeit gespeicherten Termine einsehen, dem Terminbuch neue Einträge hinzufügen, bestehende Einträge ändern

oder vorhandene Einträge aus der Datenbasis entfernen (siehe Skizze 5.3). Bei der Applikation handelt es sich um eine eigenständige Anwendung, die dem Anwender über ihre grafische Benutzungsschnittstelle den Zugriff auf den entfernten Anwendungsdienst ermöglicht (siehe Abb. 5.4). Neben dieser direkten Nutzung über die Applikation sollen die Angestellten das zentrale Terminbuch auch dann nutzen können, wenn sie sich nicht an ihrem Arbeitsplatz befinden. Hierzu sollen sie ihr WAP Telefon, ihren PDA oder auch das WWW Info–Terminal in der Hotellobby benutzen können. Um nicht für jeden dieser Zugriffskanäle eine weitere Applikation entwickeln zu müssen, kann die bestehende Anwendung in die JBSA eingebunden und sodann wahlweise über einen der genannten Kanäle benutzt werden. Am Beispiel des Zugriffs über ein WAP Telefon sollen die Funktionsweise und das Zusammenspiel der beteiligten Komponenten der JBSA verdeutlicht werden. Dabei werden die Zusammenhänge zunächst vergröbert dargestellt und Abläufe zum Teil verkürzt, um zunächst das allgemeine Verständnis zu ermöglichen. Die Beschreibung der Funktionalität, die den Komponenten jeweils zugrunde liegt, und ihrer Interaktion untereinander erfolgt dann im Detail in dem folgenden Kapitel.

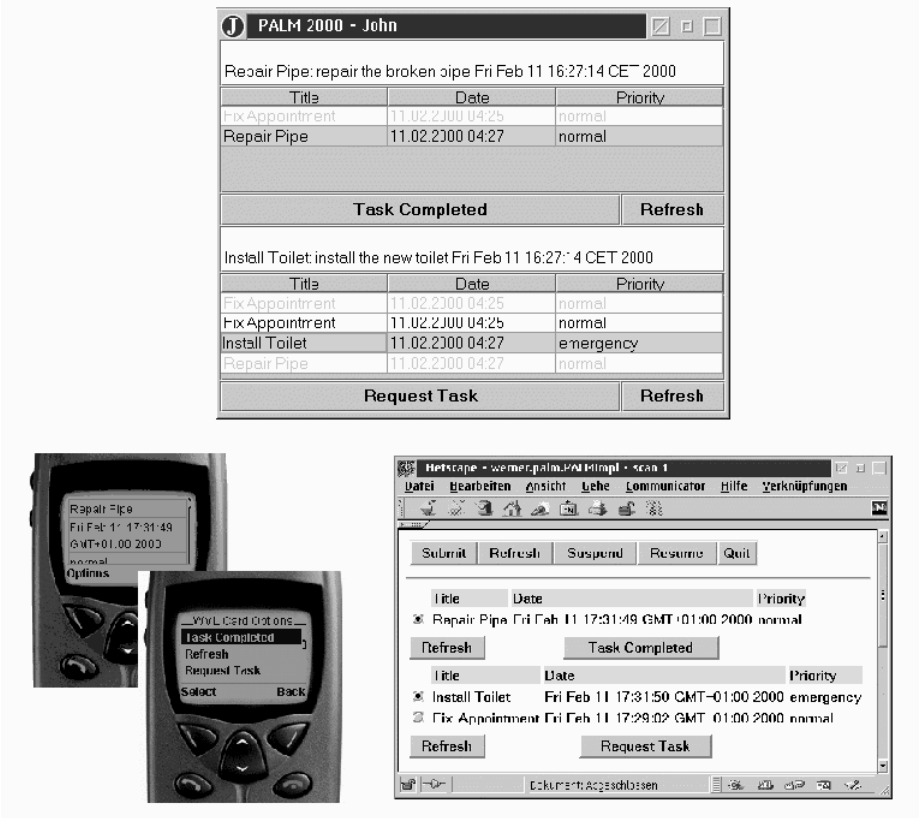


Abbildung 5.4: Darstellung einer Anwendung auf mehreren Zielplattformen

Durch Auswahl des entsprechenden Lesezeichens wird zunächst eine Kommunikationsverbindung zwischen dem WAP Telefon und dem External Communication Adapter (ECA) als Endpunkt der JBSA hergestellt. Hierbei wird in

der für WAP typischen Weise verfahren und eine Wahlverbindung mit dem Betreibernetzwerk hergestellt und danach das im Betreibernetzwerk befindliche WAP Gateway beauftragt, die unter der im Lesezeichen gespeicherten Seite in das Telefon zu laden. Der ECA wird immer in der für das Endgerät und den verwendeten Kommunikationskanal üblichen Notation adressiert, für WAP also mittels eines Uniform Resource Locators (URL). Im alternativen Fall eines ECAs zur Realisierung eines Sprachkanals wäre die Adressierung mittels einer Telefonnummer denkbar, im Falle eines ECAs für Mailkommunikation eine Mailadresse et cetera. Zumeist werden unterschiedliche Zugangskanäle einen eigenen ECA besitzen, der die Adaption in der jeweils notwendigen Weise durchführen kann und die für das Gerät spezifische Kommunikationsform in eine hinsichtlich des Gerätes generischen Kommunikationsform innerhalb der Integrationsumgebung umsetzt. So wird der ECA für gesprochene Sprache sich grundsätzlich von dem für Mailkommunikation unterscheiden. Gleicht sich andererseits die Realisierung mehrerer Zugriffskanäle, so kann auch ein ECA unter derselben Adresse für mehrere Kanäle zuständig sein. Wie im Falle von WAP und WWW ist es dann Sache des ECAs, die eigentlich benötigte Adaptionsform herauszufinden und an die Integrationsumgebung weiterzumelden.

Nimmt der ECA nun vom WAP Gateway die eingehende Anfrage des WAP Telefons entgegen, so erkennt er zunächst, ob sich die Anfrage im Rahmen einer bestehenden Gerätesitzung befindet. Eine Gerätesitzung umschließt als logische Klammer die einander folgenden Aufrufe von einem Gerät zwischen Aufbau und Abbruch der Kommunikationsverbindung zwischen ECA und Gerät. Anhand der Gerätesitzung wird im weiteren Verlauf die Zuordnung zwischen Gerät und zu steuernder Anwendung hergestellt. Beim Aufbau der Verbindung existiert zunächst keine Sitzung und so wird diese auch im vorliegenden Fall zunächst erstellt und dem Gerät zugeordnet. Danach extrahiert der ECA aus der Anfrage die Geräteparameter und leitet diese zur Analyse an den Device Class Catalog (DCC) weiter. Der DCC dient dazu, die Auswahl des für das Gerät optimalen Transformationsregelsatzes zu treffen, der im weiteren Verlauf zur Übersetzung abstrakter Schnittstellenbeschreibungen in konkrete Repräsentationen für das Endgerät verwendet wird. Anhand der übergebenen Parameter wird dazu der Gerätetyp eingegrenzt und eine bestimmte Klasse identifiziert. Geräteklassen sind in einer Hierarchie angeordnet, so dass im Falle unbekannter Parameter eine allgemeinere Klasse angenommen wird.

Die vom DCC gemeldete Klasse wird vom ECA in einer Aufforderung zur Einrichtung einer Kommunikationssitzung an das JBSA Gateway weitergereicht. Das Gateway stellt die zentrale Komponente der JBSA dar. Es koordiniert die Geräte– und Benutzersitzungen, lenkt die Anwendungsfabrik beim Start neuer Anwendungen und kommuniziert mit sodann laufenden Anwendungen, startet und steuert den Transformationsprozess und realisiert die Zugangskontrolle zur Integrationsumgebung. In der Bezeichnung als „Gateway“ (dt. Tor, Zugang) wird diese zentrale Bedeutung unterstrichen. Dabei wird es vornehmlich in einer koordinierenden Funktion eingesetzt, wobei die eigentlichen Handlungen von angeschlossenen Komponenten wie der Fabrik, der Sitzungssteuerung oder den Transformatoren übernommen werden. Auf der Basis der vom ECA kommenden Klasseninformation instantiiert das Gateway nun zunächst den für WAP notwendigen (und ggf. für den WAP Dialekt des anfragenden Gerätes optimierten) Transformator. Vom User Manager lässt es einen Anmeldedialog generieren, der

vom Transformator in die für WAP passende Zielrepräsentation übersetzt wird. Die resultierende WML Seite wird an den ECA zurückgesandt und von dort an das WAP Telefon weitergeleitet.

Die Anmeldedaten werden vom ECA wieder an das Gateway geleitet, welches anhand dieser Daten durch den User Manager prüfen lässt, ob der Benutzer des Gerätes erkannt werden konnte und es sich somit um einen gültigen Anmeldeversuch handelt. Der User Manager enthält zu jedem der Umgebung bekannten Benutzer dessen Anmeldekennung und Passwort, kann darüber hinaus aber auch zur Speicherung von benutzerspezifischen Profilen verwendet werden, in denen dann z.B. zuletzt verwendete Anwendungen oder besondere Präferenzen zu verwendeten Geräten abgelegt werden und im Zuge der Einrichtung und Nutzung einer Benutzersitzung zum Einsatz kommen können. Konnte der Benutzer identifiziert werden, so wird nun durch den Session Manager überprüft, ob zu dem Benutzer eine aktive Benutzersitzung existiert. In der Benutzersitzung werden die von einem Benutzer gestarteten und zurzeit laufenden Anwendungen und die jeweils aktive Anwendung vermerkt. Existierte für den Benutzer keine Sitzung, so wird eine solche erstellt und dann die Anwendungsfabrik mit dem Start der ersten Anwendung für diese Sitzung beauftragt.

Die Anwendungsfabrik dient zum Starten von Anwendungen, die in der Integrationsinfrastruktur benutzt werden sollen. Bei diesen Anwendungen kann es sich zum einen um solche handeln, die explizit für die Nutzung in der Hydeparc Umgebung entworfen wurden. Für sie ist es die Aufgabe der Fabrik, diese zu starten und einen Kommunikationskanal zwischen Anwendung und Gateway herzustellen. Im Falle von Anwendungen, die um ihre entfernte Nutzung nicht wissen, obliegt es der Fabrik zusätzlich, den für die Einbindung notwendigen Application Shadow an die Anwendung zu binden. Der Kommunikationskanal wird in diesem Falle zwischen Gateway und Shadow aufgebaut. Im angeführten Beispiel wird von der Fabrik die Zugangssoftware mitsamt Application Shadow gestartet und die Zugangsadresse des Shadows an das Gateway zurückgemeldet.

Das Gateway lässt die Information über die erstellte Anwendung durch den Sessionmanager in der Benutzersitzung vermerken und erfragt dann beim Shadow die erste Momentaufnahme der Benutzungsschnittstelle der Anwendung. Der Application Shadow liegt als eine Art Adapter zwischen Anwendung und Integrationsumgebung. Über die vom Betriebssystem zur Verfügung gestellten Mechanismen schaltet er sich in die Objekthierarchie der Benutzungsschnittstelle der Anwendung ein und analysiert deren Aufbau. Die Ergebnisse werden in ein generisches Format auf der Basis von XML umgesetzt und an das Gateway zurückgegeben. Das Gateway nutzt den für die Sitzung zuständigen Transformator zur Umformung der Beschreibung in die konkrete Repräsentation und sendet diese an den ECA weiter. Vom ECA gelangen die Daten wiederum zum WAP Gateway und von dort zum WAP Telefon, wo sie im Display dargestellt werden.

Nachdem nun sowohl die Gerätesitzung als auch die Benutzersitzung initialisiert sind und der Benutzersitzung eine laufende Anwendung zugeordnet ist, verlaufen die weiteren Zugriffe auf eine Anwendung, wie etwa den Terminplaner dieses Beispiels, ungeachtet des verwendeten ECAs in jeweils fünf Phasen, die unter 4.3.8 bereits eingeführt wurden und die hier nochmals am Beispiel kurz umrissen werden sollen, bevor dann im folgenden Kapitel auf technische Details

eingegangen wird.

- **Phase 1 — Eingang der Anfrage**: Der Benutzer des WAP Telefons trifft anhand der Darstellung im Display eine Auswahl und wählt beispielsweise den Knopf zum Hinzufügen eines Termins aus. Diese Eingabe wird über den Umweg des WAP Gateways an den ECA übermittelt und dort in das innerhalb der Integrationsumgebung verständliche Format umgeformt.
- **Phase 2 — Authentifizierung und Autorisierung**: Die eingegangenen Daten werden einer Gerätesitzung zugeordnet. Da die Gerätesitzung ihrerseits nach der erfolgreichen Anmeldung eines Benutzers mit einer Benutzersitzung verbunden ist, kann erkannt werden, ob die Anfrage im Rahmen einer gültigen Benutzersitzung gestellt wurde. Ist dieses der Fall, wird weiterhin geprüft, ob die in der Anfrage kodierte Handlung direkt die Anwendung betrifft (wie in diesem Fall die Auswahl des Knopfes) oder die Umgebung selbst (wie etwa beim Start einer weiteren Anwendung, beim Wechsel der aktiven Anwendung usw.). Bei Aktionen, welche die Umgebung betreffen, kann die Autorisierung hierfür überprüft werden. Bei Anwendungsaktionen ist dieses hingegen Sache der Anwendung selbst.
- **Phase 3 — Ermitteln der Ergebnisdaten**: Die vom ECA aus der Anfrage extrahierten Daten werden durch das Gateway an die betroffene Anwendung bzw. deren Shadow weitergeleitet und dort in den Ereigniskanal eingeleitet. In diesem Fall wird also das Ereignis generiert, das beim Drücken des Knopfes zum Hinzufügen eines weiteren Termins mit der Maus entstehen würde. Das Ereignis wird von der Anwendung interpretiert und führt zum entsprechenden Resultat, hier also dem Erscheinen eines Dialogfensters. Die Ergebnisdaten werden wiederum vom Shadow interpretiert und an das Gateway zurückgesendet.
- **Phase 4 — Transformation der Ereignisdaten**: Das Gateway wählt, wie zuvor, anhand der Sitzung den geeigneten Transformator aus und sendet ihm die vom Shadow empfangenen Daten zur weiteren Umwandlung. Unter Zuhilfenahme des entsprechenden Regelwerkes werden die Daten in die für das WAP Telefon passende Repräsentation umgesetzt und dann an das Gateway zurückgegeben.
- **Phase 5 — Rückgabe an den Klienten**: Das Gateway übergibt die nun in WML vorliegenden Daten über den Inhalt des Dialogfensters an den ECA, von wo diese über den Umweg des WAP Gateways wieder zurück zum WAP Telefon gelangen. Auf dem Telefon wird nun der Inhalt des Dialogfensters angezeigt, der Prozess kann erneut mit Schritt eins beginnen.

Im Beispiel des Terminplaners wurde eine bestehende Anwendung durch die Fabrik mit einem Shadow versehen und so in die JBSA integriert. Die abstrakte Beschreibung wurde durch den Shadow generiert, der die Anwendung quasi beobachtete und Veränderungen ihres Aussehens registrierte bzw. durch Modifikationen des Ereigniskanals herbeiführte. Die Entwurfsalternative hierzu würde so aussehen, dass die Anwendung unter Verwendung einer Programmierbibliothek so erstellt werden würde, dass die Funktionalität des Shadows direkt in

der Anwendung selbst läge. Eine Untersuchung der Benutzungsschnittstelle zur Generierung der abstrakten Beschreibung wäre hierbei ebenso wenig notwendig wie die nachträgliche Beeinflussung des Ereigniskanals zur Steuerung der Anwendung. Beides würde durch die Funktionen der Programmierbibliothek übernommen und direkt durch die Anwendung gesteuert.

Diese Entwurfsalternative soll anhand des bereits eingeführten Beispiels eines Messsystems zur Überwachung von Temperaturen verdeutlicht werden. Wie gesagt, hat der Treibercode eines Sensors die Aufgabe, den aktuellen Messwert abzufragen und von außen zugreifbar zu machen. Soll die Abfrage der Temperatur von einem beliebigen Endgerät erfolgen, so eignet sich hierfür der Einsatz der JBSA. Mit Hilfe der JBSA Programmierbibliothek wird dazu eine Benutzungsschnittstelle entworfen, über welche der Wert angezeigt bzw. aktualisiert werden kann. Von einem entsprechenden Werkzeug wird die Benutzungsschnittstelle mit der notwendigen Netzwerkschnittstelle verbunden (die Entsprechung zum Shadow) und beides dann mit dem JiSDA Compiler in einem Treiber zusammengefasst, der in den Sensor integriert wird.

Der Sensor meldet sich nach dem bereits beschriebenen Verfahren beim ALUS an und steht so zur weiteren Nutzung zur Verfügung. Soll nun beispielsweise von einem WAP Telefon aus die Temperatur eines Sensors abgefragt werden, so erfolgt dieses in einer Abwandlung des zuvor bereits für den Terminplaner beschriebenen Schemas. Zunächst wird über das WAP Gateway eine Verbindung mit dem ECA aufgebaut und eine Gerätesitzung und – nach erfolgreicher Anmeldung – auch eine Benutzersitzung erstellt. In diesem Falle würde nicht direkt eine Anwendung gestartet, sondern zunächst vom Gateway ein Dialog erstellt werden, in dem der Benutzer die gewünschte Anwendung anhand von Attributen spezifizieren kann, welche das zugehörige Dienstangebot im LUS kennzeichnen. Der Anwender würde vielleicht die Stichworte „Temperatursensor“ und „Wohnzimmer“ eingeben und das Gateway würde beim ALUS nach passenden Dienstangeboten suchen.

Konnte ein passendes Angebot gefunden werden, so wird dieses vom ALUS an das Gateway übergeben und kann von diesem zur Instantiierung an die Fabrik weitergeleitet werden. Die Fabrik startet das gefundene Treiberprogramm, welches dann seinen Zugangspunkt an das Gateway zurückmeldet, das diesen wie im Falle des Shadows in der Benutzersitzung zur weiteren Verwendung vermerkt. Gleichzeitig baut der Treiber eine Kommunikationsverbindung zum Sensor auf, mit welchem er in der Folge kommunizieren will.

Das Gateway nutzt den Zugangspunkt des Treibers sogleich zur Abfrage der ersten Momentaufnahme. Es sendet eine Anfrage an den Treiber, der zur Aktualisierung seiner Benutzungsschnittstelle den Sensor abfragt und mit den erhaltenen Daten die abstrakte Benutzungsschnittstelle erstellt. Die generischen Daten werden an das Gateway zurückgeschickt und von diesem an den Transformator weitergereicht, der diese dann wie zuvor beschrieben umformt. Die so entstandene Repräsentation wird vom JBSA Gateway an den ECA weitergereicht und von diesem über das WAP Gateway an das Telefon zurückgeleitet. Die weitere Nutzung unterscheidet sich dann nicht mehr von der Nutzung einer Kombination aus Anwendung und Shadow. Ein ums andere Mal werden die beschriebenen fünf Phasen zwischen Anfrage und Rückmeldung durchlaufen, bis der Treiber gestoppt und die Verbindung zum Messsystem abgebaut wird. Weitere Details

werden, wie gesagt, im nachfolgenden Kapitel besprochen.

5.3 Klassifikation der Integrationsumgebung

Im Verlauf der bisherigen Beschreibung wurde von Hydeparc sowohl als Integrationsinfrastruktur wie auch als Integrationsumgebung gesprochen. Beiden Begriffen liegt eine eher umgangssprachliche Färbung zugrunde und es stellt sich entsprechend die Frage, wie die von Hydeparc erbrachte Dienstleistung am besten bezeichnet werden kann. In der Literatur hat sich für Unterstützungssysteme, wie Hydeparc eines darstellt, der Begriff der Middleware eingeführt. In Abschnitt 5.2 wurde bereits festgestellt, dass Hydeparc auf der Middleware JINI aufsetzt. Kann es somit selbst als Middleware bezeichnet werden? Oder geht es noch über das Angebot einer Middleware hinaus und muss sich dieses in seiner Bezeichnung niederschlagen? Dieser Abschnitt dient einer entsprechenden Standortbestimmung.

5.3.1 Klassifikation als Infrastruktur oder Umgebung

Umgangssprachlich kann Hydeparc zunächst als Infrastruktur oder Umgebung bezeichnet werden. Der aus dem Gebiet der Topologie stammende Begriff „Umgebung“ bezeichnet die Menge *„[...] aller Punkte einer Ebene, deren Abstand von einem Punkt P kleiner ist als eine feste positive, reelle Zahl[...]“* [Bro52a]. Bezogen auf die Softwareentwicklung kann eine Umgebung somit als die Menge aller Programme aufgefasst werden, die hinsichtlich ihres Einsatzzwecks eng aneinander gebunden ist. Den gemeinsamen Einsatzzweck stellt für Hydeparc die Integration mobiler Geräte dar. Entsprechend lässt sich Hydeparc also als Integrationsumgebung bezeichnen.

Ein ähnlicher, wenngleich nicht mathematischer, sondern militärischer oder ökonomischer Wortstamm liegt dem Begriff „Infrastruktur“ zugrunde. Eine Infrastruktur ist laut Lexikon *„[...] der Unterbau einer Organisation [...]“* bzw. *„[...] die zusammenfassende Bezeichnung für militär. Anlagen wie Kasernen, Flughäfen, Tankstellen, Radarstationen u.a.“* [Bro52b]. Wiederum lässt sich der Begriff auf die Softwareentwicklung übertragen. Die Komponenten von Hydeparc dienen dem gemeinsamen Zweck der Einbindung von Geräten in einen Anwendungskontext. Hydeparc kann somit als Integrationsinfrastruktur bezeichnet werden.

Da sich „Infrastruktur“ und „Umgebung“ nur in ihrem Wortstamm unterscheiden, soll die im Rahmen des Projektes Hydeparc entstandene Software im Folgenden als Integrationsumgebung oder Integrationsinfrastruktur bezeichnet werden, ohne dass hierbei ein qualitativer Unterschied impliziert wird.

5.3.2 Klassifikation als Middleware

Bereits in Abschnitt 4.4.1 sowie 4.4.2 wurde näher auf die Bedeutung von Middleware für die Integration von Mobilgeräten eingegangen. Nachdem die Hydeparc

Umgebung in ihren grundsätzlichen Eigenschaften vorgestellt worden ist, stellt sich nun die Frage, ob Hydeparc selbst als Middleware bezeichnet werden kann.

Eine genauere Untersuchung ergibt, dass die Hydeparc Integrationsinfrastruktur in fast allen Punkten den von Bernstein (vgl. Abschnitt 4.4.1) formulierten Eigenschaften einer Middleware entspricht. Betrachten wir zunächst die typischen Eigenschaften von Middleware.

Keine Einschränkungen hinsichtlich Anwendung bzw. Kontext : In Hydeparc geschieht die Integration durch Introspektion der eingeklinkten Anwendungen. Dabei wird lediglich die Benutzungsschnittstelle analysiert, ohne dass das Anwendungsziel bzw. der Kontext eine Rolle spielt.

Auf verschiedenen Computersystemen lauffähig : Hydeparc ist plattformunabhängig in mehreren Dimensionen: Einerseits wird bereits durch die Realisierungssprache Java die Verwendung auf verschiedenen Plattformen ermöglicht. Darüber hinaus ist durch das modulare Design auch bezüglich der unterstützten Mobilgeräte bzw. der zu integrierenden Anwendungen keine Einschränkungen auf bestimmte Typen oder Plattformen festgeschrieben. Weitere Typen können jederzeit durch zusätzliche Module unterstützt werden.

Verteilt lauffähig : Die verschiedenen Komponenten der Hydeparc Umgebung sind durch Java RMI miteinander gekoppelt. Sie können daher beliebig in einem Netzwerk angeordnet werden. Auch die Einbindung externer Anwendungen erfolgt über Netzwerkverbindungen und ist somit verteilt.

Unterstützung standardisierter Schnittstellen und Protokolle : Auch diesbezüglich bietet Hydeparc eine Unterstützung in mehreren Dimensionen. Zunächst kommunizieren die Komponenten der Umgebung mittels standardisierter Protokolle und offener Schnittstellen. Weitere Komponenten können entsprechend nachträglich entwickelt und hinzugefügt werden. Über die Komponente zur Integration von Endgeräten kann darüber hinaus jedes Kommunikationsprotokoll unterstützt werden, das zur Interaktion mit Mobilgeräten veröffentlicht bzw. normiert ist.

Mit Blick auf ihre grundlegenden Eigenschaften lässt sich die Hydeparc Umgebung also zunächst als Middleware auffassen. Insbesondere ist hierbei die universelle Verwendbarkeit ausschlaggebend. Während verteilte Lauffähigkeit oder Plattformunabhängigkeit dank der zunehmenden Verbreitung von Java als Programmiersprache mittlerweile recht einfach für die meisten Programme zu erreichen ist, handelt es sich bei der Unterstützung standardisierter Schnittstellen und Protokolle sowie der Unterstützung beliebiger Anwendungen und Kontexte um Entwurfsmerkmale des Anwendungssystems, die nicht durch die verwendeten Werkzeuge erzwungen werden können, sondern durch diese allenfalls vereinfacht werden.

Bernstein nennt in [Ber96] darüber hinaus Unterstützungsdienste, die eine Middleware typischerweise anbietet oder anbieten könnte. Auch diese lassen sich in der Hydeparc Infrastruktur fast ausnahmslos zuordnen, wie die folgende Aufzählung zeigt:

Presentation management : Repräsentiert durch die Komponente zur Einbindung von Endgeräten. Diese bereitet die aus der Anwendung stammenden Informationen in einer Weise auf, wie sie vom Endgerät dargestellt werden können.

Computation : Repräsentiert durch das Modul zur Umwandlung der in abstrakter Form vorliegenden Schnittstellenbeschreibung in eine konkrete Repräsentation, die vom Endgerät dargestellt werden kann.

Information management : Repräsentiert durch die Module zur Verwaltung von Anwendungsklassen und Endgeräteklassen, die eine hierarchische Verwaltung von typspezifischen Merkmalen in einem Typsystem erlauben.

Communications : Repräsentiert durch die Komponente zur Einbindung von Endgeräten. Insbesondere das Modul zur Benachrichtigung von Endgeräten über den sich ändernden Zustand einer Anwendung sendet Nachrichten nicht nur innerhalb der Umgebung, sondern kommuniziert auch mit externen Entitäten.

System management : Repräsentiert z.B. durch die Kontroll– und Koordinationskomponente. Diese stellt die zentrale Verwaltungseinheit der Umgebung dar und bietet diejenigen Dienstleistungen an, die von allen Komponenten und deren Modulen genutzt werden. Weiterhin vertreten durch die Module zur Benutzerverwaltung sowie der Zugangskontrolle.

Control : Beispielsweise repräsentiert durch die Sitzungsverwaltung. Diese ordnet laufende Anwendungen einem Benutzer zu und ermöglicht innerhalb einer Benutzersitzung den Wechsel der zu einem Zeitpunkt aktiven Anwendung.

Sowohl entsprechend ihrer Eigenschaften als auch bezüglich der von ihr angebotenen Dienste lässt sich die Hydeparc Infrastruktur also als Middleware bezeichnen.

Als abschließender Nachweis soll ein Zitat von Bernstein zur Beweisführung herangezogen werden, in welchem der Autor die wichtige Rolle von Middleware zur Einbindung veralteter Anwendungen (sog. Legacy applications) in verteilte Systemumgebungen herausstreicht:

> *„One can also provide an advanced user interface to a legacy application through high–level presentation middleware services, for example, by intercepting character terminal I/O and translating it into operations on a graphical user interface (GUI). “* [Ber96]

Auch wenn Hydeparc nicht der Einbindung von veralteten Applikationen mit zeichenorientierten Bildschirmmasken in ein Grafiksystem im eigentlichen Sinne dient, lässt sich diese Aussage auf die Dienstleistung der Integrationsinfrastruktur übertragen, denn in einem gewissen Sinne lassen sich die eingebundenen Anwendungen sehr wohl als veraltet bezeichnen: bei den meisten dieser Anwendungen stand die Nutzung von mobilen Geräten nicht im Vordergrund oder war

gar vollständig ungeplant. Die Hydeparc Integrationsinfrastruktur stellt für diese Anwendungen also durchaus eine nachträgliche Erweiterung ihrer Präsentationsfähigkeiten dar, wie sie auch im angeführten Zitat als typische Anwendung von Middleware dargestellt wird.

Zusammenfassend lässt sich also feststellen, dass sich die Hydeparc Integrationsinfrastruktur als Middleware bezeichnen lässt. Sie zeigt die wesentlichen Merkmale einer Middleware, implementiert deren typische Dienste und übernimmt für Middleware typische Aufgaben.

5.3.3 Klassifikation als Framework

Als Spezialisierung des Middlewarebegriffs taucht in der Literatur das Konzept des Rahmenwerks (engl. *framework*) auf. Rahmenwerke sollen die Bedienung von Middleware vereinfachen, indem sie ihnen eine einheitliche Schnittstelle geben, sie um eventuell notwendige eigene Dienstleistungen erweitern und ihnen Werkzeuge beiordnen, die den Anwendungsentwurf vereinfachen. Rahmenwerke werden insbesondere eingesetzt, um die generische Dienstleistung der Middleware zu bündeln und diese auf ein bestimmtes Anwendungsgebiet hin zu optimieren.

> *„A framework is a software environment that is designed to simplify application development and system management for a specialized application domain. [...] Some popular types of frameworks include office system environments [...], TP Monitors [...], computer–aided software engineering workbenches [...], and system management workbenches [...].“* [Ber96]

Wird Middleware aufgrund ihres breiten Einsatzgebietes oft als „horizontales“ Konzept bezeichnet, so ist das Rahmenwerk mit seiner Beschränkung auf einen wohldefinierten Kontext als „vertikales“ Konzept zu bezeichnen. Rahmenwerke führen gewissermaßen eine Selektion hinsichtlich der von einer Middleware angebotenen Schnittstellen, Protokolle oder Dienste durch und stellen dem Anwender eine Auswahl zur Verfügung, die im Kontext des unterstützten Anwendungsgebietes als nutzbringend angesehen wird.

Auch die zur Verfügung gestellten Werkzeuge gehen in diesem Sinne über die Werkzeuge der Middleware hinaus. Die von einem Rahmenwerk angebotenen Werkzeuge setzen zumeist selbst auf den Schnittstellen des Rahmenwerkes auf und sind ebenso wie das Rahmenwerk selbst auf das jeweilige Anwendungsgebiet beschränkt. Dabei werden unter Umständen nicht nur die Schnittstellen eines der zugrunde liegenden Middleware zuzuordnenden eingesetzt, sondern gegebenenfalls auch die Schnittstellen mehrerer Dienste in Anspruch genommen.

Kann die Hydeparc Infrastruktur nun als Rahmenwerk bezeichnet werden? Mehrere Argumente sprechen dagegen:

- Hydeparc ist in seiner Dienstleistung nicht auf eine bestimmte Domäne eingeschränkt und
- Hydeparc bietet keine spezifischen Werkzeuge an, welche der Anwender zur Implementierung domänenspezifischer Applikationen nutzen kann.

Während in der Literatur keine eindeutige Definition für Rahmenwerke gegeben wird, ist deren vertikale Ausrichtung jedoch zweifelsfrei als Alleinstellungsmerkmal gegenüber dem Konzept der Middleware anzusehen. Rahmenwerke nutzen die Schnittstellen der unterliegenden Middleware ausschließlich zur Unterstützung von Bürotätigkeiten, zur Transaktionsverarbeitung oder zur Softwareentwicklung. Da es gerade die universelle Einsetzbarkeit jenseits von spezifischen Anwendungsfeldern ist, die Hydeparc als Integrationsinfrastruktur ausmacht, kann es somit kaum als Framework bezeichnet werden.

5.3.4 Klassifikation als Meta–Middleware

In jüngster Zeit ist neben den Ansatz, durch Middleware eine Abstraktionsschicht zwischen Anwendung und Betriebssystem zu schaffen, ein weiteres Konzept getreten. Vor allem durch die Zunahme voneinander unabhängiger und zumeist inkompatibler Middleware Systeme entstand das Bedürfnis nach Vereinheitlichung, das bereits als Ausgangspunkt für die Entwicklung von Middleware Systemen selbst ausschlaggebend gewesen war. Während es bei der Entstehung des Middleware Konzeptes die übergroße Zahl an Betriebssystemen, Netzwerkschnittstellen und Hardwareplattformen einerseits und die vielfältigen Probleme der Verteilung andererseits waren, auf die bei der Anwendungsentwicklung Rücksicht genommen werden musste, so liegt dem Konzept der Meta–Middleware die wachsende Zahl von Middleware Systemen selbst zugrunde.

Zum Begriff der Meta–Middleware existieren derzeit noch keine formalen Definitionen, und somit ist man in der begrifflichen Abgrenzung weitgehend auf die Betrachtung von Merkmalen angewiesen, welche vorhandene Meta–Middleware Systeme von herkömmlichen Middleware Systemen unterscheiden. Diese lassen sich wie folgt zusammenfassen:

- Meta–Middleware liegt zwischen Middleware– und Anwendungsschicht.
- Meta–Middleware bietet der Anwendung eine einheitliche Schnittstelle, in der sie die gemeinsamen Funktionen zugrundeliegender Middleware Systeme zusammenfasst.
- Meta–Middleware ist nicht auf eine bestimmte Middleware beschränkt, sondern kapselt ggf. mehrere Middleware Schnittstellen (hierin gleicht sie dem Konzept des Rahmenwerks).
- Meta–Middleware ist ebenso wie Middleware nicht auf eine bestimmte Anwendung oder einen bestimmten Anwendungskontext beschränkt. Sie ist also universell einsetzbar (hierin liegt ein wichtiges Unterscheidungsmerkmal zum Rahmenwerk).

Abbildung 5.5 skizziert die prinzipielle Anordnung von Betriebssystem, Middleware, Meta–Middleware und Anwendung und streicht nochmals den Unterschied zwischen Middleware und Meta–Middleware heraus. Während Middleware die Besonderheiten der Laufzeitumgebung vor der Anwendungsschicht verdeckt, kaschiert die Meta–Middleware die Unterschiede verschiedener Middleware Systeme. „Meta“ wird einem Begriff beigeordnet, um zu betonen, dass dieser ein

Objekt mit einer Anordnung zwischen anderen Objekten kennzeichnet. Meta–Middleware liegt also als eine besondere Form von Middleware zwischen der eigentlichen Middleware und der Anwendungsebene.

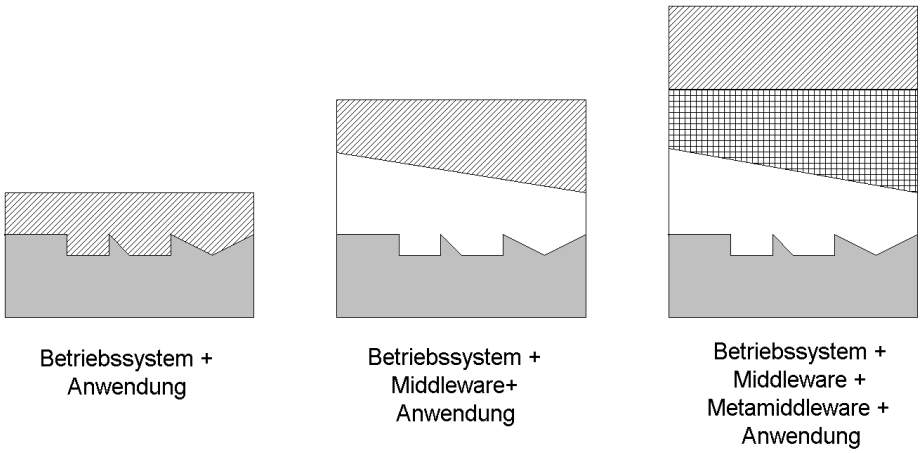


Abbildung 5.5: Betriebssystem, Anwendung, Middleware und Meta–Middleware im Zusammenspiel

Wie passt die Hydeparc Umgebung nun in dieses Modell? In diesem Kapitel wurde bereits erläutert, dass Hydeparc die Merkmale einer klassischen Middleware zeigt und somit als eine solche bezeichnet werden kann. In seinem Funktionsumfang geht Hydeparc jedoch noch über das Konzept einer klassischen Middleware hinaus: Mit seiner Komponente zur Einbindung von Endgeräten bietet es nicht nur eine bestimmte Abstraktionsebene zwischen Endgerät und Laufzeitumgebung. Indem das Integrationsmodul jederzeit den Besonderheiten eines weiteren Gerätetyps angepasst werden kann und daher nachträglich weitere Abstraktionsebenen eingeführt werden können, stellt es also nicht nur die Abstraktionsschicht einer Middleware zur Verfügung, sondern eine erweiterbare Menge an Middleware–Abstraktionen. Und die Unterstützung mehrerer Middleware Systeme ist gleichsam ein erster Hinweis darauf, dass es sich bei Hydeparc nicht nur um eine Middleware, sondern vielmehr um eine Meta–Middleware handeln könnte. Die Integrationsinfrastruktur kann also als Schicht zwischen Endgerät und Anwendung aufgefasst werden, was so zunächst einmal schlüssig ist.

Wenn es sich bei Hydeparc um eine Meta–Middleware handeln soll, so stellt sich die Frage, welchem Part im Gesamtsystem die Rolle einer Middleware zukommt, die von Hydeparc als Meta–Middleware verdeckt wird. Für Hydeparc gibt es auf diese Frage mehrere Antworten, die im weiteren Verlauf dieses Abschnitts diskutiert werden sollen.

Um sich dieser Frage zu nähern, sollen an dieser Stelle noch mal die Entitäten aufgezählt werden, die im Zusammenhang mit der Integration von Endgeräten durch die Hydeparc Umgebung eine Rolle spielen. An erster Stelle ist hierbei das Endgerät zu nennen, das eingebunden werden soll. Auf dem Endgerät läuft eine generische Anwendungssoftware, mittels der über ein Kommunikationsmedium mit der Integrationsinfrastruktur Kontakt aufgenommen werden kann; im Falle eines Mobiltelefons kann dieses beispielsweise ein WAP Browser sein. Der

Browser interagiert mit einem Unterstützungssystem, über welches er Daten an die Hydeparc Umgebung überträgt oder Nachrichten von dieser entgegennimmt. Im Falle des WAP Telefons wird das Unterstützungssystem einerseits durch das WAP Gateway verkörpert, andererseits durch den Web–Server, in den das Integrationsmodul der Hydeparc Umgebung eingebettet ist. Durch die hieran anschließenden Hydeparc Komponenten werden Befehle an das Anwendungsprogramm weitergeleitet. Das Anwendungsprogramm kann die Befehle nun seinerseits über ein Unterstützungssystem an einen entfernten Diensterbringer übergeben und diesen zur Änderung seines Zustands nutzen. Zustandsänderungen im Anwendungsprogramm werden von der Integrationsinfrastruktur aufgenommen und auf entgegengesetztem Weg wieder zurück an das Endgerät übertragen. Abbildung 5.6 verdeutlicht nochmals schematisch die Beziehung zwischen den beteiligten Komponenten.

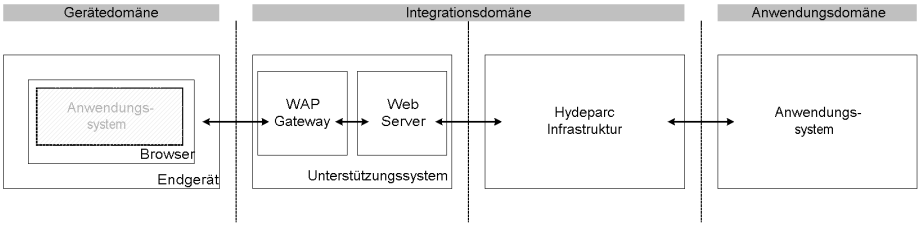


Abbildung 5.6: An der Integration beteiligte Entitäten im Überblick

Im bisherigen Verlauf dieses Kapitels wurde mehrfach auf das Anwendungsprogramm Bezug genommen, auf das durch die Hydeparc Umgebung hindurch zugegriffen wird. Da neben diesem Anwendungsprogramm im Folgenden auch die auf dem mobilen Endgerät laufende Anwendung betrachtet werden soll, ist eine begriffliche Unterscheidung zwischen diesen Anwendungen notwendig. Demgemäß soll nunmehr Folgendes gelten: Wird von der „eigentlichen Anwendung“ gesprochen, so ist hiermit das Programm gemeint, auf das durch Hydeparc hindurch zugegriffen wird. Der Begriff „Anwendung“ soll hingegen das Programm bezeichnen, das auf dem Endgerät befindlich ist und das dem Zugriff auf die Hydeparc Umgebung selbst dient (z.B. der WAP Browser). Fasst man die Integrationsinfrastruktur nun als Entität zwischen anderen Entitäten auf, so bleiben einem für die Definition von Anwendung und Middleware in dem in Abb. 5.5 dargestellten Schema nun folgende Möglichkeiten:

1. Hydeparc wird als Meta–Middleware aufgefasst, welche als Abstraktionsschicht die Middleware kapselt, die der eigentlichen Anwendung zugrunde liegt. In diesem Fall liegt zwischen Meta–Middleware und Middleware jedoch noch die eigentliche Anwendung.

2. Die eigentliche Anwendung stellt die Middleware dar, auf der die Integrationsumgebung als Meta–Middleware angeordnet ist. Der Browser auf dem Endgerät bzw. das zwischen Browser und Meta–Middleware liegende Unterstützungssystem fungiert als Anwendung.

3. Das Unterstützungssystem zwischen Endgerät und Integrationsinfrastruktur (also z.B. WAP) stellt im Sinne der schematischen Zeichnung aus Abb.

5.5 die Middleware dar, das Endgerät bzw. der hierauf laufende Browser die Anwendung.

Die genannten Zuordnungsalternativen sind in Abb. 5.7 zusammengefasst.

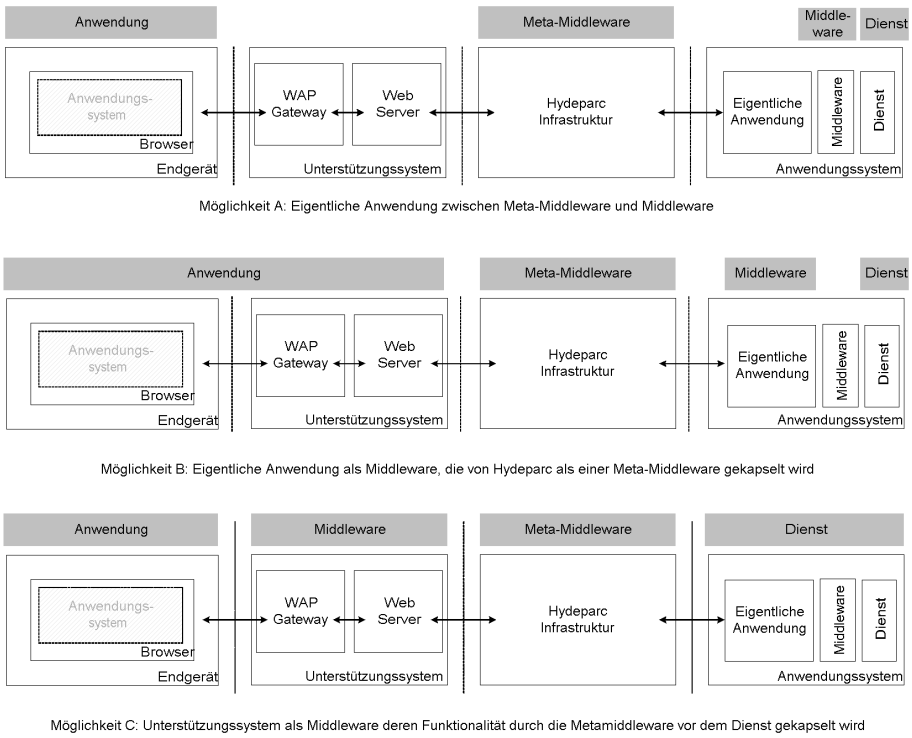


Abbildung 5.7: Drei Möglichkeiten zur Zuordnung der Begriffe ‚Anwendung', ‚Dienst', ‚Middleware' sowie ‚Meta–Middleware'

Welche der drei Alternativen als zutreffend zu bezeichnen ist, lässt sich nur schwer eindeutig beantworten, zumal in der Literatur eine diesbezügliche Diskussion bisher nicht zu finden ist. Für jede der Alternativen lassen sich sowohl Argumente als auch Gegenargumente finden und jede wirft ihrerseits Fragen auf.

Orientiert man sich zunächst an den identifizierbaren Komponenten, so scheint eine Zuordnung gemäß Alternative a) naheliegend. Hierbei wird davon ausgegangen, dass die eigentliche Anwendung, auf welche durch die Hydeparc Umgebung hindurch zugegriffen werden soll, ihrerseits nur Teil eines verteilten Informationssystems ist und selbst auf einer Middleware aufsetzt. Hydeparc ist in diesem Fall tatsächlich oberhalb der Middleware angesiedelt und realisiert eine Abstraktionsschicht zwischen dem Endgerät und dem darauf befindlichen Browser sowie der eigentlichen Anwendung und der von ihr genutzten Middleware. Änderungen in der Middleware müssen vom Endgerät nicht wahrgenommen werden, da der Zugriff auf die Middleware nicht direkt erfolgt, sondern durch Hydeparc entkoppelt und auf die eigentliche Anwendung begrenzt werden kann. Lediglich die eigentliche Anwendung selbst muss in diesem Fall angepasst werden, damit der Zugriff auf die geänderte Middleware erfolgen kann. Hydeparc kann daher nach Alternative a) als Meta–Middleware bezeichnet werden.

Was offensichtlich der bisherigen Definition von Meta–Middleware zuwiderläuft, ist die Tatsache, dass gemäß dieser Alternative die Meta–Middleware nicht direkt auf der Middleware aufzusetzen scheint. Während zwar offensichtlich eine Abstraktion durchgeführt wird, so findet diese nicht von der eigentlichen Middleware statt, sondern von der hierauf aufsetzenden eigentlichen Anwendung. Dies wirft einige grundlegende Fragen auf:

- Ist es zulässig, die Middleware, auf der die eigentliche Anwendung aufsetzt, mit in die Betrachtung aufzunehmen?
- Liegt Hydeparc als Meta–Middleware tatsächlich zwischen Anwendung und Middleware?
- Wird durch Hydeparc wirklich eine Abstraktion zwischen Anwendung und Middleware geschaffen?

Die erste Frage stellt quasi eine Grundannahme für die Tragfähigkeit der Alternative a) dar: Nur wenn die der eigentlichen Anwendung zugrunde liegende Middleware in das Betrachtungsfeld aufgenommen wird, kann die Bezeichnung von Hydeparc als Meta–Middleware überhaupt in Betracht gezogen werden. Ansonsten verbleibt hingegen allenfalls die Klassifikation als Middleware. Gegen eine Erweiterung des Blickfeldes spricht, dass die Kapselung der Middleware vornehmlich der eigentlichen Anwendung selbst zuzuschreiben ist. Änderungen der Middleware oder gar deren Austausch werden nicht durch Hydeparc abgefangen, sondern müssen ausschließlich durch Modifikationen der eigentlichen Anwendung ausgeglichen werden. Demgemäß wäre die Middleware aus der Sicht der Integrationsumgebung nicht existent, sie wäre ein internes Implementierungsdetail der eigentlichen Anwendung. Eine Rechtfertigung ergibt sich jedoch, wenn man den Standpunkt des Endgerätes wählt, von dem aus der Zugriff auf ein entferntes Informationssystem realisiert werden soll und auf das nur über eine Middleware zugegriffen werden kann, deren Nutzung direkt auf dem Gerät aus den in Abschnitt 2.3 genannten Gründen (Geschlossenheit, Ressourcenknappheit usw.) unmöglich ist. In diesem Fall ist die eigentliche Anwendung ausschließlich als Adapter zu verstehen, über den das Endgerät durch die Hydeparc Umgebung hindurch über die Middleware auf das entfernte Informationssystem zugreifen kann. Da die Realisierung solcher Zugriffspfade für Endgeräte gewissermaßen eine Grundmotivation für diese Arbeit darstellt, lässt sich die zeitweise Ausblendung der eigentlichen Anwendung (bzw. deren Verschmelzung mit der zugrunde liegenden Middleware) zum Zwecke der konzeptionellen Einordnung legitimieren.

Hieraus ergibt sich sogleich eine mögliche Antwort auf die Frage, ob Hydeparc gemäß Alternative a) tatsächlich als Meta–Middleware zwischen Anwendung und Middleware anzuordnen ist. Dieses ist nämlich vor allem eine Frage danach, welche beteiligte Komponente in diesem Fall als Anwendung bezeichnet werden soll. Verbleibt man auf dem Standpunkt des Endgerätes und fasst dabei die eigentliche Anwendung lediglich als Zugriffsadapter auf das entfernte Informationssystem auf, so kann nun der auf dem Endgerät befindliche Browser mit der von ihm dargestellten Benutzungsschnittstelle als Anwendung aufgefasst werden. Hydeparc liegt also zwischen dem Browser als Anwendung und der Verbindung aus eigentlicher Anwendung und der ihr zugrunde liegenden Middleware als Middlewareblock.

Die Abstraktionsleistung von Hydeparc lässt sich aus Alternative a) schließlich recht einfach ablesen. Vom Browser als Anwendung wird in der für das Endgerät spezifischen Weise, d.h. mittels dem Endgerät bekannter Protokolle, auf die Hydeparc Umgebung zugegriffen. Hydeparc realisiert sodann eine Abstraktion zu der Middleware (über die Nutzung der in diesem Bild mit ihr verschmolzenen eigentlichen Anwendung): Änderungen der Middleware werden von der eigentlichen Anwendung verdeckt und der Zugriff auf die eigentliche Anwendung wird ungeachtet des vom Endgerät verwendeten Zugriffspfades durch Hydeparc normiert.

Die Alternative b) stellt eine Betrachtungsvariante von Alternative a) dar. Hiernach wird von vornherein auf die Einbeziehung einer realen Middleware in das Sichtfeld verzichtet und die eigentliche Anwendung selbst als Middleware verstanden, über die der Zugriff auf entfernte Informationen erfolgt. Im Wesentlichen steht diese Variante also auf derselben argumentativen Grundlage wie die zuvor beschriebene Alternative a). Fraglich ist in diesem Fall, ob es zulässig ist, die eigentliche Anwendung selbst als Middleware zu bezeichnen. Hierin liegt gewissermaßen der Schwachpunkt dieser Alternative, denn per se kann kaum vorausgesetzt werden, dass eine Anwendung die in Abschnitt 4.4.1 genannten Merkmale einer Middleware zeigt. Die Bezeichnung der eigentlichen Anwendung als Middleware beruht mithin lediglich auf der Vermutung, dass die eigentliche Anwendung dem Zugriff auf ein entferntes Informationssystem über eine einheitliche Schnittstelle dient, welche von Hydeparc genutzt wird. Während dieses in Variante a) noch von daher angenommen werden kann, als die geforderte Existenz der Middleware dort nur durch das Vorhandensein eines entfernten Informationssystems gerechtfertigt ist, kann hiervon kaum mehr ausgegangen werden, wenn eine konkrete Middleware nicht länger gefordert wird. Insgesamt ist es also nur in wenigen Sonderfällen zulässig, die Anwendung als Middleware zu bezeichnen und Alternative b) folgerichtig kaum mehr zur Untermauerung der These heranzuziehen, dass Hydeparc als Meta–Middleware verstanden werden kann.

Kern der letzten Alternative ist es, die Blickrichtung umzukehren und nunmehr den Browser als Anwendung zu verstehen, die auf einer Middleware aufsetzt. Über standardisierte Protokolle greift diese Anwendung auf die Dienste der Middleware zu und stellt so den Kontakt zu einem entfernten Informationssystem her. Als Middleware wird in diesem Zusammenhang das Unterstützungssystem verstanden, das zur Realisierung des Zugriffspfades verwendet wird, über den das Endgerät auf die Hydeparc Umgebung zugreift, also beispielsweise WAP, das Worldwide Web oder das Mailboxsystem, über welches ein schnurloses Telefon angebunden würde. Hydeparc stellt in diesem Fall die Meta–Middleware dar, welche die spezifischen Eigenheiten der jeweiligen Middleware im Zugriff auf die eigentliche Anwendung zu verbergen hilft. Die eigentliche Anwendung dient schließlich einem beliebigen Zweck, ihr Aufsetzen auf eine Middleware wird nicht länger vorausgesetzt.

Während diese Alternative den Vorzug bietet, keine unnötigen Voraussetzungen bezüglich der eigentlichen Anwendung und ihrer Einbettung in eine Middleware zu implizieren, rückt in diesem Fall die Frage danach in den Mittelpunkt, ob die Bezeichnung des Unterstützungssystems als Middleware zu rechtfertigen ist. Weiterhin stellt sich wiederum die Frage, ob Hydeparc eine Abstraktionsschicht für Middleware darstellt und insofern als Meta–Middleware bezeichnet werden

kann. Schließlich muss geklärt werden, ob das Unterstützungssystem auch in diesem Zusammenhang als Middleware bezeichnet werden kann: Ein Ziel von Middleware ist es, den Zugriff auf ein entferntes Anwendungssystem durch normierte Schnittstellen zu vereinfachen. Im Zusammenhang mit der Hydeparc Umgebung ist der eigentlichen Anwendung jedoch nicht bewusst, dass eine solche Schnittstelle existiert, denn diese wird erst durch die Hydeparc Umgebung geschaffen – kann dennoch von einer Diensterbringer/Dienstnehmer–Beziehung auf der Basis von Middleware gesprochen werden, in welche Hydeparc nachträglich eingeschleift ist?

Der Nachweis, dass der Zugriffskanal zwischen Endgerät und Hydeparc Umgebung zurecht als Middleware bezeichnet werden darf, lässt sich besonders einfach am Beispiel des Wireless Application Protocol führen. Vergleicht man das allgemeine Konzept der Middleware mit WAP als einer konkreten Umsetzung, so lassen sich die wichtigsten Merkmale von Middleware leicht nachweisen:

- **Generizität**: WAP ist zunächst einmal generisch bezüglich seiner Einsatzmöglichkeiten. Es stellt keine prinzipiellen Anforderungen bezüglich des unterstützten Anwendungsgebiets, sondern eignet sich für fast beliebige Kontexte.
- **Plattformunabhängigkeit**: Auch ist WAP nicht auf bestimmte Plattformen eingeschränkt. Sowohl auf der Seite der Dienstnehmer als auch auf der Seite der Dienste kann eine Vielzahl von Geräten und Systemen miteinander kombiniert und zu einem integrierten Informationssystem verwoben werden.
- **Verteilung**: WAP stellt in sich eine verteilte Architektur dar, in welcher die Rollen des Dienstes sowie des Dienstnehmers fest verankert sind. Ein Großteil der in WAP definierten Mechanismen dienen der Etablierung optimierter Interaktionsbeziehungen über drahtlose Kommunikationsnetzwerke.
- **Nutzung standardisierter Schnittstellen und Protokolle**: Alle Wechselbeziehungen in WAP verlaufen nach offenen und von einem unabhängigen Herstellergremium festgelegten Protokollen. Ebenso bieten alle vorgeschriebenen und in einem zu WAP konformen Unterstützungssystem vorhandenen Dienste ihre Dienstleistungen an normierten Schnittstellen an.

Auch die für Middleware typischen Unterstützungsdienste lassen sich bei WAP zum Teil finden. Der in WAP über den sog. „Push Proxy Service“ realisierte Rückkanal zum Mobilgerät lässt sich beispielsweise der Gruppe der „Control Services“ genauso zuordnen, wie die Transaktionsunterstützung und die Unterstützung verschlüsselter Kommunkation der Gruppe der „System Management Services“. Schließlich lässt sich das WAP Gateway mit seiner Funktionalität zur Kodierung und Dekodierung in der Seitenbeschreibungssprache WML vorliegender Informationen und zur Umsetzung von Kommunikationsnachrichten in die speziellen WAP Nachrichtenformate wahlweise als Dienst dem „Presentation Management“ , dem „Communications Management“ oder dem „Computation Management“ zuordnen.

Ähnliches gilt auch für das Worldwide Web als Zugangskanal. Der Austausch von Informationen mit dem Endgerät erfolgt über HTTP, ein standardisiertes Kommunikationsprotokoll. Die Verarbeitung des Protokolls erfolgt über den Webserver, der wie das WAP Gateway eine Reihe von Diensten im Sinne einer Middleware anbieten kann. Neben der Basisfunktionalität zum Austausch von Webseiten können dies die Verschlüsselung von Daten, die Kodierung und Dekodierung in verschiedene Zielformate oder die Ausführung zusätzlicher Programme sein. Im Übrigen basiert das Worldwide Web auf derselben Architektur wie WAP und entsprechend können in ähnlicher Weise die bereits für WAP identifizierten Merkmale von Middleware erkannt werden.

Schließlich dürften sich die typischen Eigenschaften von Middleware auch bei Zugangskanälen feststellen lassen, auf die in dieser Arbeit nicht näher eingegangen wird, insbesondere dann, wenn es sich um einen Zugangskanal für eine ganze Klasse von Geräten handelt. Dieses beruht darauf, dass solche Zugangskanäle meist ohnehin nur realisierbar sind, wenn sie entlang der Entwurfskriterien einer Middleware konzipiert werden, also generisch, plattformunabhängig sowie verteilt sind und auf standardisierten Protokollen und Schnittstellen aufsetzen. Dieses gilt umso mehr im Bereich mobiler Systeme mit ihrem bereits mehrfach beschriebenen Spezialisierungsgrad, der die Interaktion nur noch auf der Basis definierter Standards überhaupt als möglich erscheinen lässt.

Wird der Unterstützungsdienst nun als Middleware bezeichnet, muss geklärt werden, inwieweit Hydeparc in Alternative c) als Meta–Middleware tatsächlich eine Abstraktion zwischen Anwendung und Middleware etabliert. Um dieses zu klären, sind die beteiligten Rollen in Tabelle 5.3.4 nochmals aufgeführt und zugeordnet.

Rolle	**Vertreten durch**
Anwendung	Browser auf Endgerät
Middleware	Unterstützungsdienst (WAP, HTTP o.ä.)
Meta–Middleware	Hydeparc Umgebung
Dienst	Eigentliche Anwendung

Tabelle 5.1: Zuordnung von Entitäten und Rollen in Hydeparc

Da in diesem Fall die Meta–Middleware nicht zwischen Anwendung und Middleware angeordnet ist, sondern zwischen Middleware und Dienst liegt, kann die von ihr durchgeführte Abstraktion also nur in einer Vereinheitlichung des Zugriffs von der Middleware auf den Dienst oder umgekehrt, vom Dienst auf die Middleware, bestehen. Nach den Ausführungen in Abschnitt 5.2 wird klar, dass nur der zuerst genannte Fall zum Tragen kommen kann: Indem die eigentliche Anwendung (hier in der Rolle des Dienstes) nichts von der Middleware bzw. Meta–Middleware weiß, die sie umschließt, kann sie nicht selbständig auf die Middleware zugreifen (der Einsatz von der Hydeparc JBSA Programmierschnittstelle sei hier zunächst ausdrücklich ausgeklammert). Die Interaktion zwischen der Anwendung auf dem Endgerät und der eigentlichen Anwendung geht immer von der Anwendung bzw. von der Hydeparc Umgebung aus. Es muss daher auch nur festgestellt werden, ob durch Hydeparc der Zugriff von der Middleware auf die eigentliche Anwendung vereinheitlicht wird. Betrachtet man nun die Funkti-

onsweise der Hydeparc Umgebung, so wird deren Wirken als Abstraktionsschicht zwischen Middleware und Dienst deutlich. Aus der Sicht des Dienstes ist es nicht erheblich, über welche Middleware die Anwendung eine Interaktionsbeziehung herstellt. Die Hydeparc Umgebung schirmt den Dienst vor den Eigenheiten der jeweils verwendeten Middleware ab und lässt die Nutzung der eigentlichen Anwendung wie eine direkte Interaktion über Maus und Tastatur wirken. Wird das Endgerät gewechselt und hierbei auf einen anderen Zugangspfad übergewechselt, so bedeutet das de facto den Übergang zu einer neuen Middleware. In diesem Fall kommt eine andere Adaptionskomponente in Hydeparc zum Einsatz, welche die Kommunikation des Endgerätes mit der eigentlichen Anwendung über diese veränderte Middleware möglich macht, ohne dass die eigentliche Anwendung den Wechsel bemerkt.

Schließlich stellt sich die Frage, ob es überhaupt gerechtfertigt ist, in Alternative c) von Middleware, Meta–Middleware und Dienst zu sprechen und diese Rollen wie geschehen zuzuordnen. Middleware erbringt insbesondere dort einen Mehrwert, wo sie im Kontext eines (zumeist heterogenen oder verteilten) Anwendungssystems verwendet wird. *„A middleware service is a general–purpose service that sits between platforms and applications“* [Ber96]. Nun wird der Unterstützungsdienst aus der Sicht der eigentlichen Anwendung erst in dem Augenblick zu einer Middleware, in dem die Hydeparc Umgebung zum Einsatz kommt und einen Durchgriff vom Unterstützungsdienst auf die eigentliche Anwendung möglich macht. Es kommt also quasi zu einem „Bootstrapping Problem“: Der Unterstützungsdienst wird erst in dem Augenblick zur Middleware, in dem die Hydeparc Umgebung einen Weg zwischen Anwendung und Dienst herstellt, die Meta–Middleware muss in diesem Fall also eigentlich konzeptionell vor der Middleware in das Bild eingetragen werden. Ist sie dann jedoch noch eine *Meta*–Middleware?

Wie lässt sich dieser Widerspruch auflösen? Middleware definiert sich als standardisierte Mittelschicht zwischen Anwendungen und den von ihnen genutzten Diensten. Obwohl hierbei zunächst oftmals eine bewusste Verwendung von Middleware zum Aufbau der Beziehung zwischen beiden Parteien vorausgesetzt wird, ist dieses nicht zwingend. Wie in [Ber96] ausgeführt wird, kann Middleware auch zur Einbindung von existierenden Anwendungen (engl. „legacy applications“) eingesetzt werden, die von ihrem nachträglich eingeführten Middleware–Kontext demnach keine Kenntnis besitzen. Es lässt sich also zunächst dafür argumentieren, zwischen Endgerät und eigentlicher Anwendung die nachträgliche Etablierung einer Middleware zuzulassen. Welcher Komponente steht nun die Bezeichnung als Middleware zu? Nach eben Gesagtem würde das bedeuten, dass Hydeparc, welches ja zur Kapselung des existierenden Systems verwendet wird, in diesem Fall also als Middleware bezeichnet werden müsste. Dem lässt sich jedoch entgegenhalten, dass Meta–Middleware als Abstraktion zwischen Anwendung und Middleware genauso zulässig ist, wie zwischen Middleware und Dienst. Geht man also davon aus, dass eine Middleware zur Kapselung existierender Anwendungen verwendet werden kann, so muss zwischen Middleware und der hier gekapselten eigentlichen Anwendung auch eine Meta–Middleware zur weiteren Abstraktion eingeschleift werden können, was genau durch Hydeparc geschieht. Fasst man im Übrigen Meta–Middleware als Vereinheitlichung mehrerer Middleware Schnittstellen auf, so wird deutlich, dass in Variante c) nur Hydeparc diese Rolle zufallen kann. Es ist demzufolge konsistent, vom Brow-

ser als Anwendung, dem Unterstützungsdienst als Middleware, Hydeparc als Meta–Middleware und der eigentlichen Anwendung als Dienst zu reden.

Wie gesehen kann Hydeparc in zwei Konstellationen als Meta–Middleware bezeichnet werden: Geht man davon aus, dass die eigentliche Anwendung, zu deren Kapselung Hydeparc verwendet wird, ihrerseits auf einer Middleware aufsetzt und über diese mit einem entfernten Informationssystem verbunden ist und sieht dabei die eigentliche Anwendung nur noch als Adapter, der aus der Sicht des Endgerätes notwendig ist, um auf die Middleware bzw. das entfernte Informationssystem zuzugreifen, so wird Hydeparc zu einer Meta–Middleware zur Abstraktion von der Middleware, welche der eigentlichen Anwendung zugrunde liegt. Fasst man andererseits das Unterstützungssystem, mit dem das Endgerät Nachrichten austauscht, als Middleware auf, so kann Hydeparc als Meta–Middleware zwischen dem Endgerät und der Anwendung verstanden werden, auf die mittels Hydeparc zugegriffen werden soll. Das zuletzt genannte Modell dürfte zu bevorzugen sein, da es keine Annahmen hinsichtlich Art und Anbindung der eigentlichen Anwendung macht. Ob die eigentliche Anwendung also selbst im Kontext einer Middleware abläuft, ist folglich unerheblich. Zudem ist hierbei die für Meta–Middleware geforderte Positionierung zwischen Anwendung und Middleware klar sichtbar, während diese im ersten Fall nur über das Hilfskonstrukt der Verschmelzung von eigentlicher Anwendung und Middleware zu einem logischen Block erreicht werden kann.

5.3.5 Klassifikation als Architektur in ‚0+n‘ Schichten

Unabhängig davon, ob es sich bei Hydeparc um eine Middleware oder deren Spezialisierung in Form einer Meta–Middleware handelt, lässt sich die Integrationsumgebung als eine Spezialform des zunehmend verbreiteten Ansatzes der Mehrschichtsysteme (Multi Layer Systeme, Multi Tier Systeme) begreifen. Der Begriff des Mehrschichtsystems liegt gewissermaßen auf höherem Abstraktionsniveau als derjenige der Middleware oder Meta–Middleware. Nachdem die vorigen Abschnitte also Ansätze für eine konkrete Klassifikation der Hydeparc Umgebung aufgezeigt haben, soll nun zum Abschluss eine Einordnung von Hydeparc als ‚0+n‘–Schichten Architektur motiviert werden.

Nachdem das Client/Server Computing in den achtziger und neunziger Jahren sehr erfolgreich zum Aufbau großer Anwendungssysteme eingesetzt wurde, haben Nachteile wie ein erhöhter Wartungsaufwand, komplexere Client—Anwendungen usw. in den vergangenen Jahren zu einer Rückbesinnung auf die zentralisierte Datenverarbeitung früherer Jahre geführt. Wurden Client/Server Anwendungen aufgrund ihrer Trennung in Anwendungslogik und Datenhaltung oft als Two–Tier Design bezeichnet, so hat nunmehr ein Übergang zum Anwendungsentwurf in drei Schichten (three tier design) Einzug gehalten. Hierbei wird die Anwendungslogik ihrerseits in zwei Teilbereiche aufgespalten und die Visualisierung der Daten aus der Anwendungslogik herausgelöst. Datenpräsentation und Datenaufbereitung werden zu eigenständigen Teilen des Anwendungssystems, wobei lediglich die zur Repräsentation notwendigen Anwendungsteile auf dem Endgerät angeordnet werden. Die zur Aufbereitung der Daten notwendige Anwendungslogik wird als so genannte *Business Logik* in das Netzwerk verlagert und fungiert als Mittler zwischen Datenhaltung und Datenvisualisierung.

Die Dreiteilung des Anwendungsentwurfs führt zu deutlichen Vorteilen gegenüber dem herkömmlichen Client/Server Computing:

- Die Einbindung neuartiger Endgeräte wird wesentlich vereinfacht. Es muss lediglich die für das Endgerät notwendige Datenvisualisierung implementiert werden, während die Anpassung der Anwendungslogik unterbleiben kann.
- Änderungen der Applikationslogik werden erheblich vereinfacht, da sämtliche Anwendungslogik in einem einzigen Bereich des Anwendungssystems zusammengefasst ist. Durchgeführte Änderungen werden automatisch von allen angeschlossenen Endgeräten realisiert.
- Da die Aufbereitung der Anwendungsdaten nicht durch das Endgerät selbst durchgeführt werden muss, können die hierfür notwendigen Ressourcen bei der Dimensionierung des Gerätes entfallen. Dieses ist insbesondere bei mobilen Endgeräten von Vorteil.
- Die Verlagerung der Anwendungslogik aus dem Endgerät heraus in das zentrale Netzwerk führt zu einer deutlich besseren Nutzung der Kommunikationsstrecke zwischen Endgerät und Anwendungslogik. Die Verknüpfung bzw. Verdichtung von Daten findet im Austausch zwischen Datenhaltung und Anwendungslogik statt, während zum Endgerät lediglich Resultate übertragen werden. Dieses ist insbesondere zur Einbindung mobiler Geräte über langsame Datenverbindungen von Vorteil.

Während der Anwendungsentwurf in drei Schichten dem klassischen Client/Server Computing insbesondere hinsichtlich einer Berücksichtigung mobiler Geräte überlegen zu sein scheint, existieren auch Nachteile. So müssen für den Austausch von Daten zwischen Endgerät und Anwendungslogik passende Datenrepräsentationen gefunden werden, welche genügend abstrakt sind, um die implizite Festlegung auf bestimmte Zieldarstellungen zu vermeiden, die gleichzeitig jedoch ausdrucksstark genug sind, um die effiziente Kodierung der zu übertragenden Informationen zu ermöglichen. Auch werden durch die Hinzunahme weiterer Schichten zwischen Darstellung und Datenhaltung die auftretenden Interaktionsbeziehungen immer komplexer. Vielfach können bewährte Methoden des Anwendungsentwurfs für verteilte Systeme im Entwurf von 3–Tier Systemen nicht länger eingesetzt werden. An ihre Stelle müssen neuartige Entwurfsmethoden treten, die oftmals noch wenig erprobt sind.

Die Aufteilung des Anwendungsentwurfs in drei Schichten stellt die am weitesten verbreitete Ausprägung der Mehrschichtsysteme dar. Neben dieser wichtigen Grundform können durch fortgesetzte Aufspaltung der existierenden Schichten zwischen Darstellung und Datenhaltung jedoch beliebige Verfeinerungen gebildet werden. Vier oder fünf Schichten sind nicht unüblich und weitere durchaus denkbar, wobei die Darstellung auf dem Endgerät jeweils das eine Ende des Kontinuums darstellt und die zentrale Datenhaltung das andere. In seiner allgemeinen Form wird die Gruppierung der funktionalen Teile eines Anwendungssystems in Schichten als „Entwurf in n Schichten“ oder auch als „n–Tier Design“ bezeichnet.

Hydeparc stellt einen Spezialfall des n–Tier Designs dar. Da kein direkter Eingriff in die Anwendungsarchitektur selbst stattfindet, ergibt sich die Aufteilung in mehrere Schichten nicht aus einer Aufspaltung des Anwendungssystems zwischen Darstellung und Datenhaltung. Wie bereits im Abschnitt über die Einordnung von Hydeparc als Meta–Middleware beschrieben wurde, lässt sich sogar argumentieren, dass in der Betrachtung von Hydeparc die innere Struktur der eigentlichen Anwendung ganz ausgeblendet werden muss und die Darstellungsschicht der Anwendung als einziger Berührungspunkt mit der Anwendung angesehen werden kann. Hiernach markieren bei der Nutzung der Anwendung von einem mobilen Endgerät genau genommen nicht Darstellung und Datenhaltung die äußeren Schichten der Architektur, sondern Darstellung auf dem Endgerät und Darstellung durch die eigentliche Anwendung. Wenngleich sich die Darstellung durch die eigentliche Anwendung durchaus als eine Form der Datenhaltung in Hydeparc definieren lässt, verbleibt doch die augenscheinliche Abweichung vom üblichen Aufbau einer mehrschichtigen verteilten Anwendung.

Mit der Bezeichnung als ‚0+n' Schichten–Architektur wird dieser Abweichung Rechnung getragen. Zunächst einmal handelt es sich bei Hydeparc im üblichen Sinne um ein Anwendungssystem, das in mehreren Schichten organisiert ist. Andererseits weicht es, wie gesagt, vom bisherigen Schema ab. Die Bezeichnung ruht demgemäß im Wesentlichen auf drei Säulen:

1. Hydeparc funktioniert ungeachtet der inneren Struktur der eigentlichen Anwendung. Deren Aufbau als verteilte Anwendung wurde zwar beim Entwurf der Hydeparc Umgebung in gewisser Hinsicht vermutet, Einflüsse auf den Entwurf gingen hiervon jedoch nicht aus. Während von der umklammerten Anwendung also nicht mit Bestimmtheit gesagt werden kann, wie viele Schichten sie besitzt, kann jedoch mit Sicherheit vorausgesetzt werden, dass sie aus ‚n' Schichten besteht.

2. Hydeparc greift nicht in die Struktur der umklammerten Anwendung ein, sondern umschließt sie in der bereits angedeuteten Weise. Durch Hydeparc wird der bestehenden Darstellungsschicht der Anwendung quasi eine weitere, externe Darstellung hinzugefügt. Dieses wird durch das vorangestellte Pluszeichen ausgedrückt.

3. Schließlich ist die durch Hydeparc realisierte Darstellungsschicht insofern bemerkenswert, als sie nicht nur extern zur eigentlichen Darstellungsschicht der Anwendung angeordnet ist, sondern vielmehr eine beliebige Form annehmen kann. So kann sie zwar durchaus eine hohe Komplexität und Ausdrucksmächtigkeit besitzen, andererseits aber auch so weit reduziert werden, dass sie auf nahezu beliebigen Mobilgeräten dargestellt werden kann. Durch die Wahl der Ziffer ‚0' als Kennzeichnung der von Hydeparc verkörperten Schicht wird zweierlei verdeutlicht. Zunächst wird hiermit daraufhin gewiesen, dass Hydeparc als zusätzliche Schicht vor der Darstellungsschicht der Anwendung als eigentlich erster Schicht positioniert ist. Außerdem soll Hydeparc genau eine so schmale und leichtgewichtige Darstellungsschicht realisieren, dass diese auch dann noch verwendet werden kann, wenn eine normale grafische Benutzungsschnittstelle nicht länger einsetzbar ist.

Die Darstellung in Abb. 5.8 soll das Gesagte nochmals verdeutlichen. Hydeparc ist als Darstellungsschicht vor der eigentlichen Anwendung positioniert. Die Anwendung selbst kann beliebig viele Schichten enthalten, denen Hydeparc selbst nur noch eine weitere, leichtgewichtige Darstellungsschicht voranstellt.

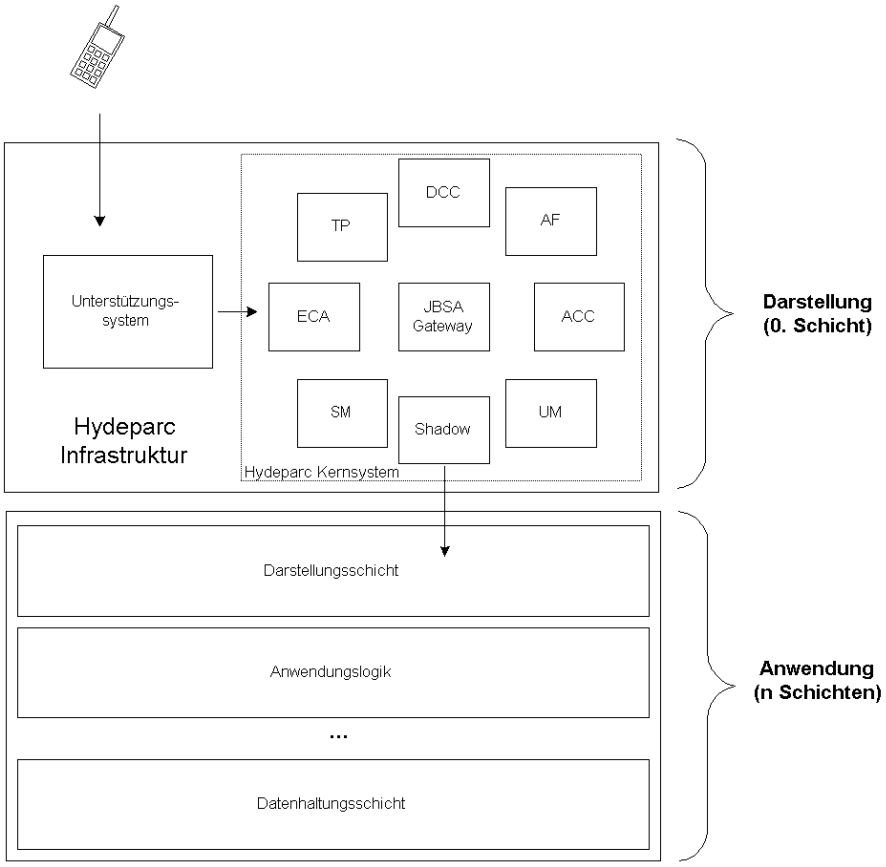


Abbildung 5.8: Hydeparc als Architektur von ‚0+n' Schichten

Die Bezeichnung von Hydeparc als ‚0+n'–Tier Architektur stellt vor allem ein Gedankenmodell dar, um die besonderen Eigenschaften zu betonen. Wählt man den allgemeineren Blickwinkel des Endgerätes, das eine Verbindung mit seinem Unterstützungssystem aufbaut, um auf entfernte Daten zuzugreifen, so lässt sich vom besonderen Dienst der Hydeparc Umgebung abstrahieren. Hydeparc wird zu einer Zwischenschicht, die zwischen der äußeren Darstellungsschicht und der Datenhaltung liegt und die Applikationslogik zur Umformung von Daten einer bestimmten Repräsentation (hier: einer Benutzungsschittstelle der eigentlichen Anwendung) in eine andere Repräsentation (hier: die vom Unterstützungssystem an das Endgerät zu vermittelnde Darstellung) und zum kontrollierten Zugriff auf diese bereitstellt. In diesem Sinne stellt die Hydeparc Umgebung dann lediglich den natürlichen Teil einer Multi–Tier Architektur dar.

5.4 Zusammenfassung

Dieses Kapitel bildete den Auftakt zum zweiten Teil der Arbeit über die Hydeparc Integrationsumgebung. Ziel war es, den notwendigen Überblick über die geschaffene Umgebung zu vermitteln, bevor in den nachfolgenden Kapiteln auf die Details eingegangen werden soll. Zunächst wurden dabei die Entwurfskriterien vorgestellt, die quasi das Fundament der Arbeiten an der Hydeparc Umgebung bildeten. Viele im Rahmen der Entwicklung getroffenen Entwurfsentscheidungen lassen sich direkt auf die in diesem Kapitel formulierten Kriterien zurückführen und demgemäß besser nachvollziehen, wenn man diese Kriterien kennt. Es zeigte sich, dass die meisten Kriterien sich einem gemeinsamen Ziel zuordnen lassen, nämlich einer maximalen Generizität: Minimale Abhängigkeiten sollen zwischen Diensten und Geräten bestehen, von denen diese Dienste genutzt werden. Die Geräte selbst sollen ungeachtet ihrer eigenen Fähigkeiten nutzbar sein, ohne dass hierbei Vorgaben hinsichtlich Hardware, Betriebssystem o.ä. erfüllt sein müssten.

Ausgehend von dem Kriterienkatalog wurde im Anschluss näher auf die eigentliche Integrationsinfrastruktur eingegangen, die auf der Basis der genannten Kriterien entwickelt wurde. Es wurden dabei Schwerpunkte formuliert, in denen sich die Funktionalität zusammenfassen lässt. Jeder der genannten Schwerpunkte wurde im Überblick beschrieben und in seinen wesentlichen Merkmalen herausgearbeitet. Wie gezeigt wurde, dient die Integration mobiler Endgeräte dazu, Zugriffspfade zu realisieren, über die auf die Integrationsumgebung zugegriffen werden kann. Zusammengesetzt aus internen und externen Komponenten sorgt sie dafür, dass die vielfältigen Kommunikationsmechanismen, wie sie von mobilen Geräten verwendet werden, auf ein einheitliches internes Protokoll innerhalb der Integrationsumgebung umgesetzt werden. Die Integration von Diensterbringern und Dienstnehmern basiert auf dem im vorigen Kapitel eingeführten Prinzip der Introspektion, bei dem die Analyse von Benutzungsschnittstellen über den Umweg einer allgemeinen Darstellung zu einer konkreten Repräsentation führt, die vom mobilen Gerät interpretiert werden kann. Diese Transformation ist es, was als Adaption zwischen Endgerät und Dienst bezeichnet wird. Die Kontrolle und Koordination der Integrationsumgebung obliegen schließlich der letzten Komponente, die in diesem Überblick beschrieben wurde.

Bereits in der Darstellung der vier Aufgabenbereiche zeigte sich, dass sich die Integrationsumgebung hinsichtlich der Unterstützung mobiler Geräte primär in zwei funktionale Schwerpunkte aufgliedern lässt: Die Integration mobiler Diensterbringer sowie die Integration mobiler Dienstnehmer. Es wurde gezeigt, dass diese Gliederung ihre Entsprechung in der Aufteilung der Integrationsinfrastruktur in die Java Border Service Architecture (JBSA) und die Jini Small Device Adapters (JiSDA) besitzt. Entlang der Aufteilung in diese zwei Bereiche wurde sodann näher auf die Funktionsweise der Infrastruktur eingegangen. Beginnend mit JiSDA wurde auf die Teilkomponenten der beiden Einheiten eingegangen, deren Arbeitsweise und Interaktion beschrieben. Anhand mehrerer Beispiele wurde des Weiteren versucht, die Nutzung der Integrationsinfrastruktur durch ein Mobilgerät exemplarisch zu beschreiben.

Im zweiten Teil dieses Kapitels wurde versucht, sich der Hydeparc Umgebung auf einer eher abstrakten Ebene zu nähern und die Dienstleistung, die durch

sie realisiert wird, begrifflich zu fassen. Es wurde ausführlich begründet, dass Hydeparc die typischen Merkmale einer Middleware besitzt und daher im allgemeinsten Fall als solche bezeichnet werden kann.

Im Weiteren wurde dann jedoch verdeutlicht, dass die Dienstleistung der Integrationsumgebung in wesentlichen Bereichen über die einer Middleware hinausgeht:

- Sie liegt als zusätzliche Schicht zwischen Anwendung und einer weiteren Middleware.
- Sie bietet eine einheitliche Schnittstelle zu verschiedenen zugrunde liegenden Middleware Systemen.
- Sie ist nicht auf bestimmte Middleware Systeme beschränkt.
- Sie ist nicht auf bestimmte Anwendungskontexte beschränkt.

Es wurden mehrere Ansätze vorgestellt, nach denen sich Hydeparc als Meta–Middleware einordnen lässt. Hierbei wurde klar, dass diese Ansätze unterschiedlich hohe Anforderungen an die Einbeziehung externer Software stellen und dass somit nicht jeder Ansatz gleichermaßen geeignet ist. Letztlich zeigte sich, dass sich der Begriff der Meta–Middleware ideal dann vertreten lässt, wenn das Unterstützungssystem des Mobilgeräts (WAP o.ä.) als Middleware aufgefasst wird, die von der Hydeparc Umgebung auf dem Anwendungskontext abgebildet wird.

Zum Abschluss wurde ein weiterer Benennungsansatz vorgestellt, der sich am logischen Aufbau der von der Hydeparc Umgebung realisierten Integrationsinfrastruktur orientiert. In der Art moderner Softwaresysteme in funktionale Schichten unterteilt, lässt sich die Hydeparc Umgebung als Schichtenmodell auffassen. Der letzte Abschnitt des Kapitels hat diesen Ansatz vertieft und dargelegt, wie Hydeparc als eine besondere Form eines solchen Schichtenmodells angesehen werden kann, in dem den bestehenden Schichten des Anwendungssystems eine weitere, leichtgewichtige „Nullte“ Integrationsschicht vorgeschaltet wird und das bestehende System so zu einem Modell in ‚0+n‘ Schichten zu erweitern ist.

Nachdem dieses Kapitel mit allgemeineren Betrachtungen die Beschreibung der Hydeparc Umgebung eingeleitet hat, sollen die folgenden beiden Kapitel im Detail auf die technische Realisierung eingehen, welche die Integrationsinfrastruktur im Inneren ausmacht. Dabei wird zunächst mit der Java Border Service Architecture begonnen und im Anschluss auf die Realisierung der Jini Small Device Adapters eingegangen.

Kapitel 6

Die „Java Border Service Architecture“ (JBSA)

Nachdem im vierten Kapitel die allgemeinen Konzepte beschrieben wurden, auf denen die Integration durch Abstraktion auf Anwendungsebene basiert und im vorigen Kapitel dann der Vorschlag für eine auf den genannten Konzepten basierende Infrastruktur im Überblick beschrieben wurde, soll in diesem Kapitel auf einige wesentliche Implementationsdetails eingegangen werden. Die Umsetzung des Integrationskonzepts in eine konkrete Infrastruktur besteht aus verschiedenen Modulen, die in ihrem prinzipiellen Zusammenspiel bereits im vorigen Kapitel vorgestellt wurden. Die folgenden Abschnitte beschreiben herausragende Aspekte der Realisierung eben jener Module.

Wie im vorigen Kapitel ausgeführt wurde, lassen sich die Konzepte der Java Border Service Architektur im Wesentlichen in neun funktionalen Modulen zusammenfassen, nämlich:

- Externen Kommunikationsadaptern,
- Device bzw. Application Class Catalog (DCC / ACC),
- Session Manager bzw. User Manager,
- Advanced Lookup Service (ALUS),
- Application Factory,
- Application Shadow sowie
- JBSA Gateway als zentralem Element der Umgebung.

In der prototypischen Umsetzung wurde an dieser Aufteilung prinzipiell festgehalten, wobei einzelne Module zunächst entsprechend ihrer Beziehung zueinander in gemeinsamen Anwendungen gruppiert wurden. So werden die Verwaltung von Benutzern und die Überprüfung von Zugangsberechtigungen vom Gateway übernommen und die Klassifikation von Geräte– und Anwendungsprofilen durch

die externen Kommunikationsadapter realisiert. Schließlich werden Selektion und Start von Anwendungen innerhalb der Integrationsumgebung in einer Kombination aus Application Factory und Advanced Lookup Service zusammengefasst. Eigenständige Module zur Verwaltung von Benutzern, zur Verwaltung von Anwendungs– und Geräteprofilen sowie zur Selektion von Anwendungen wurden in der prototypischen Umsetzung hingegen nicht realisiert. Andererseits wurde die Transformation von abstrakten Beschreibungen in konkrete Repräsentationen aus Gründen der Effizienz aus dem Gateway heraus fakturiert: Eine beliebige Menge von Transformatoren kann unabhängig voneinander die Übersetzung von Schnittstellenbeschreibungen übernehmen.

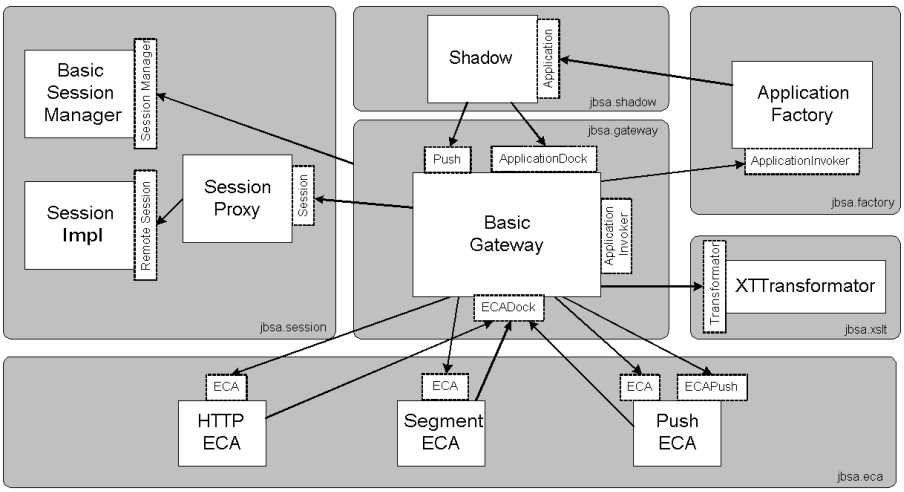


Abbildung 6.1: Module der prototypischen Umsetzung und deren Schnittstellen

Abbildung 6.1 zeigt die Aufteilung der Umgebung in Module. Die grau schattierten Kästen geben die Pakete des Prototypen wieder, wesentliche Klassen der prototypischen Umsetzung sind als Boxen darin zu sehen. Die Namen wichtiger Schnittstellen, auf die zum Teil im weiteren Verlauf noch eingegangen werden soll, sind in der Abbildung in gestrichelten Kästen eingezeichnet.

Mit der Betrachtung von Implementationsdetails soll zunächst in Abschnitt 6.1 mit einem kurzen Überblick über die verwendete Zwischenrepräsentation JSML begonnen werden. Die Abstraktionsschicht auf Seiten der Anwendung, die vornehmlich durch den *Application Shadow* und die *Application Factory* realisiert ist, wird dann im folgenden Abschnitt 6.2 näher erläutert. Die an entgegengesetzter Seite angeordnete Abstraktionsschicht zum mobilen Endgerät hin, repräsentiert durch die Externen Kommunikationsadapter, wird sodann in Abschnitt 6.4 beschrieben. Abschnitt 6.5 widmet sich schließlich den Kernteilen der Integrationsumgebung, die der Realisierung einer Sitzungssemantik dienen und die im JBSA Gateway sowie dem Sessionmanager zusammengefasst sind.

6.1 Die Auszeichnungssprache JSML

Wie bereits beschrieben, wird die Abstraktionsschicht zum Anwendungssystem hin auf der Ebene der Benutzungsschnittstelle realisiert. Alle sichtbaren Elemente des GUIs werden in ein abstraktes Format überführt und dann gemäß der Anforderungen des zugreifenden Endgerätes nach entsprechender Transformation in eine konkret darstellbare Zielrepräsentation zurückübersetzt. Essentiell für die Übertragung in prinzipiell beliebige Zielrepräsentationen ist die Nutzung eines universellen Zwischenformats. Dieses sollte keine inhärenten Abhängigkeiten von bestimmten Zielplattformen besitzen, um die Transformation in eine möglichst große Anzahl verschiedener solcher Formate zu erlauben und so die Verwendung auf möglichst beliebigen Zielplattformen zu ermöglichen. Um andererseits auch die Nutzbarkeit mit möglichst vielen Ausgangsrepräsentationen (d.h. Ansätzen zur Realisierung von Benutzungsschnittstellen außer Java SWING) zu erreichen, sollte dieses Zwischenformat auch diesbezüglich neutral konzipiert sein. Auch sollte die Transformation des Zwischenformats in die Zielrepräsentation effizient genug sein, um eine Nutzung des Mechanismus zur Laufzeit der Anwendung zu ermöglichen. Und schließlich sollte die Transformation durch Regeln gesteuert werden, die nicht in die Infrastruktur einkodiert sind und jederzeit erweitert werden können, sofern die Unterstützung weiterer Zielplattformen gewünscht ist.

Auf die konzeptionellen Aspekte einer adäquaten Lösung ist bereits in Kapitel 4 ausführlich eingegangen worden. Zur Abdeckung all dieser Anforderungen in der konkreten Umsetzung wurde die Nutzung von XML gewählt. XML ermöglicht die flexible Definition einer eigenen Auszeichnungssprache, die die oben formulierten Ziele in sich vereinen kann. In Kombination mit XSL können darüber hinaus Regelwerke formuliert werden, die eine effiziente Transformation von Dokumenten aus der Ausgangssprache in eine Zielsprache realisieren.

Die XML Anwendung JSML stellt das Resultat der geschilderten Bemühungen dar. Beim Entwurf von JSML stand die Abbildung von grafischen Benutzungsschnittstellen in einem abstrakten Datenformat im Vordergrund. Wenngleich die prototypische Realisierung auf der Basis von SWING, der GUI Bibliothek von Java 2, entstand und somit der Schwerpunkt zunächst auf der Unterstützung der in SWING vorkommenden Elemente einer Benutzungsschnittstelle lag, wurde dennoch von Beginn an auf die Erweiterbarkeit der Sprache Wert gelegt. Insofern wurde auch versucht, möglichst allgemeine Objekte innerhalb grafischer Benutzungsschnittstellen zu finden, die dann in der Sprache verankert wurden.

In JSML wird der Zustand einer grafischen Benutzungsschnittstelle zu einem bestimmten Augenblick kodiert. Wie mit einer imaginären Kamera wird ein Schnappschuss gemacht, der vom Application Shadow in eine interne JSML Datenrepräsentation übertragen und dann zur weiteren Transformation an das JBSA Gateway übergeben werden kann. In JSML müssen demnach keine Informationen über den inneren Zustand der betrachteten Anwendung erfasst werden. Vornehmlich sichtbare (und wenige unsichtbare, hierzu gleich mehr) Elemente der Benutzungsschnittstelle werden in XML repräsentiert und dann entweder lesbar (als XML Dokument) oder nicht unmittelbar lesbar (in einer dynamisch erstellten Datenstruktur, dem *Document Object Model*, DOM) ausgetauscht.

Ausgangspunkt im Entwurf von JSML war die typische Zusammensetzung

von grafischen Benutzungsschnittstellen in verschachtelten Datenstrukturen, oft auch als *Container* bezeichnet. So stellt das Fenster die oberste Ebene der Hierarchie einer Benutzungsschnittstelle dar. In ihr werden enthaltene Elemente wie Menüs, Rollbalken, Textfelder usw. als untergeordnete Datenstrukturen eingefügt, und auch diese können ihrerseits als Container für weitere Elemente herhalten. Die Beziehung zwischen den einzelnen Elementen lässt sich bildlich als eine Schichtung von Objekten darstellen, die miteinander in einer Hierarchie verbunden sind. Container können zum einen generisch sein. Generische Container dienen ausschließlich der Gruppierung der in ihnen enthaltenen Elemente, ohne dass sie tatsächlich auf dem Bildschirm sichtbar werden. Der allgemeinste dieser Container ist das Hauptfenster der Anwendung selbst. In SWING als „Frame“ bezeichnet, bildet dieses als generischer Container die Fläche, auf der alle weiteren Elemente der Benutzungsschnittstelle angeordnet werden können. Container werden auch verwendet, um zu entscheiden, welche Knöpfe einer grafischen Oberfläche gemeinsam als Gruppe verstanden werden sollen, in der immer nur ein Knopf ausgewählt sein darf. Die Objekte, welche die Knöpfe realisieren, werden hierzu gemeinsam in einem generischen Container eingeschlossen. Dieser spezielle Container wird z.B. in SWING als *Buttongroup* bezeichnet.

In JSML wurden beide Arten von Containern realisiert. Auf oberster Ebene der Hierarchie steht das `Application` Element. Alle weiteren Elemente der abstrakten Beschreibung der Benutzungsschnittstelle müssen in einem solchen Element zusammengefasst werden. Auf nächst niederer Ebene steht als weiterer generischer Container das `Container` Element. Jedes JSML Dokument kann über einen oder mehrere Container verfügen, die ihrerseits der Gruppierung von weiteren Elementen der Benutzungsschnittstelle dienen. Container können, wie in GUI Bibliotheken allgemein üblich, beliebig ineinander geschachtelt sein. Neben weiteren Containern enthalten Container dann schließlich die eigentlichen Elemente der grafischen Benutzungsschnittstelle, also Textfelder, Druckknöpfe, Listen et cetera. Abbildung 6.2 skizziert die Hierarchie der Elemente, die in JSML derzeit definiert sind.

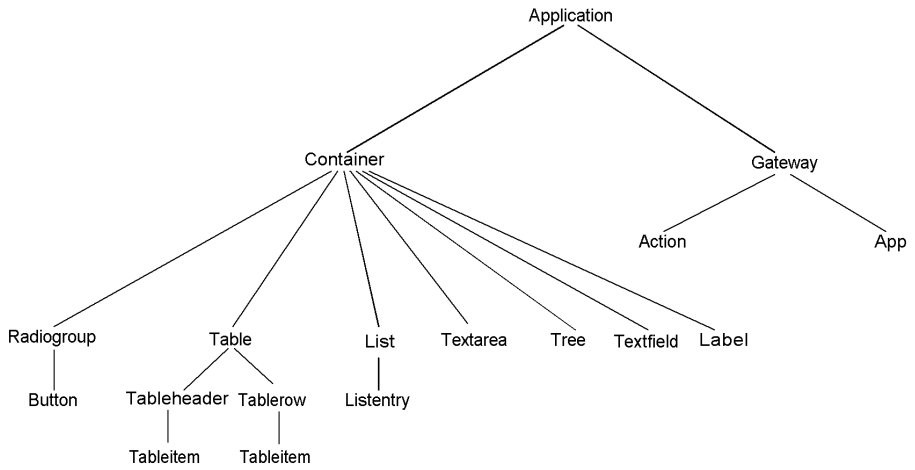


Abbildung 6.2: Die in JSML definierten GUI Elemente

Wie in der Abbildung zu sehen ist, können in JSML die wesentlichen Elemente

einer grafischen Schnittstelle ausgedrückt werden. Die in der Grafik dargestellten Elemente stellen dabei nur die Hauptelemente selbst dar. Über Parameter können die Elemente noch weiter verfeinert werden. So kann bei Buttons zwischen einfachen Druckknöpfen (`Pushbutton`), Teilen einer Gruppe von Knöpfen (`Radiobutton`), Ankreuzkästchen (`Checkbox`) usw. unterschieden werden. Ähnliches gilt auch für die übrigen Elemente.

Die folgenden Abbildungen zeigen eine einfache Benutzungsschnittstelle, deren Entsprechung in JSML Notation sowie die Darstellung nach Transformation in die Zielrepräsentation HTML. In Abbildung 6.3 ist zunächst die grafische Schnittstelle einer einfachen Terminverwaltung abgebildet. In dieser können Informationen zu Terminen erfasst und in einer Tabelle aufgeführt werden. Bereits eingetragene Termine können modifiziert oder gelöscht werden, und die Ansicht kann auf einen bestimmten Zeitraum eingeschränkt werden. Die erfassten Daten werden bei Beendigung der Anwendung persistiert und nach dem Neustart automatisch wieder geladen.

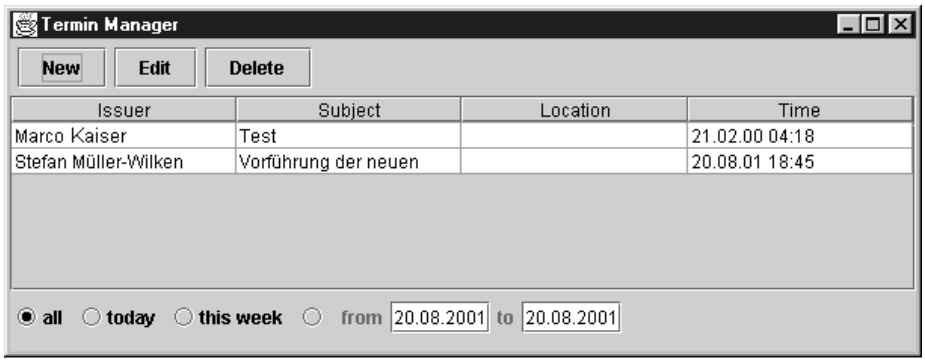


Abbildung 6.3: Eine einfache Terminverwaltung

Die abstrakte Repräsentation der grafischen Schnittstelle, wie sie als Ergebnis einer Analyse durch den Application Shadow erfasst wurde, ist in Abbildung 6.5 dargestellt. Hierin sind die typischen Elemente eines JSML Dokuments wie `Container`, `Button`, `Table` usw. sichtbar. Auch sind zwei weitere Aspekte zu erkennen, auf die im weiteren Verlauf näher eingegangen wird. Zunächst wird jedes Element durch eine eindeutige Kennung (seine ID) identifizierbar gemacht. Es ist diese Kennung, mit deren Hilfe die Integrationsumgebung auf dem Rückkanal Ereignisse an die Anwendung zurückmeldet. Des Weiteren enthält die abstrakte Beschreibung einen Verweis auf den so genannten *Gateway Header*. An Stelle der Variablen `&gwheader` wird vor der Transformation in die Zielrepräsentation ein Schnittstellenelement eingefügt, das dem Benutzer die Steuerung der Sitzung ermöglicht, in welcher die Anwendung zur Verfügung gestellt wird. Abbildung 6.4 zeigt den an dieser Stelle eingefügten JSML Quelltext der Sitzungssteuerung, 6.6 zeigt die nach der Transformation in den oberen Bereich der Darstellung eingeblendeten Schaltknöpfe.

Die Sprache JSML kann in ihrem Umfang jederzeit dahingehend modifiziert werden, neben SWING auch weitere GUI Systeme zu unterstützen. Hierzu muss lediglich die Dokumententyp–Definition (DTD) entsprechend angepasst und der Application Shadow um die Erkennung der zusätzlichen Elemente erweitert wer-

```
<gateway address="http://www.hydeparc.de/hydelets/eca/67970"
         session="2"
         name="de.hydepark.demo.termin.TerminFrame (frame0)">
  <action name="ctrlsubm" value="Submit" />
  <action name="ctrlrefr" value="Refresh" />
  <action name="ctrlsusp" value="Suspend" />
  <action name="ctrlresu" value="Resume" />
  <action name="ctrlquit" value="Quit" />
</gateway>
```

Abbildung 6.4: Befehle der Sitzungssteuerung im Gateway–Header

den. Darüber hinaus ist zusätzlich die Umsetzung der in die Sprache übernommenen GUI Elemente im Zuge der Transformation sicherzustellen, was durch die Erweiterung der hierfür verwendeten Transformationsregelsätze vonstatten geht. Die Programm–Module der Anwendungsumgebung müssen hierfür nicht umgeschrieben werden.

6.2 Realisierung der Abstraktionsschicht

Eine der wesentlichen Eigenschaften des in dieser Arbeit entworfenen Konzepts stellt die Tatsache dar, dass Anwendungen ungeachtet ihres Kontextes von zu ihrem Entwicklungszeitpunkt nicht bekannten Endgeräten zugegriffen werden können und, mehr noch, dass zum Entwurfszeitpunkt der Anwendungen keine Kenntnis darüber bestehen muss, ob diese zu späterem Zeitpunkt überhaupt ferngesteuert werden. Die Realisierung dieser Eigenschaft wird durch zwei Teilkonzepte ermöglicht:

1. Die Extraktion der für die Erstellung des Zwischenformats notwendigen Informationen aus der Anwendung zu deren Laufzeit und ohne deren „Wissen“ durch geeignete Analyseverfahren sowie

2. die in entgegengesetzter Richtung erfolgende Steuerung der Benutzungsschnittstelle der Anwendung, die ebenfalls zur Laufzeit der Anwendung und ohne deren Wissen stattfindet.

Aufgrund ihrer engen Kopplung mit der grafischen Benutzungsschnittstelle der Anwendung wird die Komponente der Infrastruktur, der die genannten Aufgaben obliegen, als *Application Shadow* bezeichnet. Der Application Shadow ist in höchstem Maße auf einen bestimmten Einsatzkontext hin optimiert. Da er auf sehr niedriger Ebene mit der Anwendung und dem zugrunde liegenden Betriebssystem interagieren muss, ist ein generischer Ansatz an dieser Stelle nicht möglich. Da der Application Shadow jedoch über standardisierte Daten– sowie Aufruf–Schnittstellen mit den übrigen Komponenten der Infrastruktur verbunden ist (in der prototypischen Umsetzung primär XML und RMI) und der Austausch über das Netzwerk stattfindet, ist es möglich, für verschiedene Kombinationen von GUI System, Realisierungssprache und Betriebssystem eigene Im-

```
<!DOCTYPE application [<!ENTITY gwheader SYSTEM
"http://www.hydeparc.de/header.jsml">]>

<application name="de.hydepark.demo.termin.TerminFrame"
title="dialog0" scan="6">
  &gwheader;
  <container id="root" title="dialog0">
    <container id="3">
      <container id="5">
        <button id="x233442339" text="New" type="push" />
        <button id="a563213277" text="Edit" type="push" />
        <button id="a976876509" text="Delete" type="push" />
      </container>
      <container id="6">
        <table id="x700320862" rows="2" cols="4" mode="multi">
          <tableheader tableid="x700320862">
            <tableitem>Issuer</tableitem>
            <tableitem>Subject</tableitem>
            <tableitem>Location</tableitem>
            <tableitem>Time</tableitem>
          </tableheader>
          <tablerow id="x7003208620" row="0" tableid="x700320862" checked="false">
            <tableitem>Marco</tableitem>
            <tableitem>Test</tableitem>
            <tableitem></tableitem>
            <tableitem>21.02.00 04:18</tableitem>
          </tablerow>
         <tablerow id="x7003208621" row="1" tableid="x700320862" checked="false">
          <tableitem>Stefan Müller-Wilken</tableitem>
          <tableitem>Vorführung der neuen</tableitem>
          <tableitem></tableitem>
          <tableitem>20.08.01 18:45</tableitem>
        </tablerow>
      </table>
    </container>
    <container id="8">
      <radiogroup id="x674368547">
        <button id="a1039791068" text="all" type="radio" checked="true" />
        <button id="a1066005468" text="today" type="radio" checked="false" />
        <button id="a1060238300" text="this week" type="radio" checked="false" />
        <button id="a563999708" text="" type="radio" checked="false" />
      </radiogroup>
      <label text="from" />
      <textfield id="a635040732" text="20.08.2001" />
      <label text="to" />
      <textfield id="a697168860" text="20.08.2001" />
    </container>
   </container>
  <container id="10" />
  </container>
</application>
```

Abbildung 6.5: Die mittels Introspektion generierte Beschreibung in der Anwendungsschnittstelle

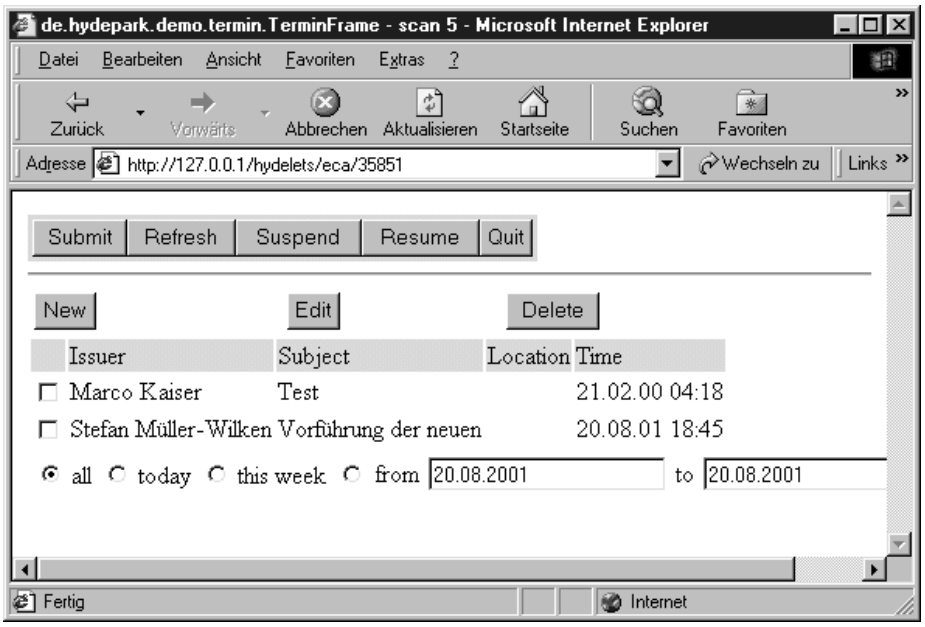


Abbildung 6.6: Darstellung des transformierten Abbildes in einem Webbrowser

plementationen des Application Shadow bereitzustellen und diese unabhängig voneinander an den zentralen Kern der Infrastruktur anzubinden.

Entsprechend der oben genannten zwei Aufgaben des Application Shadows (Analyse und Steuerung) lässt sich die Komponente in zwei Teilmodule aufspalten, die getrennt voneinander beschrieben werden sollen. Die Analyse der Benutzungsschnittstelle wird durch den *Scanner* durchgeführt, die Kontrolle der Schnittstelle erfolgt über den *Updater*. Beiden ist gemeinsam, dass sie direkt in die Datenstrukturen der Benutzungsschnittstelle eingreifen müssen. Der Scanner muss dieses tun, um die Elemente der Schnittstelle erkennen und in eine übereinstimmende allgemeine Repräsentation übersetzen zu können. Der Updater muss die vom entfernten Endgerät ausgehenden Kommandos in Befehle umsetzen, die aus der Sicht der Anwendung wie Aktionen durch Maus und Tastatur wirken; kurz: Er muss die Ereignisbehandlung der Schnittstelle beeinflussen können.

6.2.1 Initialisierung des Application Shadow

In Java läuft eine Anwendung üblicherweise in einer eigenen Laufzeitumgebung, der Java Virtual Machine (JVM) ab. Ein unmittelbarer Zugriff auf die Datenstrukturen der Anwendung mittels Java Reflection API o.ä. ist nur dann möglich, wenn der Zugriff aus derselben JVM erfolgt. Soll der Zugriff aus einer anderen Anwendung, d.h. einer anderen JVM erfolgen, so ist dieses nur über höhere Realisierungsmechanismen der Programmiersprache bzw. des Betriebssystems wie RPC, Pipes, Shared Memory usw. möglich. Da diese Mechanismen eine aktive Rolle der zu steuernden Anwendung voraussetzen, d.h. die Anwendung explizit auf deren Funktionalität hin ausgelegt bzw. hieran angepasst sein muss, sind sie

im Zusammenhang mit den Anforderungen an den Application Shadow unzureichend. Ein wichtiger Realisierungsaspekt des Application Shadows für Java und SWING liegt also zunächst in der Notwendigkeit, in einer gemeinsamen JVM mit der zu kontrollierenden Anwendung zur Ausführung zu kommen.

Unter Java stehen zwei unterschiedliche Ansätze hierfür zur Verfügung. Auf Ebene der JVM selbst kann mit Hilfe eines speziellen Einsprungpunkts eine so genannte *Assistive Technology* an die virtuelle Maschine gekoppelt werden. Wird eine neue JVM Instanz initialisiert, so wird automatisch das als *Assistive Technology* definierte Java Programm in die JVM eingebunden. Primär für die Anbindung von Hilfsmitteln für körperlich eingeschränkte Anwender (z.B. Braille Tastatur, Sprachsteuerung usw.) an die grafische Schnittstelle eines in Java entwickelten Anwendungssystems gedacht, bietet der Ansatz nicht die notwendige Flexibilität, da er nicht selektiv einsetzbar ist. Es ist also nicht möglich, die Assistive Technology für eine Instanz der JVM zu nutzen, während sie für weitere Instanzen ausgeschaltet bleibt. Als Assistive Technology definiert, würde der Application Shadow mit jeder Java Applikation gestartet, also auch für die Komponenten der Infrastruktur selbst. Es ist zudem unmöglich, bestimmte Assistive Technologies gezielt bestimmten Anwendungen zuzuordnen, also gleichzeitig verschiedene Realisierungsansätze zu verfolgen.

Als Alternative bietet sich die Ausnutzung der durchgehenden Verfügbarkeit nebenläufiger Kontrollflüsse innerhalb von Java an. Durch Threads können mehrere Anwendungsteile parallel ablaufen, wobei sie gemeinsamen Zugriff auf die internen Datenstrukturen der Anwendung behalten. Gelingt es nun, die zu untersuchende Anwendung in einem zusätzlichen Thread dem Application Shadow beizuordnen, so kann dieser die Überwachung der GUI Datenstrukturen sowie die Kontrolle der Ereigniswarteschlange parallel und in derselben Java VM durchführen. Um das zu erreichen, ist ein spezielles Initialisierungsverfahren entwickelt worden, das in einer Dreieckskonstellation zwischen JBSA Gateway, Application Factory und Application Shadow den Start einer Anwendung initiiert, wobei letztlich Shadow und Anwendung gemeinsam in einer VM zusammengefasst werden.

Wird über eine laufende Anwendersitzung der Befehl zum Start einer Anwendung erteilt, so generiert das Gateway zunächst eine eindeutige Kennung, welche im weiteren Lauf zur Kennzeichnung der Kombination aus Anwendung und Shadow dient und die als wichtiger Bestandteil der Sitzung verwaltet wird. Gemeinsam mit Angaben über die gewünschte Applikation wird die generierte Kennung in einem Containerobjekt zusammengefasst, dem `FactoryServiceObject` (FSO). Neben der Kennung enthält das FSO den Pfad zu der zu startenden Anwendung, etwaige Startparameter sowie eine textuelle Beschreibung der Anwendung. Diese Daten werden in einem Aufruf der `ApplicationInvoker` Schnittstelle an die Application Factory weitergegeben und die Factory sodann beauftragt, den Anwendungsstart einzuleiten. Die Application Factory startet einen zunächst leeren Application Shadow und übergibt diesem genaue Angaben zu der zu startenden Anwendung. Diese umfassen neben dem Pfad zur gewählten Anwendungsklasse und deren vollständigen Namen wiederum die eindeutige Kennung. Mit Hilfe der VM wird die Anwendungsklasse in den Speicher geladen, deren Startmethode ermittelt, ein eigener Thread generiert und die Startmethode dann in dem neuen Thread zur Ausführung gebracht.

In einem letzten Schritt muss der Application Shadow nun in die Anwendersitzung integriert werden. Der Shadow ruft dazu eine Registrierungsmethode der `Application Dock` Schnittstelle des JBSA Gateways auf, wobei er eine Referenz auf seine eigene Aufrufschnittstelle übergibt und zusätzlich die zu Beginn generierte Kennung aus dem `FactoryServiceObject`. Anhand dieser Kennung kann das JBSA Gateway die passende Sitzung identifizieren und in ihr die übergebene Referenz ablegen. Der Prozess des Bootstrappings ist damit abgeschlossen: Die Anwendung residiert gemeinsam mit dem Application Shadow in einer einzigen virtuellen Maschine, welche direkt auf deren Datenstrukturen und Ereignisbehandlung zugreifen kann. Der Shadow ist an das JBSA Gateway gekoppelt und eindeutig einer bestimmten Sitzung zugeordnet, und Anfragen eines bestimmten Endgeräts können durch das JBSA Gateway direkt an einen bestimmten Application Shadow weitergeleitet werden, wobei das Gateway einfach per RMI auf den zur Sitzung gehörigen Shadow zugreifen kann.

6.2.2 Introspektion mittels Scanner

Bekommt der laufende Application Shadow vom Gateway über seine `Application` Schnittstelle den Befehl zur Übertragung des aktuellen Zustands der grafischen Anwendungsschnittstelle, so wird der Scanner als Teilkomponente zur Überwachung des GUIs aktiv. Der Scanner greift hierzu nicht direkt auf die hierarchisch miteinander verketteten Datenstrukturen zu, die der grafischen Benutzungsschnittstelle zugrunde liegen. Aus Gründen der Laufzeiteffizienz ist eine wiederholte vollständige Analyse der Schnittstellenstruktur nicht sinnvoll. Vielmehr wird bei der Initialisierung des Application Shadows eine erste Analyse der grafischen Schnittstelle durchgeführt, in deren Verlauf jeder Komponente der Schnittstelle eine Verwaltungskomponente im Application Shadow beigeordnet wird, die ausschließlich zur Überwachung der GUI Komponente dient. Die Verwaltungskomponente registriert sich für den Empfang von Ereignissen, die vom korrespondierenden Element der Schnittstelle stammen und kann so auch außerhalb nachfolgender Aktualisierungsläufe seinen Zustand anpassen. Ausgehend von einer Analyse des `TopLevelWindow` der grafischen Schnittstelle entsteht rekursiv während dieses ersten Schritts eine zweite Hierarchie, die weitgehend mit der tatsächlich sichtbaren Schnittstelle übereinstimmt. Hierzu sei auf Abbildung 6.7 verwiesen: Der unter a) schematisch dargestellte Dialog, bestehend aus drei Auswahlknöpfen, wird durch SWING in der unter b) verbildlichten Weise strukturiert. Neben den sichtbaren GUI Elementen besteht er auch aus nicht sichtbaren, strukturgebenden Anteilen, den so genannten Layout Elementen.

Das Schema unter c) zeigt schließlich die interne Organisation des GUIs und der korrespondierenden Schattenhierarchie. Jedes Objekt der Schattenhierarchie besitzt einen Verweis auf das überwachte GUI Objekt sowie auf die Schattenobjekte derjenigen GUI Objekte, die dem überwachten Objekt untergeordnet sind. Es verfügt lediglich über eine sehr kleine Aufrufschnittstelle, als deren wichtigste Methode `ToJSML()` zu nennen ist. `ToJSML()` erkundet mittels Java Reflection API den Zustand des überwachten GUI Objekts und generiert daraus ein kurzes Code–Fragment in der Beschreibungssprache JSML. Es ruft sodann die `ToJSML` Methoden der untergeordneten GUI Elemente auf und fügt die von dort empfangenen JSML Blöcke in das zuvor generierte JSML Fragment ein,

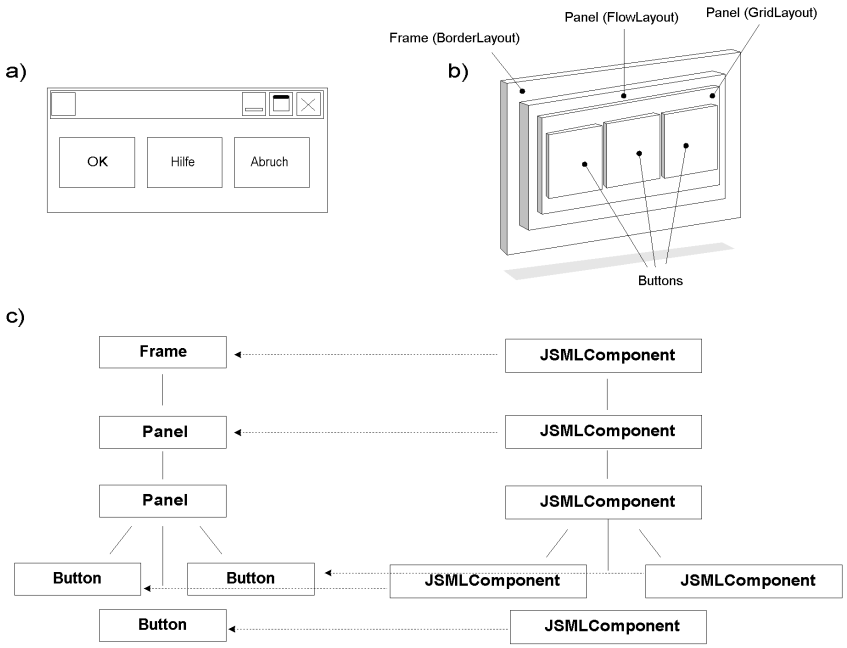


Abbildung 6.7: GUI mit beigeordneter Hierarchie von Überwachungsobjekten

bevor es dieses schließlich an den Aufrufer zurückgibt. Auf diese Weise entsteht nach Traversierung der Schattenhierarchie eine vollständige Zustandsbeschreibung der Schnittstelle in der Beschreibungssprache JSML. Abbildung 6.7 zeigt im Überblick den mittels `ToJSML` aus der GUI Hierarchie erzeugten Code in der Sprache JSML. Ineinander verschachtelt werden sämtliche Bestandteile der grafischen Schnittstelle dargestellt und in ihren Eigenschaften genauer beschrieben.

Gleichzeitig mit dem Aufbau der Schattenhierarchie registriert sich der Scanner für den Empfang der wesentlichen Ereignisse, die von SWING im Zusammenhang mit der Anzeige von Schnittstellen generiert werden. Dieses umfasst neben Meldungen über den Wegfall oder das Hinzukommen von GUI Elementen auch Nachrichten über Änderungen der angezeigten Elemente. Insbesondere letztere sind im Zusammenhang mit der Unterstützung proaktiver Analyseverfahren

```
<CONTAINER ...> (Frame / BorderLayout)
  <CONTAINER ...> (Panel / FlowLayout)
    <CONTAINER ...> (Panel / GridLayout)
      <BUTTON...> (Auswahlknopf)
      <BUTTON...> (Auswahlknopf)
      <BUTTON...> (Auswahlknopf)
    </CONTAINER>
  </CONTAINER>
</CONTAINER>
```

Abbildung 6.8: Aus der in Abbildung 6.7 mittels `ToJSML` erzeugte Beschreibung

notwendig. Dabei wird die Analyse nicht durch den Benutzer angestoßen, sondern durch Änderungen in der Schnittstelle. Der Benutzer kann durch den so initiierten „Push“ ohne sein Zutun über Aktivitäten der benutzten Anwendung informiert werden. Hierzu später mehr.

6.2.3 Behandlung von Layout Aspekten

Neben den eigentlichen GUI Elementen stellt die Behandlung der strukturgebenden Aspekte der grafischen Schnittstelle einen sehr wichtigen Teilkomplex im Zuge der Analysephase dar. So spielt für den Wunsch nach Berücksichtigung der Anordnung von GUI Elementen nicht nur die Ästhetik der Darstellung eine Rolle, oftmals werden durch die Positionierung der Elemente semantische Informationen übertragen. So wird durch die Anordnung von GUI Elementen zum Teil deren Zusammengehörigkeit dargestellt.

Dabei ist auf konzeptioneller Seite zu beachten, dass nicht alle unterstützten Endgeräte über dieselbe Ausdrucksmächtigkeit verfügen. Schnittstellen, die sich auf dem Bildschirm eines bestimmten Geräts abbilden lassen, können für ein anderes bereits aufgrund ihrer Größe nicht darstellbar sein. Eine detailgetreue Reproduktion der zuvor analysierten Schnittstelle ist nur in sehr wenigen Fällen überhaupt realisierbar. Während die Analyse eine maximale Menge an Informationen über das Layout der Schnittstelle ableiten sollte, ist also gleichzeitig die schnelle und möglichst akkurate Reduktion der entsprechenden Informationen zu ermöglichen, sofern die Beschränktheit des Endgerätes ein solches erfordert. Die extrahierte Information muss klar genug strukturiert sein, um eine effiziente Umsetzung möglich zu machen.

Zusätzlich ist ein rein technischer Gesichtspunkt zu betrachten. Im Zusammenhang mit der Analyse von SWING GUIs stellt sich als problematisch heraus, dass Java auf der Verwendung von so genannten *Strategy Patterns* [GHJV95] beruht. Das verwendete Layout ergibt sich dabei nicht direkt aus der geeigneten Parametrisierung übergeordneter (Container–) GUI Elemente wie Frames oder Panels. Vielmehr wird das Layout durch zusätzliche, nicht sichtbare GUI Elemente bestimmt, die so genannten *Layout Manager*. Layout Manager können einem Container über Referenzen beigeordnet werden und sodann die Anordnung der diesem untergeordneten GUI Elemente bestimmen. Das Layout ist somit keine *direkte* Eigenschaft des übergeordneten GUI Elements, sondern haftet ihm nur indirekt an. Hieraus ergibt sich ein technisches Realisierungsproblem, da Java zwar die Analyse von Objekten ermöglicht, nicht ohne weiteres jedoch die transitive Analyse beigeordneter Elemente, sofern dieses nicht explizit durch deren Entwickler gewünscht war. Während die Traversierung der GUI Hierarchie durch entsprechende Teile der SWING API unterstützt wird, ist die Analyse assoziierter Layout Manager somit nur bedingt möglich.

Der geschilderten Problematik wird in der prototypischen Umsetzung begegnet, indem die Analyse der Schnittstelle nicht bis auf die Ebene der Layout Manager hinuntergebrochen wird. Stattdessen wird auf der Basis von relativen Positionen der GUI Elemente zueinander angesetzt. Der entwickelte Reduktionsalgorithmus geht von den tatsächlichen Pixel–Koordinaten der GUI Elemente innerhalb des Anwendungsfensters aus. Er vereinfacht diese so lange, bis sich schließlich eine Tabelle ergibt, in deren Zellen sich jeweils identifizierte GUI Elemente befinden

und die in der Anzahl ihrer Zeilen und Spalten minimal ist. Diese Tabelle wird sodann in JSML kodiert und zur weiteren Verarbeitung an die Transformationseinheit übergeben. Über die Breite und Höhe der Zellen lässt sich das ursprüngliche Aussehen der grafischen Schnittstelle recht genau annähern. In Abbildung 6.9 ist der Prozess grob skizziert. Nachdem die Koordinaten der einzelnen GUI Elemente erfasst sind, wird über die so erhaltene Anordnung ein virtuelles Raster gelegt. In mehreren Schritten werden sodann leere Zeilen und Spalten entfernt. Des Weiteren wird die Anordnung normalisiert, so dass schließlich ein minimales Layout verbleibt, das sich einfach transformieren lässt. Abbildung 6.9/d) zeigt das Ergebnis einer solchen Transformation. Das ursprüngliche Layout wird im HTML Browser mittels HTML Tabellen simuliert. Wie deutlich wird, bleibt die ursprüngliche Anordnung der GUI Elemente auch nach der Reduktion noch weitgehend erhalten.

Andererseits ist durch die Vergröberung der Informationen über das Schnittstellenlayout eine sehr effiziente Umsetzung auf stark reduzierte Zielplattformen möglich. Schließlich wird durch den Verzicht auf die nähere Untersuchung des verwendeten Layout Managers die oben angedeutete Schwierigkeit im Zugriff auf lediglich indirekt zugängliche Datenstrukturen umgangen.

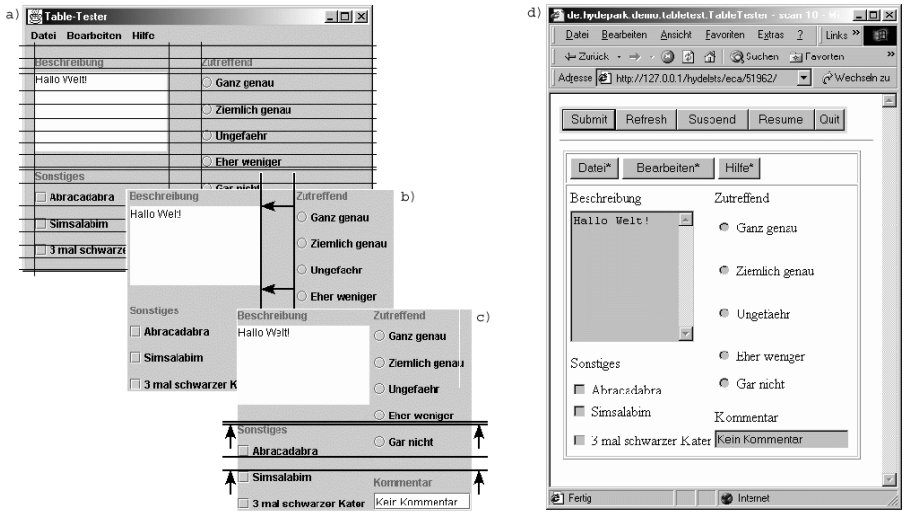


Abbildung 6.9: Vor seiner Transformation (rechts) wird das Layout der Schnittstelle in mehreren Schritten reduziert (links)

In Untersuchungen hat sich gezeigt, dass sich aus diesem Vorgehen keine qualitativen Nachteile ergeben, da Layout Manager vornehmlich bei der Anordnung der GUI Elemente während des Anwendungsstarts und bei der späteren Änderung der Fenstergröße eine Rolle spielen. Beides fällt jedoch im vorliegenden Anwendungsfall nicht ins Gewicht. Die grafische Schnittstelle wird auf dem mobilen Endgerät lediglich dargestellt, Änderungen in Größe und Position werden hingegen ausschließlich durch die internen Kontrollmechanismen der Anwendung selbst bzw. das zugrunde liegende GUI-System gesteuert. Die bloße Erfassung der relativen Position von GUI Elementen ist also vollkommen ausreichend für den Einsatz im vorliegenden Anwendungsfall.

```
<xsl:template match="application">
    <html>
    <head>
        <title>
            <xsl:value-of select="@name"/>
            <xsl:text> - scan </xsl:text>
            <xsl:value-of select="@scan"/>
        </title>
    </head>
    <body bgcolor="#ffffff">
    <form method="POST">

        <xsl:attribute name="action">
            <xsl:value-of select="gateway/@address"/>
        </xsl:attribute>

        <xsl:apply-templates select="menubar"/>
        <xsl:apply-templates/>
    </form>
    </body>
    </html>
</xsl:template>
```

Abbildung 6.10: Regel zur Umsetzung des `Application` Tags von JSML nach HTML

6.3 Transformation von JSML Dokumenten

Nachdem der Application Shadow eine Momentaufnahme der grafischen Schnittstelle erstellt hat, muss diese in eine konkrete Zielrepräsentation überführt werden. Die hierzu notwendige Transformation obliegt einem der vom JBSA Gateway gesteuerten Transformatoren, der diese Aufgabe unter Verwendung eines der verfügbaren Transformationsregelsätze (Stylesheets) versieht.

Im Rahmen der Umsetzung wird jedem Element in der Zwischenrepräsentationssprache eine Menge von Elementen in der Zielrepräsentation zugeordnet. Entsprechende Regeln des verwendeten Regelsatzes sorgen für eine Umsetzung von Elementen der Ausgangssprache in die Zielsprache. Die Umsetzung kann ihrerseits wiederum Elemente der Ausgangsrepräsentation enthalten, so dass sich durch wiederholte Anwendung der Regeln so die endgültige Zielrepräsentation ergibt. Abbildung 6.10 zeigt einen Ausschnitt aus dem Regelsatz zur Transformation von JSML nach HTML.

Aus der Abbildung ist ersichtlich, dass die Übersetzungsregel aus einer Kombination von Elementen der Zielsprache sowie der Anwendung weiterer Regeln besteht. So wird als äußerer Rahmen der Zielrepräsentation in HTML ein `Head` Tag erzeugt, in dem wiederum einige Statusinformationen als Titelzeile eingefügt werden. Wie aus Abbildung 6.5 zu entnehmen ist, stammen die Variablen `name` sowie `scan` aus der Parameterliste des `Application` Tags. Im hierauf folgenden `Body` Tag wird dann zunächst die spätere Rückübertragung von Daten an den

ECA vorbereitet. Hierzu wird ein so genanntes HTML Form Element geöffnet, dem als Rücksprungadresse die Belegung der Variablen `gateway` sowie `address` übergeben wird. Im Anschluss daran werden dann die weiteren Transformationsregeln des Stylesheets abgearbeitet, wobei zunächst die Behandlung von Menüzeilen erfolgt, danach dann die Anwendung aller übrigen Regeln.

Die entstehende Abbildungsfunktion ist nicht zwangsläufig bijektiv: Weist die Zielplattform eine geringere bzw. grundlegend andere Ausdrucksmächtigkeit als die Ausgangsplattform auf, so ist im Zuge der Transformation u.U. eine Umsetzung bestimmter Elemente in Elemente eines gänzlich unterschiedlichen Typs notwendig. Als Beispiel hierfür mag die Umsetzung einer Tabelle herangezogen werden, wie sie in Abbildung 6.3 dargestellt ist. Während die Umsetzung in HTML relativ einfach vonstatten geht, taucht bei Umsetzung nach WML das Problem auf, dass eine horizontale Verschiebung des Sichtfelds auf WAP Telefonen üblicherweise nicht möglich ist. Entsprechend müssen die Tabellenzeilen in eine vertikale Form umgesetzt werden, wobei jede Zeile in eine eigene, einspaltige Tabelle eingefügt wird.

6.4 Der HTTP–ECA

Sämtliche Zugriffe auf die Integrationsumgebung und die durch sie hindurch zur Verfügung gestellten Anwendungen finden durch die angeschlossenen externen Kommunikationsadapter statt. Jeder ECA bildet die spezifischen Merkmale des jeweils zu unterstützenden Kommunikationskanals ab. Die Unterstützung gesprochener Sprache wird von einem anderen ECA erbracht als die Unterstützung des Zugriffs mittels HTTP, die Unterstützung herkömmlicher Webbrowser von einem anderen ECA als die Unterstützung von i–Mode Telefonen. Andererseits lässt sich auch die Unterstützung mehrerer Kommunikationskanäle zusammenfassen. So wird für den Zugriff mittels Webbrowsern derselbe ECA genutzt wie für den Zugriff mittels WAP Telefonen, da beide letztlich das HTTP Protokoll verwenden und lediglich der darüber vermittelte Inhalt sich unterscheidet. Schließlich können auch mehrere ECAs für einen Kommunikationskanal zur Verfügung stehen, beispielsweise um unterschiedliche Merkmale dieses Kanals zu betonen. So sind zur Unterstützung des HTTP–basierten Kommunikationskanals gleich mehrere Varianten denkbar, so u.A. für die Segmentierung von Daten und für die Berücksichtigung des WAP Push Protokolls.

Anhand von Details zur Unterstützung des HTTP Kanals soll nun die Funktionsweise eines ECAs verdeutlicht werden. Der im Folgenden als HTTP–ECA bezeichnete Adapter dient als Kommunikationsendpunkt für alle Endgeräte, die den Datenaustausch auf Basis des HTTP–Protokolls realisieren. Dieses sind primär alle Endgeräte (PCs, Handhelds usw.) mit einem herkömmlichen Webbrowser. Aber auch für andere Ansätze, die unmittelbar auf HTTP basieren (so z.B. WAP Telefone) kann auf den HTTP–ECA zurückgegriffen werden.

Prinzipiell ist es die Aufgabe eines ECAs, Aufrufe vom Endgerät entgegenzunehmen, die im Aufruf enthaltenen Informationen zu extrahieren und als gültige Befehle an die funktionalen Komponenten der Infrastruktur weiterzuleiten. Im Fall des HTTP–ECA wurde die notwendige Applikationslogik in Form eines java–basierten Servlets direkt in den Webserver integriert. Das Servlet regist-

riert sich nach dem Start des Webservers bei der Integrationsumgebung und kann fortan Daten via RMI austauschen. In Richtung des JBSA Gateways findet der Datenaustausch über die `ECADock` Schnittstelle statt. Über diese Schnittstelle kann der ECA nach dem Start seine eigene Aufrufschnittstelle beim Gateway registrieren und von diesem Inhalte abfragen. Die `ECA` Schnittstelle des Servlets dient dem JBSA Gateway in entgegengesetzter Richtung dazu, transformierte Informationen zur Weiterleitung an das Endgerät an den ECA zu übergeben.

6.4.1 Geräte– und Anwendungssitzung im HTTP–ECA

Unter Abschnitt 4.3.8 wurde erläutert, dass sich im Zusammenhang mit der Unterstützung von Endgeräten drei unterschiedliche Sitzungstypen identifizieren lassen, nämlich Gerätesitzung, Anwendungssitzung sowie Benutzersitzung. Der ECA ist dabei primär für die Verwaltung von Gerätesitzungen zuständig, wobei er dem Gateway die notwendigen Informationen zur Realisierung der beiden übrigen Sitzungstypen zur Verfügung stellen muss.

Bei HTTP handelt es sich um ein prinzipiell zustandsloses Protokoll. Entsprechend muss jede Sitzungssemantik durch die Anwendung selbst etabliert werden, die den Datenaustausch mittels HTTP realisiert. Im Falle des HTTP–ECAs kommt hierbei die Servlet–API zum Einsatz: Für jede Verbindung, die von einem entfernten Endgerät aufgebaut wird, kommt es zur Generierung einer eigenen Instanz des HTTP–ECAs, der eine eindeutige Kennung vom Webserver übergeben wird. Diese Kennung fungiert innerhalb der Integrationsumgebung als Identifikation der Gerätesitzung, die für die gesamte Dauer der Kommunikationsverbindung bestehen bleibt. Die Sitzungskennung wird mit jeder Anforderung von Inhalten an das Gateway übergeben, wobei die Kennung als Variable der in der Anfrage übergebenen `UpdateRequest` Struktur kodiert ist. Anhand der Kennung kann das Gateway die zugehörige Anwendungssitzung identifizieren und ihr die Anfrage zuordnen, wobei beim ersten Zugriff von einer ECA–Instanz zunächst eine Anwendungssitzung erstellt wird. Dieser wird die Sitzungskennung der Gerätesitzung beigeordnet, so dass die Verzahnung von Gerätesitzung und Anwendungssitzung fortan gewährleistet ist.

In entgegengesetzter Richtung wird ähnlich verfahren. Transformierte Inhalte, die vom ECA an das Endgerät weitergeleitet werden sollen, werden gemeinsam mit der kodierten Gerätekennung in einem Aufruf der `ECA` Schnittstelle übergeben. Der ECA kann die verantwortliche Gerätesitzung erkennen und damit das Gerät bestimmen, an welches der Inhalt weitergeleitet werden soll. Die Kennung wird dabei zusätzlich in den Inhalt selbst integriert, so dass er in einer etwaigen Antwort des Gerätes wieder an den ECA zurückgesendet wird. Selbst für den Fall, dass die Verbindung mit dem Endgerät kurzzeitig abreißt, kann so über die beim nächsten Zugriff des Geräts eingehende Kennung die Gerätesitzung wiedererkannt werden.

6.4.2 Ermittlung von Gerätetypen im HTTP–ECA

Das genaue Wissen über Gerätemerkmale spielt eine sehr wichtige Rolle bei der optimalen Unterstützung von Endgeräten. Auf mögliche Konzepte wie die

```
Cache-Control: no-cache
Connection: keep-alive
Date: Mon, 27 Aug 2001 14:39:27 GMT
Pragma: no-cache
Accept: text/vnd.wap.wml,text/vnd.wap.wmlscript,[...]
Accept-Charset: UTF-8, ISO-8859-1, ISO-10646-UCS-2
Host: wap.hydeparc.de:8080
User-Agent: Nokia-WAP-Toolkit/2.0
```

Abbildung 6.11: Vom Endgerät übermittelte Geräteinformationen am Beispiel eines emulierten WAP–Telefons

Nutzung von *Header Fields* bzw. *Capability Profiles* wurde in Kapitel 4 im Detail eingegangen.

In der prototypischen Realisierung obliegt die Entgegennahme und erste Strukturierung der vom Gerät ausgesandten Informationen dem ECA. Da die zuvor eingeführten Ansätze wie RDF und CC/PP derzeit noch nicht hinreichend durch vorhandene Geräte, Browser und Serversysteme unterstützt werden, stützt sich der HTTP–ECA primär auf die im HTTP Protokoll weit verbreiteten Header Fields und hierbei insbesondere auf die `Accept--` sowie `User--Agent` Attribute, welche aus den vom Endgerät übertragenen HTTP Protokollnachrichten extrahiert werden können. Abbildung 6.11 zeigt ein Beispiel für eine solche Protokollnachricht und die in ihr enthaltenen Felder. Wie zu erkennen ist, akzeptiert das Endgerät WML sowie WMLScript und daneben eine Reihe verschiedener Zeichensätze. Als User Agent wird das Nokia WAP–Toolkit angegeben, es handelt sich beim Sender der Nachricht also um ein emuliertes WAP Telefon.

Der HTTP–ECA normiert die empfangenen Informationen über die Gegenstelle anhand eines internen Klassifikationsschemas und vereinfacht sie zu einer zweidimensionalen Matrix, basierend auf einem Haupttyp (Mimetype) und einem Untertyp (Clienttype). Mit ihrer Hilfe wird eine Kennung ermittelt, die den optimalen Transformator für das Endgerät identifiziert. Aufgrund der ermittelten Identifikation kann das Gateway sodann den verantwortlichen Transformator starten und diesen zur weiteren Unterstützung des ECA bereitstellen. Während über den Haupttyp jederzeit ein minimal gültiger Transformator ausgewählt werden kann, lassen sich durch Berücksichtigung des Untertyps zusätzliche Verfeinerungen hinsichtlich des verwendeten Regelsatzes realisieren.

Von besonderer Wichtigkeit sind die Geräteinformationen und die aus ihnen extrahierte Gerätekennung beim Erstkontakt des Endgerätes, da zu diesem Zeitpunkt der für das Endgerät zuständige Transformator ausgewählt werden muss. Darüber hinaus wird die Kennung mit jeder Anfrage erneut übermittelt, so dass auch im Verlauf der Interaktion auftretende Änderungen der Anforderungen des Endgeräts berücksichtigt werden können. Werden Geräteparameter umgestellt, so kann demgemäß auch nachträglich noch der zuständige Transformator ausgewechselt und so die Generierung der Zielrepräsentation an die neuen Bedingungen angepasst werden.

6.4.3 Kaskaden zur Behandlung spezifischer Merkmale

Wie bereits angedeutet, können spezifische Merkmale eines Kommunikationskanals durch die Etablierung von Varianten des zuständigen ECAs behandelt werden. Im Fall des HTTP–ECAs wurde dieses anhand der Behandlung des so genannten Segmentierungsproblems nachgewiesen. Das Segmentierungsproblem tritt bei der Darstellung von WML Seiten auf WAP Telefonen auf. Deren lokaler Puffer ist zumeist derartig begrenzt, dass bereits kleine Seiten von einigen hundert Zeichen zu einem Pufferüberlauf führen können und eine Darstellung der übertragenen Seite unmöglich wird. Die darzustellenden Inhalte müssen demnach auf mehrere Seiten verteilt werden, bevor sie nacheinander zum Endgerät übertragen und dort angezeigt werden können. Da WAP Browser jedoch nur korrekt geklammerte WML Dokumente darstellen können, in denen jedem öffnenden WML–Tag ein korrespondierendes schließendes WML–Tag zugeordnet ist, stellt die bloße Zerlegung der darzustellenden Inhalte in genügend kleine Segmente keine Lösung dar. Es ist vielmehr notwendig, das WML Dokument in Segmente zu zerlegen und diese vor der Weiterleitung zum Endgerät um die syntaktisch notwendigen schließenden Tags sowie die Möglichkeit zur Navigation zwischen den einzelnen Segmenten zu ergänzen. Abbildung 6.12 zeigt die in eine segmentierte Seite eingefügten Befehlsoptionen zum Segmentwechsel.



Abbildung 6.12: Möglichkeit zum Segmentwechsel im Push ECA

In der prototypischen Realisierung der JBSA wird die entsprechende Funktion durch den segmentierenden HTTP–ECA erfüllt. Der segmentierende HTTP–ECA entscheidet anhand des Gerätetyps zunächst, ob eine Zerlegung der zu übertragenden Inhalte notwendig ist. Ist dieses der Fall, so leitet er die Anfrage an den herkömmlichen HTTP–ECA weiter, von dem er den transformierten Inhalt übernimmt. Bevor dieser zum Endgerät verschickt wird, erfolgt nun die Zerlegung in Fragmente und deren jeweilige Ergänzung um korrespondierende schließende Tags zu allen geöffneten Tags. Die entstehenden Seiten werden in einem lokalen Zwischenspeicher des segmentierenden ECA abgelegt. Das erste Segment wird sodann an das Endgerät übertragen, wobei neben der Sitzungskennung eine Kennung des ausgesandten Segments verschickt wird. Die Segmentnummer wird wieder an den ECA übertragen, sofern der Benutzer den Befehl zur Anzeige des nächsten Segments erteilt. In diesem Falle wird das angeforderte Segment aus dem lokalen Zwischenspeicher des segmentierenden ECA entnommen, ohne dass hierfür eine Aktualisierung der Schnittstelleninformation beim HTTP–ECA angefordert werden würde. Wird andererseits ein Element der Anwendungsschnittstelle bedient oder explizit die Aktualisierung der Schnittstelle angefordert, so leert der segmentierende ECA seinen Puffer und leitet die

Anfrage wiederum an den HTTP–ECA weiter.

In ähnlicher Weise wie für das Beispiel des Segmentierungsproblems lassen sich theoretisch beliebig viele ECAs zur Behandlung von Aspekten eines gemeinsamen Kommunikationskanals miteinander kaskadieren. So basiert der ECA zur Unterstützung des WAP Push–Protokolls seinerseits auf dem segmentierenden ECA, ein ECA zur Unterstützung von VoxML könnte wiederum auf dem Push ECA aufsetzen usw.

6.5 Sitzungssteuerung

Die Sitzungssteuerung stellt ein zentrales Element der Integrationsumgebung dar. Beginnend mit dem Eintreffen einer Anfrage vom ECA werden mit ihrer Hilfe die verschiedenen Teilschritte der Generierung von Inhalten, deren Transformation und schließlich deren Rückmeldung an den ECA koordiniert und miteinander synchronisiert.

Die Sitzungssteuerung wird im Wesentlichen von zwei Modulen der Integrationsumgebung wahrgenommen. Der in Abbildung 6.1 sichtbare *Basic Session Manager* ist für die Verwaltung der zu einer laufenden Anwendungssitzung gehörigen Steuerungsdaten zuständig, die jeweils pro Sitzung in einem Sitzungsobjekt zusammengefasst sind. Das Sitzungsobjekt speichert neben der eindeutigen Kennung der Anwendungssitzung die Referenzen auf innerhalb der Anwendungssitzung laufende Application Shadows sowie den gerade aktiven Shadow, eine Referenz auf den derzeit der Sitzung zugeordneten ECA, den derzeit aktuellen Gerätetyp sowie weitere Informationen, die zur Unterstützung des Endgerätes in der Sitzung notwendig sind. Auf der Basis der im Sitzungsobjekt gespeicherten Daten werden also Gerätesitzung und Anwendungssitzung miteinander verkettet.

Die eigentliche Steuerung der aktiven Sitzungen wird durch das JBSA Gateway geleistet. Dieses nimmt über seine `ECADock` Schnittstelle die Anfragen von den angeschlossenen ECAs entgegen, startet für diese jeweils einen nebenläufigen Kontrollfluss und initiiert bzw. koordiniert die diversen Teilschritte, die auf dem Weg von der Zugangskontrolle über die Weiterleitung der Anfrage an den zuständigen Application Shadow bis hin zur Transformierung der JSML Dokumente durchlaufen werden müssen. Daneben ist es Aufgabe des JBSA Gateways, neue Sitzungen einzurichten, den Bootstrapping Prozess der zu steuernden Anwendung und des mit ihr verbundenen Application Shadows einzuleiten und diesen in das initialisierte Sitzungsobjekt einzutragen.

6.5.1 Phasensteuerung durch das JBSA Gateway

Die Interaktion von Endgerät und Anwendung mit der zwischengeschalteten Integrationsumgebung wird anhand des Informationsflusses in Folgen von Anfragen (*Update Requests*), jeweils gefolgt von zugehörigen Antworten (*Update Responses*), zerlegt. Request und Response bilden dabei zusammen einen Schritt der Sitzungssteuerung. Jeder Schritt der Sitzungssteuerung wird wiederum in sieben Teilschritte zerlegt. Dieses sind im Einzelnen:

1. **Initialisierungsphase** : Im ersten Teilschritt wird der vom ECA eingehende Request aufgenommen und hinsichtlich seiner grundsätzlichen Gültigkeit untersucht.
2. **Zugangssteuerungsphase** : In diesem Teilschritt wird festgestellt, ob der Request einer laufenden Sitzung zugeordnet werden kann oder ob eine neue Sitzung generiert werden muss.
3. **Quellenauswahlphase** : Im nächsten Teilschritt wird dem Request eine Datenquelle zugeordnet. Im Fall einer laufenden Sitzung ist dies der zugehörige Application Shadow der aktiven Anwendung. Bei Initialisierung einer Sitzung wird hingegen der User Manager als Quelle gewählt, um einen Anmeldeschirm darstellen zu lassen.
4. **Informationsgewinnungsphase** : Nach Festlegung der Datenquelle wird in diesem Teilschritt die Bereitstellung von Informationen eingeleitet.
5. **Transformationsphase** : Vor Weiterleitung an das Endgerät müssen die generierten Daten in das gewünschte Zielformat transformiert werden. Die Transformation findet in dieser Phase statt.
6. **Sendephase** : Im Zuge der Sendephase werden die transformierten Daten an den zuständigen ECA zurückübertragen.
7. **Aufräumungsphase** : Den Abschluss bildet dieser Teilschritt, in dem eventuell zuvor belegte Ressourcen freigegeben werden können.

Der für eine Anwendungssitzung zur Zeit aktuelle Teilschritt wird im Sitzungsobjekt vermerkt und jeweils nach Abschluss der Phase fortgeführt. Jede Anfrage befindet sich demnach von ihrem Eintreffen an jederzeit in einer der sieben Phasen, die sie sequentiell durchläuft, bevor das Ergebnis an den ECA zurückgegeben wird.

Geht eine Anfrage vom ECA ein, so überprüft das Gateway zunächst anhand der übergebenen Kennung der Gerätesitzung, ob hierzu eine Anwendungssitzung vorliegt. Mit Ausnahme des Erstzugriffs, bei dem eine Sitzung erst eingerichtet wird, sollte dieses immer der Fall sein. Das Gateway initialisiert für den Request ein `JbsaRequest`–Containerobjekt, überträgt alle Daten der eingegangenen Anfrage in das Request Objekt und initialisiert dessen Phasenzähler. Es erstellt dann einen nebenläufigen `Request Processor`, welchem es das Request Objekt überträgt und der fortan die Kontrolle der Anfrage übernimmt.

Durch die Verwendung von `Request Processor` Objekten wird die Effizienz des JBSA Gateways deutlich gesteigert. Mehrere Aufrufe können parallel voneinander abgearbeitet werden, ohne dass diese einander blockieren. Das Gateway selbst ist nur für die Zeit der Initialisierung eines Prozessors gebunden und steht nach dessen Start für die weitere Kommunikation zur Verfügung.

6.5.2 Abarbeitung einer Anfrage

Der `Request Processor` leitet die Abarbeitung mit der initialen Phase ein. Hierin wird zunächst anhand der im Request abgelegten Kennung der Gerätesitzung

die verantwortliche Anwendungssitzung identifiziert und auch deren Kennung in den Request übernommen. Alle weiteren Schritte sind nur dann möglich, wenn eine gültige Anwendungssitzung erkannt werden konnte.

In der folgenden Zugangssteuerungsphase wird dann überprüft, ob für die ermittelte Anwendungssitzung eine gültige Benutzerkennung vorliegt. Wurde die Sitzung zuvor erst erstellt, ist dieses nicht der Fall. Anstelle einer Anwendungsschnittstelle muss also nachfolgend zunächst eine Anmeldemaske generiert, transformiert und dargestellt werden. Diese Tatsache wird gegebenenfalls im Request Objekt vermerkt.

Nach Durchlaufen der Zugangskontrolle wird nunmehr der Generator ermittelt, der für die Lieferung des auf die Anfrage zurückzumeldenden Inhalts verantwortlich ist. Konnte zuvor keine gültige Benutzerkennung erkannt werden, so ist es die Integrationsumgebung selbst, die den JSML Code einer Anmeldemaske zu erzeugen hat. Auch kann unter bestimmten Bedingungen die Sitzungssteuerung verantwortlich sein, beispielsweise, wenn der Anwender innerhalb der aktuellen Anwendungssitzung einen Wechsel der aktiven Applikation wünscht. In den meisten Fällen wird jedoch der mit der aktiven Anwendung verbundene Application Shadow diese Rolle übernehmen. Als Resultat dieser Phase wird der `JbsaRequest` um die ermittelte Referenz des verantwortlichen Lieferanten erweitert.

In der nächsten Phase erfolgt dann die eigentliche Erzeugung von darzustellenden Inhalten. Dazu wird dem zuvor ermittelten Generator das Request Objekt mit dem Befehl übergeben, dieses um den zu transformierenden Quelltext zu erweitern. Je nachdem, welcher Generator zuständig ist, wird die Anfrage entweder an einen Application Shadow weitergereicht oder das Gateway selbst wird mit der Erzeugung von JSML aktiv.

Der durch den Generator in den Request eingefüllte JSML Quelltext muss in der nunmehr folgenden Transformationsphase in das zum Endgerät passende Zielformat transformiert werden. Hierzu überprüft die Sitzungssteuerung, ob zu dem im Request vermerkten `Agent Type` ein gültiger Transformationsprozessor vorliegt. Ist dieses der Fall, so wird dem verantwortlichen Transformator das `JbsaRequest` Objekt übergeben und dem JSML Quelltext im Request Objekt eine korrespondierende Repräsentation in der geforderten Zielsprache zugeordnet. Konnte hingegen kein passender Transformator identifiziert werden, so erfolgt die Übersetzung in einfachen Text, die allgemeinste mögliche Zielrepräsentation.

Die auf diese Weise erbrachte Schnittstellenbeschreibung in der Zielrepräsentation wird in der nun folgenden Sendephase aus dem Request Objekt extrahiert und in ein `Update Response` Objekt eingefüllt. Dieses wird dem Externen Kommunikationsadapter über seine `ECA` Schnittstelle wieder zurückgegeben und kann in der für den Kommunikationskanal passenden Weise an das Endgerät weitergeleitet werden.

Der `Request Processor` gibt in der abschließenden Aufräumungsphase zuvor zwischengespeicherte Objekte frei und wird letztlich selbst entsorgt.

Ein wichtiges Realisierungsmerkmal der Sitzungssteuerung ist, dass alle zur Anfrage gehörigen Informationen ausschließlich im Request Objekt zusammengefasst sind. Auf diese Weise ist ein eventuell notwendiges Ausbrechen aus der Reihenfolge der Teilschritte möglich. Ergibt sich beispielsweise nach Abschluss

der Inhaltsgenerierung ein Fehlerfall, so kann der zuvor generierte JSML Quelltext problemlos durch eine Fehlermeldung ersetzt werden. Andererseits kann die Erzeugung von Inhalten auch vorgezogen werden, wie dieses im Fall der Neuanmeldung eines Benutzers notwendig wird. Obwohl in beiden Fällen zwar von der eigentlichen Bearbeitungssequenz abgewichen wird, kann dennoch nachfolgend mit dem Transformationsschritt fortgefahren werden, ohne dass die vorherige Ausnahmebehandlung hierauf Einfluss hätte.

6.5.3 Signalisierung in der Sitzungssteuerung

Für die möglichst nahtlose Verbindung der über externe Kommunikationsadapter angeschlossenen Endgeräte mit der Integrationsumgebung ist es notwendig, Steuerungsinformationen mit dem Gerät auszutauschen. Dieses ist insbesondere für den HTTP–ECA der Fall, der einen zustandslosen Kommunikationskanal etabliert und für den die Sitzungssemantik demnach auf den eigentlichen Informationsfluss aufmoduliert werden muss. In der konkreten Umsetzung der Integrationsumgebung existieren eine ganze Reihe von Informationen, die zur Etablierung der Sitzungssemantik von Gateway, Session Manager sowie ECA in den Datenstrom eingeschleift werden und die nach Auslösung einer Aktion vom Browser in Form von Datenfeldern oder anderweitig kodiert über den ECA wieder an die Sitzungssteuerung zurückfließen.

Welche Informationen werden zur Sitzungssteuerung benötigt? Von elementarer Wichtigkeit ist zunächst die Möglichkeit zur Reidentifikation der Sitzung über mehrere aufeinander folgende Zugriffe auf die ECA–Instanz. Dieser Anforderung wird im Fall des HTTP–ECA mittels *URL–Encoding* oder *Cookies* begegnet, größtenteils durch das zugrunde liegende Servlet. Darüber hinaus fügt der ECA die Kennung der Gerätesitzung in den Datenstrom ein. Indem eintreffende Anfragen von einem Endgerät auf solche Kennungen untersucht werden, kann die Zugehörigkeit der Anfrage zu einer existierenden Sitzung sofort erkannt und die Anfrage korrekt zugeordnet werden. Diese Form des *Out–Of–Band Signalling* wird in der prototypischen Umsetzung ansonsten lediglich durch den segmentierenden ECA für WAP Geräte benutzt. Dieser kodiert die Nummer der aktuell angezeigten Seite des lokalen Zwischenspeichers in die Adresse ein, über welche sich der WAP Browser im Falle einer Aktion beim ECA meldet. Auf diese Weise kann er durch die Seiten des Zwischenspeichers navigieren, ohne dass hierfür Rückfragen bei der Sitzungssteuerung anfallen würden.

Weitaus üblicher ist die Einbettung von Informationen der Sitzungssteuerung in die Datenfelder, die nach Auslösen von Aktionen auf dem Endgerät an den ECA gesendet werden. Diese werden direkt in den JSML Quelltext eingefügt, der in die jeweilige Zielrepräsentation transformiert und dann an das Endgerät weitergeleitet wird. Bei diesem als *In–Band Signalling* bezeichneten Verfahren werden vornehmlich Befehle kodiert, die der Benutzer zur unmittelbaren Sitzungssteuerung an die Integrationsumgebung senden kann. Abbildung 6.6 zeigt ein Beispiel dieses Vorgehens. Die im oberen Bereich des Browserfensters sichtbaren Knöpfe (Submit, Refresh, Suspend usw.) wurden von der Sitzungssteuerung in die Ausgabe eingeblendet. Wird beispielsweise der Suspend–Knopf zur vorübergehenden Persistierung der Sitzung gedrückt, so werden die in Abbildung 6.13 dargestellten Daten an den ECA übermittelt. Dieser extrahiert die

```
POST /hydelets/eca/71171 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, [...]
Accept-Language: de
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Host: www.hydeparc.de:8080
Content-Length: 83
Connection: Keep-Alive

ctrlsusp=Suspend&x674368547=a1039791068&a635040732=20.08.2001
```

Abbildung 6.13: Nach Drücken des in Abb. 6.6 sichtbaren Suspend–Knopfes an den ECA übermittelte Daten

übertragenen Datenfelder und leitet sie als Teil eines Requests an das Gateway weiter, wo das übermittelte `ctrlsusp` Feld als Signal für die Sitzungssteuerung erkannt wird.

Die Verarbeitung der empfangenen Signale erfolgt in der Quellenauswahlphase der Sitzungssteuerung. Hier wird durch die Analyse der eingegangenen Datenfelder die Verantwortung verschiedener Teilkomponenten der Integrationsumgebung für die weitere Bearbeitung der Anfrage überprüft. Signale, die durch den Prefix „`ctrl`“ eingeleitet werden, zeigen an, dass die Erzeugung des JSML Quelltextes durch die Sitzungssteuerung erfolgen muss. Der Prefix „`ac`“ weist die Rolle des Inhaltsliferanten der Zugangskontrolle zu. Wurde schließlich kein Signal aus einer der genannten Signalfamilien übertragen, so fällt die Aufgabe dem Application Shadow zu.

Signal	**Modul**	**Bedeutung**
ctrlrefr	App. Shadow	Aktualisierung der Schnittstelle
ctrlsubm	App. Shadow	Übertragung von Daten an die Anwendung
ctrlresu	Session Mgr.	Wiederaufnahme einer Sitzung
ctrlnews	Session Mgr.	Einrichtung einer neuen Sitzung
ctrlstrt	Session Mgr.	Start einer neuen Anwendung
ctrlswit	Session Mgr.	Wechsel der aktiven Anwendung
ctrlquit	Session Mgr.	Beenden der aktuellen Sitzung
aclogin	Access Mgr.	Übertragung des Benutzernamens
acpasswd	Access Mgr.	Übertragung des Passwortes

Tabelle 6.1: Die Steuerungssignale der Border Service Architektur

Im Resultat konnte durch Nutzung von In–Band–Signalisierung erreicht werden, dass das Endgerät kein explizites Wissen über die Steuerung der Integrationsumgebung besitzen muss. Alle notwendigen Informationen werden in Form von Datenfeldern in den Datenstrom eingefügt, der an das Endgerät ausgeliefert wird. Nach Auslösen eines Ereignisses in der dargestellten Schnittstelle laufen gewisse Steuerinformationen als Teil der übertragenen Nutzdaten an die Inte-

grationsumgebung zurück und können sodann in der oben skizzierten Weise weiterverarbeitet werden. Eine bewusst steuernde Intervention des Gerätes findet dabei jedoch nicht statt.

6.6 Umkehrung der Rollenverteilung mittels Push

Der im Bereich von Browsersystemen übliche Informationsfluss basiert auf einer aktiven Rolle des Dienstnutzers. Indem ein Hyperlink angeklickt oder der Aktualisierungs–Knopf gedrückt wird, kommt eine Kommunikationsverbindung mit dem Server zustande und die angeforderte Information wird zum Browser übertragen. Diese Vorgehensweise, entsprechend der vorherrschenden Flussrichtung als *Information Pull* bezeichnet, ist im Umfeld mobiler Anwendungen als nicht ökonomisch anzusehen, da die Änderung der entfernten Information nur durch fortlaufende Überprüfung (sog. „Polling“) erkannt werden kann. Finden die Zugriffe über langsame, kostenintensive Medien statt, so ist der ergebnislose Zugriff unbedingt zu vermeiden. Gerade für mobile Anwendungen bietet sich daher die Umkehrung der Rollenverteilung an, so dass fortan der Diensterbringer den Dienstnutzer darüber informiert, dass sich die darzustellende Information geändert hat, woraufhin dieser lediglich einen einzigen Zugriff benötigt, um die Darstellung zu aktualisieren. Der *Information Pull* wandelt sich zum *Information Push*, und das Polling kann entfallen.

Der in dieser Arbeit betrachtete Anwendungsfall der Übertragung grafischer Schnittstellen eignet sich in besonderem Maße für den Übergang zum Information Push. Applikationsschnittstellen ändern mit recht hoher Wahrscheinlichkeit auch ohne Benutzerintervention ihren Zustand: Asynchron eintreffende Nachrichten von entfernten Rechnerknoten, Meldungen über das Ende zuvor angestoßener Hintergrundprozesse oder Ereignisse aus anderen Quellen müssen dem Benutzer präsentiert werden, was sich zumeist in einer Änderung der Schnittstelle, beispielsweise durch Einblendung eines Dialogfensters oder durch Änderung eines Textelements, äußert.

In der prototypischen Umsetzung der Integrationsumgebung wurde das Push Paradigma am Beispiel des HTTP–ECA untersucht und für WAP Endgeräte umgesetzt. In der WAP Spezifikation wird seit der Version 1.2 eine Push Architektur standardisiert. Einen Überblick über WAP Push gibt [Sch00], Details sind der Spezifikation (siehe [For99a, For99d, For99b, For99c]) zu entnehmen. An dieser Stelle sollen die Zusammenhänge daher nur grob skizziert werden.

WAP Push definiert drei Rollen, welche in einer Push Architektur besetzt werden müssen:

Push Initiator : Der Diensterbringer, von welchem die Informationen ausgehen.

Push Proxy Gateway : Der Vermittler, welcher Nachrichten vom Push Initiator entgegennimmt und sie unter Verwendung eines speziellen *Over–The–Air* (OTA) Protokolls an den Dienstnehmer weiterleitet.

Client : Der Empfänger, welcher die Nachrichten dem Benutzer präsentiert.

Die Rolle des Push Proxy Gateways wird dabei oftmals durch das WAP Gateway übernommen, während es sich beim Klienten um ein entsprechend angepasstes WAP Endgerät (Telefon, PDA) handelt, so dass im Rahmen der Umsetzung vor allem die Funktion des Push Initiators betrachtet werden muss.

Aus konzeptioneller Sicht tritt die gesamte Java Border Service Architektur inklusive der durch sie zugreifbaren Anwendung als Push Initiator auf. Auf der dem Push Proxy Gateway zugewandten Seite steht dabei der Externe Kommunikationsadapter. Wie zuvor bildet der ECA eine Abstraktionsschicht in Richtung des verwendeten Kommunikationskanals, wobei sich die Kommunikationsrichtung umkehrt: Vom Gateway eintreffende Nachrichten werden entgegengenommen und in einem speziellen, IP–basierten Protokoll, dem *Push Access Protocol*, an das Push Proxy Gateway weitergeleitet, von wo sie schließlich im WAP spezifischen *Push Over–The–Air* zum Endgerät gelangen.

Grundsätzlich beruht die Umsetzung von Push, wie gesagt, auf der Umkehrung der Kommunikationsrichtung. In der Java Border Service Architektur wurden die betroffenen Funktionsmodule dafür jeweils um die erforderliche Schnittstelle erweitert: Das JBSA Gateway um eine Schnittstelle, über welche der Application Shadow Änderungen vermelden und das daraufhin erzeugte JSML Dokument übermitteln kann, und der HTTP–ECA um eine Schnittstelle, über welche das Gateway die transformierte Repräsentation zur Weiterleitung an den Klienten übergeben kann. Abbildung 6.14 gibt einen Überblick über die verwendeten Schnittstellen und skizziert die einzelnen Schritte, auf die im Folgenden näher eingegangen wird.

Zur Unterstützung von Push–Konzepten muss der Application Shadow von einer vornehmlich passiven zu einer aktiven Komponente der Integrationsumgebung erweitert werden. Hierzu wurde der Analysemechanismus soweit angepasst, dass nunmehr beim Aufbau der Schattenhierarchie die Überwachung weiterer Ereignisse einbezogen wird. Neben das zuvor bereits betrachtete Hinzukommen oder Wegfallen von Elementen tritt die Betrachtung von Ereignissen, welche die Änderung ihres Zustands zum Inhalt haben. Jede Änderung in einem der GUI Elemente führt nun zu einer selbständigen Analyse durch den Shadow, der das Ergebnis dem JBSA Gateway über dessen `pushJsml`–Methode mitteilen kann. Neben dem eigentlichen JSML Code übergibt der Application Shadow dabei eine Referenz auf sich selbst, um die Zuordnung zu einer Anwendungssitzung möglich zu machen.

Bei der Adaption des JBSA Gateways steht die Berücksichtigung von Informationen der verschiedenen Sitzungskontexte im Vordergrund. Damit das JBSA Gateway den passenden Transformator nutzen kann, muss die Gerätesitzung bekannt sein, durch welche die Zielrepräsentation an das Endgerät ausgeliefert werden soll. In der Gerätesitzung ist der vom ECA ermittelte Gerätetyp abgelegt, mit dessen Hilfe die Selektion des Transformators durchgeführt werden kann. Die Gerätesitzung wiederum ist im JBSA Gateway nicht direkt bekannt, sondern nur über die korrespondierende Anwendungssitzung zu erschließen, die vom Gateway unter Rückgriff auf den Session Manager verwaltet wird. Mit der vom Shadow übergebenen Referenz auf sich selbst kann das Gateway die passende Anwendungssitzung vom Session Manager erfragen. Aus ihr lässt sich dann

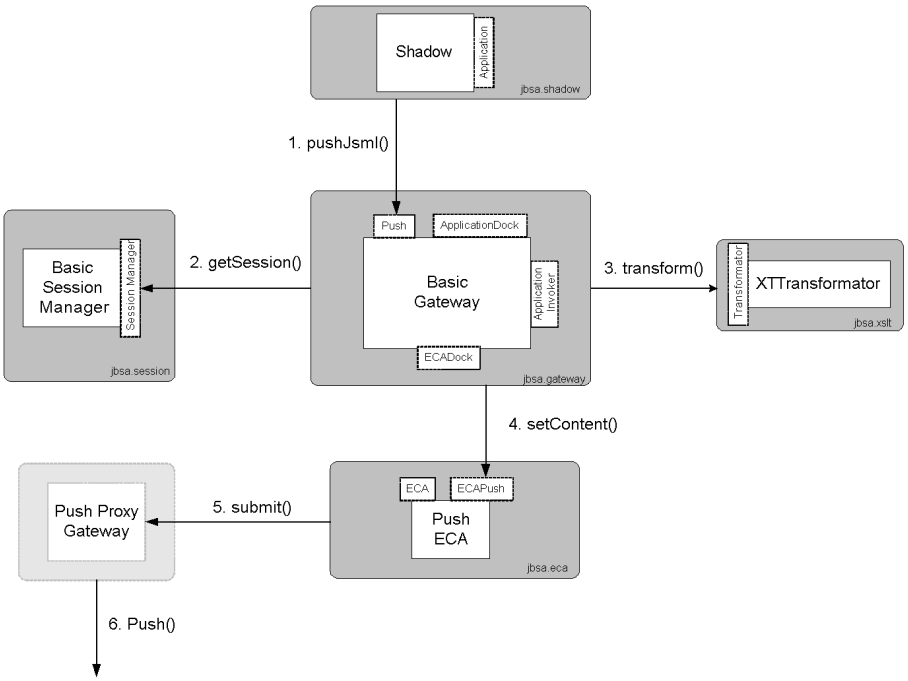


Abbildung 6.14: Verarbeitung von Push Nachrichten durch die JBSA

die Gerätesitzung extrahieren und so schließlich der Gerätetyp erschließen.

Bei näherer Betrachtung der Sitzungssteuerung wird deutlich, dass sich hinsichtlich der Verarbeitung einige Unterschiede zwischen Push und Pull ergeben. Zunächst muss keine Überprüfung der Zugangskontrolle erfolgen; die Sitzung, in welcher der Shadow aktiv ist, muss ohnehin bereits zur Verfügung stehen, da nur in einer gültigen Sitzung ein Application Shadow instantiiert wird. Auch kann die Bestimmung der Quelle des JSML Codes entfallen, da der Application Shadow hierfür bereits feststeht. Schließlich muss die Erzeugung des Inhalts nicht mehr durch die Sitzungssteuerung koordiniert werden, da auch dieser Teilschritt bereits durch den Shadow abgeschlossen wird, bevor die Sitzungssteuerung nach Aufruf von `pushJsml` zum Zuge kommt.

Insofern beschränkt sich die Steuerung durch den `PushRequestProcessor` somit auf die Transformations– sowie die Sendephase. Wie zuvor wird nach Eingang der Informationen ein `JsmlRequest` erstellt, dem die Referenzen auf die Anwendungssitzung und auf den Gerätettyp sowie der vom Shadow stammende JSML Code übergeben werden. Der Request wird in eine Instanz des `PushRequestProcessors` eingefüllt, welcher als eigenständiger Thread die Transformation sowie die hierauf folgende Übertragung zum ECA einleitet und koordiniert. Die Übertragung zum ECA erfolgt hierbei wiederum in einem Containerobjekt, dem `PushUpdate`, in welchem die generierte Zielrepräsentation nebst einer Referenz auf die Gerätesitzung übergeben wird.

Der `PushECA` nimmt die Aktualisierung über seine `Push` Schnittstelle entgegen. Über die Referenz auf die Gerätesitzung identifiziert er den zugehörigen Kom-

munikationskanal und kann so das betroffene Endgerät feststellen. Nach einer Segmentierung der vom Gateway erhaltenen Zielrepräsentation erstellt er eine spezielle Datenstruktur, in welche er die für das Push Proxy Gateway benötigten Kontrollinformationen sowie das erste Segment der Schnittstellenbeschreibung einfüllt. Schließlich baut er eine Kommunikationsverbindung zum Push Proxy Gateway auf und überträgt ihm die Daten nebst einer Referenz auf das Endgerät, welches in der Gerätesitzung vermerkt ist. Über das Push Proxy Gateway gelangen die Daten zum Endgerät, wo sie schließlich angezeigt werden können.

6.7 Zusammenfassung

Dieses Kapitel diente einer Beschreibung der konkreten Umsetzung zuvor allgemein eingeführter konzeptioneller Ergebnisse der vorliegenden Arbeit. Es wurde nochmals die Aufteilung der Integrationsumgebung in funktionale Module skizziert und diese wurden sodann punktuell in größerem Detail beschrieben. Dabei wurde auf die Markup Sprache JSML als Grundlage zur abstrakten Beschreibung von Schnittstellen genauso eingegangen wie auf Application Shadow und Externe Kommunikationsadapter, die verantwortlich für die Etablierung der Abstraktionsschicht in Richtung der Anwendungsseite sowie in Richtung der zu integrierenden Endgeräte zeichnen. Darüber hinaus wurden auch Einblicke in die Funktionsweise der Sitzungssteuerung gegeben und erklärt, wie die Phasensteuerung zur Verzahnung der einzelnen Teilschritte der Generierung, Umformung, Anzeige und entfernten Nutzung von Abbildungen einer grafischen Anwendungsschnittstelle. Dabei wurde auch auf technische Details wie die Motivation für eine Kombination von Anwendung und Shadow in einer JVM und deren Realisierung geblickt.

Schließlich wurde auch auf den Freiraum eingegangen, den die gewählte Architektur zur nachträglichen Erweiterung der Integrationsumgebung um zusätzliche Funktionalität bietet. Anhand von Beispielen wie die Umkehrung der vorherrschenden Kommunikationsrichtung durch Etablierung eines Push Protokolls oder die nachträgliche Segmentierung von bereits transformierten Inhalten durch den Externen Kommunikationsadapter wurde dargestellt, wie dieses in einer Weise vonstatten gehen kann, bei der die Kernkomponenten der Umgebung unberührt bleiben.

Nicht alle Aspekte der konkreten Umsetzung konnten im Zuge dieses Kapitels im Detail beschrieben werden, teils weil hierdurch der Rahmen dieser Abhandlung gesprengt worden wäre, teils weil die Erläuterungen nicht zum weiteren Verständnis des Konzepts und seiner Umsetzung beitragen könnten. So wurde auf Ausführungen zur Komponente zur Dienstselektion verzichtet. Wenngleich diese im Rahmen der Einbindung des Integrationskonzepts in die Middleware JINI eine nicht unwichtige Rolle spielt, ist aus ihrer Beschreibung nur wenig Relevantes zur Verdeutlichung des Integrationsansatzes zu entnehmen. Auch wurde nur am Rande auf die Klassifikationen von Anwendungen durch den Application Class Catalog eingegangen, da dieser nur im Zusammenhang mit der Dienstselektion zum Einsatz kommt. Schließlich wurde nur grob auf den Device Class Catalog eingegangen, da dessen Funktion derzeit größtenteils vom JBSA Gateway sowie dem Externen Kommunikationsadapter übernommen wird und

im Übrigen die meisten der im vierten Kapitel vorgestellten Konzepte zum Aufbau einer Klassenhierarchie derzeit nicht durch gängige Implementationen von Webservern usw. unterstützt werden.

Kapitel 7

Die „Jini Small Device Adapters“ (JiSDA)

Das unter Abschnitt 4.4.3 beschriebene Konzept der opaken Treiber wurde im Rahmen einer prototypischen Implementation umgesetzt. In diesem Kapitel werden Aspekte jener Umsetzung beschrieben. Dazu wird zunächst nochmals im Detail auf die Schritte der Dienstnutzung in JINI eingegangen. Im Einzelnen werden die Schritte der Erschließung der Infrastruktur (Discovery & Join), der Registrierung des Dienstes bei einem Lookup Service, die Vermittlung von Dienstangeboten an einen Dienstnehmer sowie dessen Zugriff auf den Diensterbringer mit Hilfe des übertragenen Treibers erläutert. Vor dem Hintergrund dieser Details wird dann das in dieser Arbeit entworfene Konzept der Nutzung opaker Treiber beschrieben und dabei auf wesentliche Realisierungsdetails eingegangen. Schließlich wird anhand eines Beispiels gezeigt, wie ein mobiler Diensterbringer unter Verwendung eines opaken Treibers in eine JINI Umgebung eingebunden werden kann.

7.1 Dienstregistrierung und Dienstlokation in JINI

Bereits in den Abschnitten 4.4.2.3 sowie 4.4.3 wurde auf die zentrale Stellung eingegangen, die der Prozess der Dienstregistrierung bzw. Dienstlokation in der Konstruktion dynamischer verteilter Anwendungsszenarien mit JINI einnimmt. Wie dort angedeutet, bildet die hierbei erfolgende Übertragung von Treibern eine Grundlage für die flexible Kommunikation zwischen Dienstnehmer und Diensterbringer, deren Details dabei weitgehend vor dem Dienstnehmer verborgen bleiben können. Da es letztlich die Aspekte dieses Registrierungsprozesses sind, die eine Einbindung nicht java–fähiger Diensterbringer in eine JINI Umgebung zunächst unmöglich machen, soll an dieser Stelle zunächst eine genaue Betrachtung der hiermit verbundenen Abläufe erfolgen.

7.1.1 Erschließung der Infrastruktur – Discovery

Vor seinem Eintritt in eine JINI Umgebung ist es für jeden Teilnehmer notwendig, die für die Umgebung verantwortlichen Lookup Services zu identifizieren. Diensterbringer (zur Anmeldung ihres Dienstes) genauso wie Dienstnehmer (zur Auffindung des passenden Dienstangebots) müssen die entsprechenden Instanzen finden und nutzen, um an einem entsprechenden Anwendungsszenario teilnehmen zu können.

JINI stellt zu diesem Zweck drei Verfahren zur Verfügung, derer sich ein Dienst(–nehmer) bedienen kann:

Unicast Discovery Protocol : In dieser statischen Variante werden die vorhandenen Lookup Services mitsamt ihrer TCP/IP Adresse fest in den Diensterbringer bzw. Dienstnehmer eingespeichert. Die Kontaktaufnahme erfolgt ausschließlich mit zuvor bekannten Instanzen.

Multicast Request Protocol : Bei Eintritt in das Netzwerk sendet der Teilnehmer Botschaften an eine bestimmte Multicast Adresse, welche zu allen Lookup Services reflektiert wird, die sich hierfür registriert haben. Interessierte Lookup Service Instanzen können sich dann an den Teilnehmer wenden.

Multicast Announcement Protocol : In regelmäßigen Abständen senden Lookup Services ihrerseits Nachrichten an eine bestimmte Multicast Adresse, um ihre Existenz zu melden. Wie zuvor werden diese Nachrichten weitergeleitet, in diesem Fall an interessierte Dienstnehmer und Diensterbringer.

Die erstgenannte Form der Discovery unterscheidet sich nicht von anderen Verfahren, wie sie in herkömmlichen Middleware Systemen verwendet werden. Der Diensterbringer bzw. Dienstnehmer baut eine Verbindung mit einem fest vorgegebenen Lookup Service auf und wickelt hierüber die weitere Kommunikation ab. Die Adresse des angesprochenen Lookup Service muss dem Teilnehmer dabei zuvor bekannt sein. Als Teil der Unicast Discovery wird auch der Treiber vom Lookup Service an den Teilnehmer übertragen, und so verbirgt sich letztlich auch hinter den anderen Verfahren die Unicast Discovery: Sofern der Teilnehmer via Multicast von der Existenz eines Lookup Service erfahren hat, erfolgt die weitere Kommunikation, d.h. die Übertragung des Treibers zur Anmeldung oder Abfrage des Lookup Service, mittels Unicast Discovery Protokoll.

Im Falle des Multicast Request Protocol können Dienste und Diensterbringer selbst aktiv werden, um ihre Infrastruktur zu erkennen. Hierbei wird ein so genanntes Multicast Request Paket vom Teilnehmer ausgesendet, das von allen Lookup Services empfangen werden kann, die sich bei einer bestimmten Multicast Adresse angemeldet haben. Wie aus Abbildung 7.1 ersichtlich ist, wird das Paket durch eine Versionsnummer eingeleitet, welche die vom Teilnehmer verstandene Protokollversion kennzeichnet. Diese wird gefolgt von einer Portnummer, über welche der Teilnehmer auf Antworten von Lookup Services wartet. Schließlich kommen zwei Listen, nämlich die Namen der Djinns, deren Lookup Service relevant sind, und die Identifikationen eventuell zuvor erkannter Lookup Services.

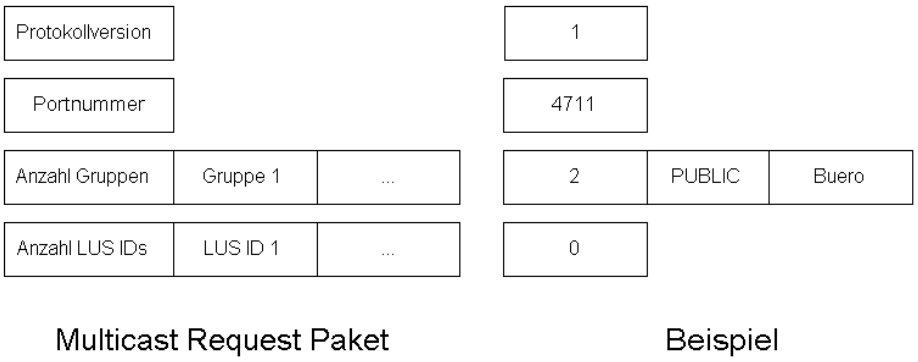


Abbildung 7.1: Aufbau des Multicast Request Pakets

Der Teilnehmer stellt ein solches Paket nach seinem Start zusammen und sendet es periodisch an die festgelegte Multicast Adresse. Danach lauscht er kurzfristig auf mögliche Antworten, bevor er erneut ein entsprechendes Paket aussendet. Haben sich zwischenzeitlich Lookup Services beim Sender gemeldet, so fügt er deren Identifikation dem Paket zu, bevor dieses verschickt wird. Dieser Vorgang wird einige Male wiederholt, bevor anzunehmen ist, dass sich alle erreichbaren Lookup Services gemeldet haben, und der Discovery Prozess schließlich abgebrochen wird.

Wird einem Lookup Service über die Multicast Adresse ein Request Paket zugestellt, so kann er zunächst anhand der Versionsnummer erkennen, ob er ein vom Sender des Pakets unterstütztes Registrierungsprotokoll beherrscht. Gegenwärtig existiert nur ein solches Protokoll, so dass dieser Test zumeist quasi trivial verläuft. Über die im Paket genannten Gruppen kann er des Weiteren feststellen, ob der Sender an einer von ihm verwalteten Gruppen interessiert ist. Abgesehen von der allgemeinen, unbenannten Gruppe, die als „public group“ bezeichnet wird und die jeder Lookup Service unterstützen sollte, kann der Lookup Service weitere Gruppen wie „Wohnzimmer“ oder „Präsentationsraum“ verwalten. Er wird nur dann auf ein Request Paket antworten, wenn darin eine ihm bekannte Gruppe enthalten ist. Aus Gründen höherer Effizienz kann der Sender schließlich die Kennungen derjenigen Lookup Services im Request Paket kodieren, deren Antwort er bereits erhalten hat. Nur wenn der Empfänger eines Multicast Request Pakets seine Identifikation nicht als Teil des Pakets empfängt, sollte er hierauf reagieren. In seiner Antwort wendet er sich dabei an die Adresse des Senders, wobei er die im Request Paket spezifizierte Portnummer verwendet.

Im Gegensatz zum Multicast Request Protokoll, das vom Teilnehmer angestoßen wird, geht beim Multicast Announcement Protokoll die Initiative vom Lookup Service aus. Dieser sendet während seiner Laufzeit in regelmäßigen Abständen ein Announcement Paket an einen Multicast Verteiler, von dem aus es an alle registrierten Empfänger weitergeleitet wird. Wie Abbildung 7.2 zeigt, wird auch dieses Paket durch eine Versionsnummer eingeleitet, gefolgt von Angaben über den Lookup Service, von dem das Paket ausging. Neben dem Rechnernamen und der Portnummer handelt es sich dabei um die Identifikation des Lookup Service. Wie im Request Paket wird auch im Announcement Paket eine Liste von Gruppen übertragen, in diesem Fall die Gruppen, die der Sender vertritt.

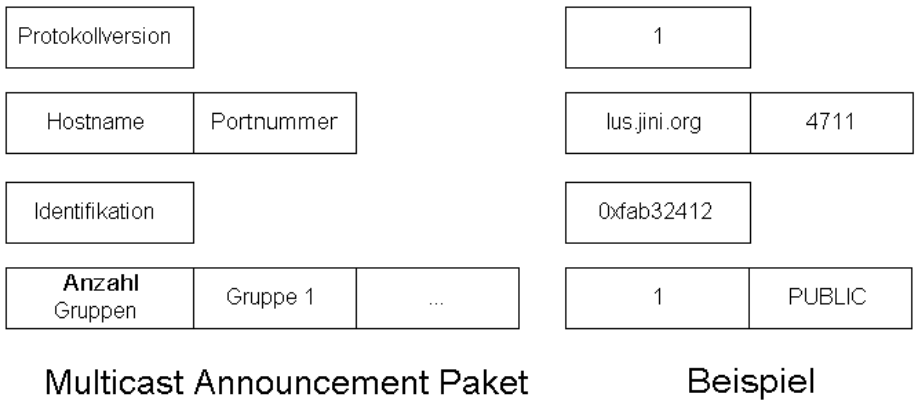


Abbildung 7.2: Aufbau des Multicast Announcement Pakets

In Übereinstimmung mit dem zuvor beschriebenen Verhalten des Lookup Service bei Eintreffen eines Request Pakets handelt auch der Teilnehmer einer JINI Umgebung bei Erhalt eines Announcement Pakets. Er überprüft zunächst anhand der Protokollversion, ob er das verwendete Protokoll beherrscht. Ist dieses der Fall, kann er sodann anhand der Identifikation feststellen, ob der gemeldete Lookup Service bereits bekannt ist und schließlich anhand der Gruppen, ob dieser eine passende Gruppe vertritt. Ist dieses der Fall, kann er zur weiteren Kontaktaufnahme mit dem Lookup Service fortschreiten.

Alle drei Varianten der Discovery gleichen sich ab dem Zeitpunkt, zu dem Lookup Service und Teilnehmer die Identität des jeweils Anderen kennen. In diesem Moment fordert der Teilnehmer den Lookup Service zur Übertragung eines Stellvertreterobjekts auf, des Treibers, mit dessen Hilfe der Teilnehmer fortan mit dem Lookup Service kommunizieren soll. Bei dem Treiber handelt es sich um ein serialisiertes Java Objekt, das die Aufrufschnittstelle des Lookup Service darstellt. Es wird vom Empfänger aus dem empfangenen Datenstrom in ein echtes Objekt umgewandelt und kann nunmehr zur Registrierung bzw. zur Abfrage von Dienstangeboten genutzt werden. Wie bereits oben angedeutet wird dieser Teil des Discovery–Prozess der Einfachheit halber dem Unicast Discovery Protokoll zugeschlagen.

7.1.2 Registrierung eines Dienstes – Join

Nach Abschluss des einleitenden Discovery–Prozesses kann in einem zweiten Schritt die Anmeldung eines Dienstangebots durchgeführt werden[1]. Der Diensterbringer bedient sich dabei des Treibers, den er zuvor vom Lookup Service erhalten hat. Er ruft dessen `register` Methode auf, der er das Dienstangebot in Form eines `ServiceItem` Objekts sowie die gewünschte Gültigkeitsdauer der Registrierung übergibt. Als Antwort auf die erfolgreiche Registrierung erhält der

[1] Im Gegensatz zu anderen Middleware Systemen wird bei JINI im Zuge der Registrierung genau genommen kein Dienstangebot (also lediglich Informationen *über* den Dienst) angemeldet, sondern ein vollständiges Dienstobjekt, das in der Laufzeitumgebung des Empfängers selbst aktiviert wird.

Diensterbringer ein `ServiceRegistration` Objekt zurück. Dieses enthält unter anderem eine eindeutige Kennung, die den Diensterbringer für seine gesamte Lebenszeit identifiziert. Alle folgenden Registrierungen bei anderen Lookup Services müssen mit dieser Kennung erfolgen.

Im `ServiceItem` werden das Dienstobjekt selbst sowie die Menge beschreibender Attributobjekte (so genannte `Entry` Objekte) zusammengefasst. Wie unter Abschnitt 4.4.3 erläutert, gibt es verschiedene Varianten von Dienstobjekten, wobei das RMI Objekt den einfachsten Fall darstellt. Das zu registrierende Objekt wird ebenso wie die `Entry` Objekte instantiiert und dann in ein gemeinsames `ServiceItem` Objekt eingefügt. Wesentlich sowohl für Attribut– als auch für Dienstobjekte ist deren Serialisierbarkeit. Da JINI zur Registrierung von Dienstangeboten auf RMI zurückgreift und im Zuge der Registrierung zuvor instantiierte Objekte verwendet, müssen diese vor Übertragung zum Lookup Service entsprechend vorbereitet, d.h. serialisiert werden.

Attribute dienen zur näheren Beschreibung des Dienstobjekts. Mit ihrer Hilfe können unveränderliche sowie dynamische Merkmale des Dienstes spezifiziert und als Teil der Anmeldung übergeben werden. In einfachen Attributobjekten werden solche Merkmale als Zeichenketten oder Zahlen(–objekte) abgelegt, die vom Empfänger eines Dienstangebots direkt ausgelesen werden können. Komplexere Attribute können darüber hinaus als so genannte Java Beans realisiert werden, bei denen der Zugriff auf die enthaltenen Werte über entsprechende Zugriffsmethoden kontrolliert wird. Über den Umweg von `get--` und `set--`Methoden lässt sich dabei wie für Dienstangebote ein Protokoll realisieren, mit dem Attributwerte direkt beim Dienst erfragt werden können. Sofern das Dienstangebot in die Laufzeitumgebung des potentiellen Nutzers übertragen ist, kann dieser auf die Methoden zugreifen. Die aufgerufene Methode übernimmt dann die Wertabfrage mit einem zwischen Dienstangebot und Diensterbringer vereinbarten privaten Protokoll. Attribute können über das bei Registrierung übertragene `ServiceRegistration` Objekt manipuliert werden. Es können Attributwerte abgefragt oder geändert sowie dem Dienstangebot weitere Attribute hinzugefügt werden.

Das an den Lookup Service übergebene Dienstangebot wird von diesem in einem Objektspeicher abgelegt. In der aktuellen Version von JINI handelt es sich bei diesem Speicher um Java Spaces [Sun99a], eine Tupelraum Implementation in Java. Prinzipiell könnte jedoch auch jede andere Form der Speicherung verwendet werden, bei der Wahl des Objektspeichers handelt sich ausschließlich um ein Implementationsdetail des Lookup Service.

7.1.3 Abfrage von Dienstangeboten – Lookup

Im Fall des Dienstnehmers sollte nach dem Discovery–Schritt die Suche nach einem geeigneten Diensterbringer folgen. In JINI wird dieser Vorgang als „Lookup“ bezeichnet. Wie beim Join des Diensterbringers kommt auch beim Lookup des Dienstnehmers der zuvor vom Lookup Service übertragene RMI Stellvertreter zum Einsatz.

Der Dienstnehmer spezifiziert die Eigenschaften des gewünschten Dienstangebots in einem `ServiceTemplate` Objekt. Zumeist werden im `ServiceTemplate`

eine oder mehrere Java Objektklassen vorgegeben, die der Dienst ausfüllen soll. Darüber hinaus können auch konkrete Merkmale vorgegeben werden, die der gesuchte Dienst erfüllen muss. Neben der expliziten Angabe von Attributwerten kann dabei auch die bloße Existenz eines bestimmten Attributs gefordert werden.

Das zuvor erstellte `ServiceTemplate` wird in einem RMI Aufruf der Methode `lookup` an den Lookup Service übergeben, der in seiner internen Datenbasis nach passenden Angeboten sucht. Etwaige Ergebnisse werden in einer Liste von `ServiceItem` Objekten an den Aufrufer zurückübertragen. Die übertragenen `ServiceItem` Objekte können anhand der jeweils gespeicherten Attribute vom Empfänger weiter untersucht werden, und es kann so unter ihnen letztlich das passende Dienstangebot ausgewählt werden.

Der im ausgewählten Dienstangebot abgelegte Treiber wird aus der Liste extrahiert und kann nun zur Kommunikation mit dem entfernten Dienst verwendet werden. Da der Treiber bereits in der Laufzeitumgebung des Nutzers vorhanden ist, steht seine Methodenschnittstelle entsprechend direkt zur Verfügung.

7.1.4 Erneuerung von Angeboten — Leasing

In JINI werden alle Beziehungen zwischen Teilnehmern einer verteilten Umgebung mittels Leasing zeitlich begrenzt. Jede Nutzung entfernter Ressourcen wird zu Beginn mit einer maximalen Nutzungsdauer gekennzeichnet. Spätestens zu deren Ablauf ist es die Pflicht des Nutzers, die Ressource erneut – und wiederum für eine vorzugebende Dauer – zu belegen. Erfolgt die Erneuerung der Beziehung nicht oder wird diese vom Besitzer der Ressource abgelehnt, so verfällt die Bindung zwischen Nutzer und Anbieter.

Auch Dienstangebote stellen keine Ausnahme dieser Regelung dar. Will ein Diensterbringer ein `ServiceItem` registrieren, so muss er eine Dauer angeben, für welche die Registrierung beim Lookup Service gültig bleiben soll. Rechtzeitig vor Ablauf der Registrierung muss er das Dienstangebot erneuern bzw. die Gültigkeit seiner Registrierung verlängern. Dabei steht der Nutzer der Ressource selbst in der Pflicht, die Verlängerung zu initiieren. Er hat über die belegten Ressourcen Buch zu führen und anstehende Verlängerungen einzuleiten. Der Besitzer der Ressource darf die Belegung hingegen jederzeit fallen lassen, nachdem die Gültigkeitsdauer der Registrierung überschritten wurde.

Wie für die Registrierung selbst nutzt der Diensterbringer auch für die Verlängerung der Registrierungsdauer einen RMI Stellvertreter, den er zuvor vom Lookup Service erhalten hat. Hierzu fragt er über die Schnittstelle des bei Registrierung seines Dienstes erhaltenen `ServiceRegistration` Objekts das zur Registrierung gehörige `Lease` Objekt ab. An diesem kann er sodann eine `renew` Methode aufrufen, um die Gültigkeitsdauer der Registrierung zu verlängern. Schlägt die Verlängerung fehl, so ist die Registrierung als verfallen anzusehen. Glückt die die Verlängerung hingegen, so ist sie bis zu ihrem erneuten Ablauf gültig. Sie muss dann wie zuvor vom Diensterbringer erneuert werden.

7.2 JiSDA: Ein Treiberkonzept für ressourcenschwache JINI Dienste

Vor dem Hintergrund der genaueren Beschreibung der JINI Discovery, Join sowie Lookup Protokolle soll nunmehr erläutert werden, wie das in dieser Arbeit entworfene Konzept der opaken Treiber umgesetzt werden kann. Wie aus obigen Ausführungen hervorgeht, sind im Wesentlichen drei Aspekte zu betrachten:

- JINI macht im Rahmen seiner Discovery und Join Protokolle intensiven Gebrauch von Java RMI. Um die Unterstützung nicht java–fähiger bzw. nicht RMI–fähiger Diensterbringer zu ermöglichen, ist die geeignete Abänderung der Protokolle notwendig. Dieses betrifft ebenfalls die Behandlung von Registrierungen, die in JINI per RMI verwaltet werden.
- Die Zusammenstellung des Treibers und der beschreibenden Dienstattribute in einem Dienstangebot erfolgt in JINI üblicherweise zur Laufzeit, wobei im Zuge der Serialisierung des Dienstangebots Bezüge zur Laufzeitumgebung einkodiert werden. In der Erstellung opaker Treiber sowie im zu realisierenden Übertragungsprotokoll muss dieses Berücksichtigung finden.
- Insbesondere die Belegung dynamischer Dienstattribute ergibt sich erst zur Laufzeit des Dienstes bzw. bei Übergabe des Dienstangebots an den Lookup Service. Da der opake Treiber oftmals bereits vor dem Start des Dienstes fixiert wird, muss ein spezielles Protokoll die Aktualisierung auch nach Registrierung des Angebots jederzeit ermöglichen.

Bei allen Punkten stand die Maßgabe im Vordergrund, ohne den Einsatz komplexer bzw. umfangreicher Bibliotheken auskommen zu können. Die Realisierung der Protokolle sowie deren Nutzung sollte unter Verwendung von ANSI C möglich sein, der im Bereich mobiler Geräte gängigsten Programmiersprache. Wichtig war weiterhin, dass der Realisierungskern sich problemlos auf neue Plattformen portieren lassen sollte. Entsprechend wurden alle benötigten Unterstützungsfunktionen selbst realisiert, ohne dass hierbei auf externe und womöglich plattformspezifische Bibliotheken zurückgegriffen wurde.

Vorausgesetzt wurde lediglich die Fähigkeit des einzubeziehenden Geräts zur Verwendung des TCP/IP Protokolls sowie zur Teilnahme an IP Multicast Gruppen. Beide stellen gemeinsam die Basis aller JINI Protokolle dar, so dass deren Umgehung ohnehin als nicht sinnvoll zu erachten ist.

Darüber hinaus war unbedingt zu gewährleisten, dass notwendige Modifikationen an bestehenden JINI Protokollen bzw. den Diensten, welche diese Protokolle realisieren, zur bestehenden Funktionalität kompatibel waren. Ein abgeänderter Lookup Service soll weiterhin in der Lage sein, Registrierungen herkömmlicher JINI Dienstleister zu verarbeiten und Anfragen herkömmlicher JINI Dienstnehmer zu beantworten. Auch sollen bisherige Lookup Services mit modifizierten Lookup Services in gewohnter Weise interagieren können, beispielsweise beim Aufbau von Föderationen. Alle genannten Forderungen wurden in JiSDA berücksichtigt.

7.2.1 Ein modifiziertes Lokationsprotokoll für JINI

Am Beginn der Beziehung zwischen Diensterbringer und Lookup Service steht bei JINI das Discovery Protokoll. Wie bereits beschrieben, existieren mit Multicast Discovery, Multicast Announcement und Unicast Discovery drei Protokollvarianten, die hierbei zum Einsatz kommen können. Am Ende der Discovery Phase steht in jedem Fall die Unicast Discovery, in deren Verlauf der Lookup Service ein `ServiceRegistrar` Objekt als RMI Stellvertreter an den Diensterbringer überträgt. Zunächst ist es also notwendig, die Discovery Protokolle so zu erweitern, dass sie neben dem bisherigen Vorgehen auch einen Verzicht auf die Übertragung und Nutzung dieses RMI Stellvertreters ermöglichen.

Wie beschrieben, nutzt JINI die Versionsnummer zur Kennzeichnung der vom Teilnehmer verwendeten Protokollversion. Die nachträgliche Erweiterung von JINI um abgewandelte Discovery, Join sowie Lookup Protokolle ohne Beeinträchtigung existierender JINI Netzwerke stellt also kein grundsätzliches Problem dar. Der angepasste Lookup Service kann die vom Diensterbringer oder Dienstnehmer erwartete Protokollversion erkennen und in Abhängigkeit der Protokollversion auf herkömmliche Teilnehmer anders antworten als auf nicht java–fähige Teilnehmer. Versucht andererseits ein nicht java–fähiger Diensterbringer mit einem herkömmlichen Lookup Service Kontakt aufzunehmen, so kann dieser allein auf der Basis der Protokollversion die weitere Kommunikation ablehnen, woraufhin der Diensterbringer sich an einen anderen Lookup Service wenden kann.

Abgesehen von der Verwendung einer anderen Protokollnummer sind die Schritte des modifizierten Multicast Request Protokolls zunächst mit denen des urpsrünglichen Protokolls identisch. Der Diensterbringer sendet Multicast Request Pakete an den vorgegebenen Multicast Port und wartet auf Antworten von Lookup Services, die mit dem Port verbunden sind und das angeforderte Protokoll beherrschen. Abweichungen ergeben sich dann beim Übergang zum Unicast Discovery Protokoll. Sendete der Lookup Service zuvor auf die Aufforderung des Diensterbringers hin neben einer Liste verwalteter Gruppen einen RMI Stellvertreter zurück, ist es im modifizierten Protokoll nun die Information über die Socket–Verbindung, die er dem Diensterbringer für die weitere Kommunikation anbieten kann. Im Einzelnen sind dieses Rechnername und IP Adresse, Identifikation des Lookup Service, Port Adresse der angebotenen Registrierungsverbindung sowie eine Liste der vom Lookup Service verwalteten Gruppen.

Empfängt ein Diensterbringer eine solche Antwort, kann er mit den enthaltenen Informationen eine einfache Socket–Verbindung mit dem Lookup Service aufbauen. Diese Socket–Verbindung ersetzt sodann die zuvor notwendige RMI Kommunikation. Während an dieser Stelle zuvor das Dienstangebot zusammengesetzt und an den Lookup Service übertragen wurde, sendet der Diensterbringer nach dem neuen Ansatz nunmehr Informationen über den Ort, an dem sich das vorbereitete Dienstangebot befindet, in Form einer URL über die Socket–Verbindung sowie den Namen der Treiberklasse, die dort abgelegt ist. Der Lookup Service empfängt die Information, baut eine Verbindung zu der spezifizierten Adresse auf und übernimmt von ihr den opaken Treiber. Hierfür realisiert der Diensterbringer einen rudimentären Webserver, der den Zugriff auf den gespeicherten Treiber über das HTTP Protokoll ermöglicht. Der zusätzliche Indirektionsschritt bietet dabei zwei signifikante Vorteile gegenüber einer direkten Übertragung des opaken Treibers durch den Diensterbringer:

1. Der Treiber muss nicht unbedingt im Festspeicher des Diensterbringers abgelegt werden.
2. Zur Übernahme sowie Instantiierung des Treibers können Standardmechanismen von Java bzw. JINI zur Anwendung kommen.

Zwar stand im Entwurf des Treiberkonzepts die Unterstützung von *Ad–Hoc* Konstellationen im Vordergrund, bei denen der Diensterbringer als in sich geschlossener und vollständiger Teilnehmer eines spontanen Zusammenschlusses von Teilnehmern auftreten kann. Insofern sollte nach Möglichkeit der Treiber direkt im Gerät abgelegt sein und vom Lookup Service dort abgerufen werden. Andererseits war es jedoch ein wichtiger Aspekt, besonders ressourcenarme Diensterbringer zu unterstützen. Für diese kann nicht in jedem Fall vorausgesetzt werden, dass der notwendige Speicherplatz zur Aufnahme eines opaken Treibers zur Verfügung steht. Es ist also wünschenswert, dass Treiber auch außerhalb des Geräts abgelegt werden können, ohne dass hierfür ein weiteres Alternativprotokoll notwendig würde.

Im Übrigen sollte die Unterstützung nicht java–fähiger Diensterbringer sich möglichst homogen in die bestehende JINI Architektur einfügen. Die Übernahme entfernter Objekte über eine Socket–Verbindung und deren Instantiierung aus einem Binärblock heraus stellt ein gängiges Vorgehen in Java dar und wird im Zusammenhang mit Applets häufig gebraucht. Die Übertragung vorab serialisierter Java Objekte unter Angabe einer URL und deren Aktivierung in der Laufzeitumgebung des Empfängers erfordern entsprechend keine weiteren Anpassungen innerhalb der Java Umgebung. Die Modifikation am Lookup Service beschränkt sich somit darauf, bei Nutzung des alternativen Protokolls nach Erhalt der URL unter Verwendung der Standardmethode `loadClass` den opaken Treiber zu übernehmen und diesen in der eigenen Laufzeitumgebung zu aktivieren.

Im Gegensatz zur herkömmlichen Anmeldeprozedur werden Treiber und Attribute nicht gemeinsam in einem Dienstangebot gebündelt und übertragen, sondern getrennt voneinander behandelt. Nachdem der Treiber in der Laufzeitumgebung des Lookup Service aktiviert wurde, müssen demgemäß nunmehr die Attribute übertragen werden. Wie zuvor der Treiber selbst, werden nun auch die zugehörigen Attributobjekte von der entfernten Quelle angefordert und nacheinander aktiviert, so dass schließlich mit Treiber und Attributen der Inhalt eines regulären Dienstangbots in der Laufzeitumgebung des Lookup Service vorliegt. Im Fall der Attribute werden keine vollständigen Objekte übertragen, sondern die in ihnen enthaltene Information mit Hilfe eines frei wählbaren Protokolls übermittelt. Da die Übernahme vom in der Laufzeit des Lookup Service aktivierten Treiber selbst kontrolliert wird, kann der Entwickler beliebige Daten an den Lookup Service übertragen, bevor die Generierung der Attribute beginnt. So können eventuell dynamische Wertbelegungen ausgetauscht werden oder die tatsächlich erstellten Attribute vom Zustand der Anwendungsumgebung abhängig gemacht werden.

Zum Abschluss muss nun der Treiber mitsamt den dazugehörigen Attributen als Dienstangebot registriert werden. Um auch in diesem Zusammenhang die Modifikation des Lookup Service streng begrenzen zu können, nimmt dieser hierzu die Position eines tatsächlichen Diensterbringers ein. Der aktivierte Treiber

wird gemeinsam mit den Attributen in einem `ServiceItem` zusammengefasst und dann über die nunmehr lokale `ServiceRegistrar` Schnittstelle des Lookup Service angemeldet. Von diesem Zeitpunkt an ist das Dienstangebot für die weitere Nutzung innerhalb der JINI Umgebung ununterscheidbar von anderen Angeboten, die von herkömmlichen JINI Diensterbringern registriert wurden.

7.2.2 Aufbau des opaken Treibers

Nachdem der vorige Abschnitt die Registrierung des opaken Treibers beim Lookup Service behandelt hat, soll nunmehr auf den Aufbau des Treibers sowie seiner Attribute näher eingegangen werden.

Um Modifikationen auf den Lookup Service bzw. die Kommunikation zwischen diesem und dem Diensterbringer beschränken zu können, war es notwendig, auf Änderungen des eigentlichen JINI Treiberkonzepts zu verzichten. Aus der Sicht des Dienstnehmers unterscheidet sich der von einem nicht java–fähigen Diensterbringer stammende Treiber nicht von einem herkömmlichen Treiber. Entsprechend der Ausführungen unter 4.4.3 sind somit auch bei Verwendung opaker Treiber grundsätzlich vier verschiedene Arten denkbar, auf die Treiber in JINI konstruiert werden können:

- Lokal beschränkt auf die Laufzeitumgebung des Empfängers,
- passiv unter Verwendung von RMI,
- aktiv unter Verwendung von RMI sowie
- aktiv unter Verwendung alternativer Protokolle (z.B. Sockets).

Offensichtlich eignen sich für die Verwendung im Zusammenhang mit nicht java–fähigen Diensterbringern nur die erste sowie die letzte Alternative. Der erste Fall ist dabei quasi trivial: Der Treiber wird in die Laufzeitumgebung des Dienstnehmers übertragen und dort, beispielsweise zur Berechnung einer Prüfsumme, ausgeführt.

Als weitaus realistischer ist die vierte Alternative anzusehen. Auch hierbei wird der Treiber in die Laufzeitumgebung des Dienstnehmers übertragen und dort aktiviert. Über die vom Treiber angebotene Aufrufschnittstelle kann der Dienstnehmer sodann mit dem Diensterbringer in Kontakt treten. Das zwischen beiden Kommunikationspartnern verwendete Protokoll hängt dabei einzig von den Fähigkeiten des Diensterbringers bzw. der Komplexität des Treibers ab. Im einfachsten Fall wird der Datenaustausch über eine Socket–Verbindung stattfinden. Da der Diensterbringer bereits zur Interaktion mit dem Lookup Service eine auf Sockets basierende HTTP Verbindung verwendet, bietet sich ein solches Vorgehen an. Über diesen Kanal können dann beispielsweise auf XML basierende RPC Mechanismen wie *XML–RPC* oder *SOAP* zum Einsatz kommen, die mit herkömmlichen Methoden wie RMI in ihrer Mächtigkeit vergleichbar sind. Im Bedarfsfall ist jedoch auch die Nutzung höherwertiger Austauschmechanismen wie DCE RPC oder CORBA RPC denkbar. Dieses mag insbesondere dann sinnvoll erscheinen, wenn besondere Effizienz erforderlich ist oder der sichere Rückgriff auf die Dienste eines bestehenden Systems gewünscht ist, wie beispielsweise in Telekommunikationsnetzen zum Tragen kommt (vgl. [MWR98]).

Ähnliches, wie das oben gesagte, gilt auch im Zusammenhang mit den zum Angebot gehörigen Dienstattributen, die zur Beschreibung des jeweils aktuellen Zustands eines Dienstangebots dienen. Nachdem sie mit Hilfe des alternativen Protokolls in der Laufzeitumgebung des Lookup Service erstellt und sodann in einem herkömmlichen Dienstangebot abgelegt worden sind, können sie im Rahmen einer Dienstanfrage zum potentiellen Dienstnehmer übertragen werden. Wird ein solches Attribut über seine Schnittstelle abgefragt, so kann es, genauso wie der Treiber selbst, eine Verbindung mit dem Diensterbringer aufbauen und die gewünschte Information übertragen. Wichtig ist in diesem Zusammenhang lediglich, dass bei der Konzeption der Attribute auf die Definition offener Schnittstellen geachtet wird, wobei sich insbesondere die Verwendung von *Java Beans* (vgl. [Eng97]) oder ähnlicher Ansätze eignet.

7.2.3 Entwicklungsprozess

Wie aus Abbildung 7.3 ersichtlich ist, lässt sich die im nicht java–fähigen Diensterbringer befindliche Software im Wesentlichen in fünf Teilkomponenten zerlegen:

Betriebssystem : Je nach Komplexität kann es sich hierbei um ein vollwertiges Betriebssystem wie WindowsCE oder Embedded Linux handeln oder um eine rudimentäre Schnittstelle zum Zugriff auf die im Gerät enthaltene Hardware.

Applikationskern : Im Applikationskern ist die eigentliche Funktionalität des Diensterbringers untergebracht. Der Systemkern interagiert mit der Diensterbringerschnittstelle über ein privates Protokoll, das im einfachsten Fall auf einfachen Prozeduraufrufen beruht.

Opaker Treiber nebst Dienstattributen : Dieser Teil der Software wird als eine Menge vom Diensterbringer nicht interpretierbarer Objekte in dessen Festspeicher abgelegt und bei Bedarf über den Webserver ausgesandt.

MRP Engine : Die Multicast Request Protokoll Engine dient zur Interaktion mit dem Lookup Service. Sie übernimmt neben der Realisierung des eigentlichen Mutlicast Request Protokolls auch den darauf folgenden Schritt der Unicast Discovery.

Webserver : Der Webserver dient vornehmlich zur Übertragung des opaken Treibers an den Lookup Service. Er kann allerdings darüber hinaus auch als Grundlage der Diensterbringerschnittstelle dienen, beispielsweise bei Austausch von Nachrichten über XML–RPC oder SOAP.

Diensterbringerschnittstelle : Dieser Teil des Anwendungscodes ist verantwortlich für die Realisierung des Protokolls, das zwischen dem in den Dienstnehmer integrierten Treiber und dem Diensterbringer verwendet wird.

Dabei lassen sich drei Gruppen identifizieren, die in Abbildung 7.3 anhand der verwendeten Farben kenntlich gemacht sind:

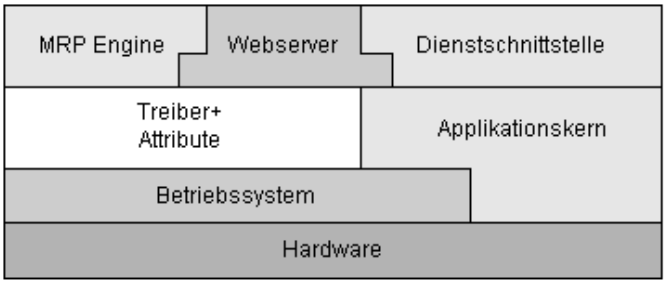


Abbildung 7.3: Funktionale Komponenten eines nicht java–fähigen JINI Diensterbringers

- Dem System zuzurechnende Software, die nicht in unmittelbarem Zusammenhang mit der Nutzung von JINI steht. Hierzu zählt das Betriebssystem genauso wie der Webserver.
- Dem erbrachten Dienst zuzurechnende Sofware. Hierzu zählt der eigentliche Anwendungskern ebenso wie die MRP Engine und die Diensterbringerschnittstelle.
- Vom Diensterbringer nicht interpretierbare Software. Diese umfasst den opaken Treiber nebst Attributen.

Als opake Treiber kommen serialisierte Java Objekte zum Einsatz. Sie unterscheiden sich in ihrem Aufbau nicht von herkömmlichen Objekten, wie sie in JINI–konformen Anwendungen Verwendung finden. Entsprechend kann im Rahmen der Anwendungsentwicklung deren Erstellung und Registrierung bei einem Lookup Service zunächst der Einfachheit halber aus einem herkömmlichen Java Programm heraus erfolgen.

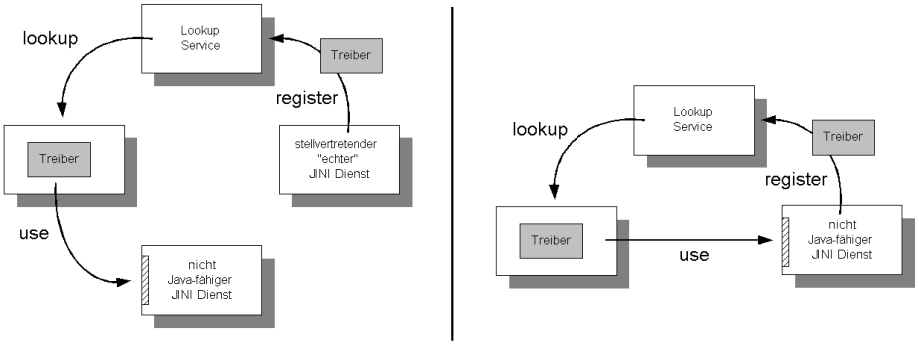


Abbildung 7.4: Beschleunigter Enwticklungsprozess durch Nutzung eines stellvertetenden JINI Dienstes (links)

Besitzt der zu integrierende Diensterbringer bereits eine festgelegte, funktionstüchtige Dienstschnittstelle, so ergibt sich die in Abbildung 7.4 links gezeigte Konstellation. Der zur Kommunikation mit dem Diensterbringer zu entwickelnde Treiber wird auf die für JINI übliche Weise realisiert und vorläufig

durch einen herkömmlichen JINI Dienst angemeldet. Dabei wird das bekannte Registrierungsprotokoll verwendet. In seinen Registrierungsparametern wird dem Treiber als späterer Kommunikationspartner ein Verweis auf den eigentlichen Diensterbringer mitgegeben. Wird der Treiber in den Dienstnehmer übernommen und dort aktiviert, so tritt er mit dem eigentlichen Diensterbringer in Verbindung, wobei das reduzierte Kommunikationsprotokoll zwischen Treiber und Diensterbringer benutzt und getestet werden kann. Das aus der Dienstvermittlungstheorie bekannte „Trading–Dreieck“ wird dabei vorübergehend zu einem Viereck entartet. In diesem Aufbau können Treiber, Diensterbringer und Dienstnehmer relativ einfach getestet werden. Erst zum Abschluss wird dann der Treiber mit einem speziellen Programm versiegelt und in einen opaken Treiber überführt (in Abbildung 7.4 rechts dargestellt). Es wird gegebenenfalls ein Übernahmeprotokoll für Dienstattribute zwischen Lookup Service und Diensterbringer vereinbart und schließlich der opake Treiber fest in den Diensterbringer integriert. Die weitere Kommunikation erfolgt wieder direkt mit dem nicht java–fähigen Diensterbringer und unter Verwendung des alternativen Registrierungsprotokolls.

7.3 JiSDA Beispiel–Szenario: Nutzung einer Miniatur–Kamera

Anhand eines Beispiels soll die Anwendung des entwickelten Konzepts illustriert werden. Im Bereich der Gebäudeüberwachung wird in zunehmendem Maße auf den Einsatz intelligenter Miniatur–Kameras gebaut. Um die Kosten für Installation und Verkabelung zu senken, werden derartige Kameras nicht mehr über spezielle Anschlüsse angesteuert, sondern beherrschen das TCP/IP Protokoll und können direkt an das lokale Computernetzwerk angeschlossen werden. Sie enthalten einen rudimentären Webserver, über den sie wie ein Informationsdienst im Worldwide Web vom Webbrowser angesprochen werden können. In fortlaufend übermittelten Webseiten ist sodann das von der Kamera aufgezeichnete Geschehen zu verfolgen. Moderne Varianten solcher Kameras enthalten mittlerweile einen vollständigen Kleinstrechner, der mit gängigen Programmiersprachen wie C oder C++ programmiert werden kann und ein herkömmliches Betriebssystem wie WindowsCE oder Linux beherbergt.

Abbildung 7.5 zeigt eine Miniatur–Kamera der Firma Axis. Sie verfügt über eine besonders stromsparende CPU, einen ROM–Speicher mit einer Kapazität von 2 MB sowie einen 8 MB großen Flash–RAM Speicher. Basierend auf *Micro Linux* als Betriebssystem–Kern bietet die Kamera eine vollwertige Implementation des TCP/IP Protokolls sowie des HTTP Protokolls. Über einen *Twisted–Pair* Anschluss kann sie direkt an ein Ethernet–Netzwerk angeschlossen werden.

Die Nutzung solcher Kameras in einem Überwachungssystem entspricht genau den Merkmalen eines typischen JINI Szenarios. Miniatur–Kameras stellen klassische Konsumelektronik dar. Sie werden oftmals von ungeschulten Benutzern installiert und sollten demgemäß ohne manuelle Konfiguration auskommen. Beim Einsatz einer größeren Anzahl solcher Kameras im Rahmen eines Überwachungssystems kann sich ein sehr dynamisches Anwendungsszenario ergeben: Kameras werden nach Bedarf an das Netzwerk angeschlossen, später vom Netz getrennt,



Abbildung 7.5: Miniatur–Kamera der Firma Axis mit eingebautem Kleinstcomputer

an einen anderen Ort transportiert und dort wieder an das Netzwerk angeschlossen. Dabei werden sie von einem oder mehreren Überwachungsplätzen aus angesteuert, die jederzeit die Kontrolle über alle verfügbaren Kameras besitzen sollten. Die Registrierung an das Netzwerk angeschlossener Kameras sollte entsprechend automatisch vonstatten gehen.

Aufgrund ihrer stark eingeschränkten Ressourcen ist derzeit der Einsatz einer vollwertigen Java Virtual Machine inklusive RMI in einer marktverfügbaren Miniatur–Kamera wie der Axis Webcam unmöglich. Für die beschriebene Nutzung in einem JINI Szenario ist daher das in dieser Arbeit entwickelte Konzept der opaken Treiber zu favorisieren.

7.3.1 Aufgabe und Funktion des opaken Treibers

Wie skizziert, bietet die in Abbildung 7.5 dargestellte Kamera einen Webserver, über den sowohl die Konfiguration der Kamera vorgenommen als auch das vom Objektiv aufgefangene Bild abgerufen werden kann. Hierzu sendet der Klient (z.B. ein Webbrowser) einen `GET`–Befehl an die IP Adresse der Kamera. In der Antwort wird dann eine Web–Seite zurückgeliefert, die ein Bild im ‚GIF‘–Format enthält.

In einem JINI Szenario soll das zum Einsatz kommende rudimentäre Protokoll vor dem Dienstnehmer verborgen werden. Über eine vollwertige Java Aufrufschnittstelle sollen Bilder von der Kamera angefordert werden können, wobei diese in für die Speicherung von Grafiken in Java üblichen `Image` Objekten verpackt sein sollen. Die Aufgabe des Treibers ist somit, einen entsprechend aktiven Stellvertreter zu verkörpern, der die homogene Einbettung der Kamera–Funktionalität in den Dienstnehmer ermöglicht. Als vollwertiger JINI Treiber soll der Stellvertreter nur zeitgebunden vermittelt werden. Wird eine Kamera vom Netzwerk entfernt, soll ihr Dienstangebot verfallen und der Treiber spätestens

```
public Image getImage(Component c) throws Exception
{
  URL u;
  Image img;

  u = new URL("http://" + host + "/cgi-bin/fullsize.jpg?"
              + System.currentTimeMillis());

  return c.createImage((ImageProducer)u.getContent());

}
```

Abbildung 7.6: Vom opaken Treiber angebotene Methode zur Nutzung durch den Klienten

nach Ablauf der Registrierungsdauer aus dem Lookup Service gelöscht werden. Das im Lookup Service registrierte Dienstangebot soll neben dem eigentlichen Treiber über beschreibende Attribute verfügen. In ihnen werden Konfigurationsparameter wie IP Adresse bzw. Port des Webservers der Kamera gespeichert.

Abbildung 7.6 zeigt die vom Treiber realisierte Aufrufschnittstelle zur Nutzung durch den Klienten. Als Rückgabewert der Methode `getImage` erhält der Dienstnehmer ein `Image` Objekt mit dem aktuellen Bild der Kamera. Die Verwendung des HTTP Protokolls und die angesprochene Webadresse bleiben vor dem Klienten verborgen. Sie sind in der Abbildung nur zur Illustration der Funktionsweise dargestellt. Die Nutzung des Treibers im Rahmen der Programmierung ist stark verkürzt in Abbildung 7.7 ersichtlich. Nach Spezifikation des gesuchten Diensttyps, d.h. der `CameraService` Klasse, erfolgt ein Aufruf der Suchschnittstelle des Lookup Service. Wurde ein passendes Dienstangebot gefunden, so kann die vom Treiber angebotene Methode aufgerufen und das so erhaltene Bild nachfolgend genutzt und beispielsweise angezeigt werden. Wiederum wird deutlich, dass das tatsächlich verwendete Protokoll durch den Klienten nicht ersichtlich ist.

7.3.2 Dienstattribute im Kamera–Szenario

Im Gegensatz zum eigentlichen Treiber werden die Dienstattribute nicht als vollwertige Java–Objekte im Festspeicher der Kamera vorgehalten, sondern in einem privaten Protokoll zwischen dem in den Lookup Service übertragenen Treiber und der Unterstützungssoftware der Kamera ausgetauscht.

Das Vorgehen ist dabei der späteren Nutzung des Treibers durch den Klienten ähnlich. Der Treiber ist verpflichtet, die spezielle Methode `unmarshalParameters` anzubieten, welche vom Lookup Service aufgerufen werden kann, sobald der Austausch der Attribute beginnen soll. Als Aufrufparameter wird der Methode ein Verweis auf die zuvor zwischen Kamera und Lookup Service aufgebaute Socket–Verbindung übergeben. Über diese kann der Treiber sodann die Attributwerte von der Kamera anfordern. Das hierbei verwendete Protokoll ist nicht vorab festgelegt, sondern kann entsprechend den

```
// ...
// RMI Schnittstelle des Lookup Service holen
ServiceRegistrar reg = lookupService.getRegistrar();

// Suchmaske (Template) generieren. Gesucht wird
// nach Instanzen der Klasse "CameraService"
Template template = CameraClient(new LookupLocator("jini://localhost"),
                                 new ServiceTemplate(null,
                                 new Class[] { CameraService.class },
                                 null)

// Lookup Service befragen. Als Aufrufresultat wird
// der beim Lookup Service registrierte Treiber
// an den Klienten zurückgeliefert.
camera = (CameraService)reg.lookup(template);

// Bild abfragen und darstellen. Der Klient verwendet
// die vom Treiber angebotene Aufrufschnittstelle.
Image img = camera.getImage(pane);

// Das von der Kamera übertragene Bild kann
// in herkömmlicher Weise verwendet werden.
canvas.setIcon(new ImageIcon(img));
// ...
```

Abbildung 7.7: Verkürzte Darstellung der Einbettung des JINI Treibers in den Dienstnehmer

von der Anwendung diktierten Anforderungen zwischen Treiber und Diensterbringer frei gestaltet werden. Für die Kamera reicht die einfache Übergabe textueller Informationen über Modell und Systemversion aus. Komplexere Systeme könnten auch die Verschlüsselung der übertragenen Werte oder die Anwendung eines transaktional gesicherten Protokolls vorsehen.

Für jeden von der Kamera übernommenen Attributwert erzeugt die `unmarshalParameters` Methode ein `Entry` Objekt, so dass nach Abschluss der Übertragung eine vollständige Attributmenge vorliegt, welche den aktuellen Zustand des Geräts wiedergibt. Es ist Sache des Treiber–Programmierers, gegebenenfalls komplexe Attributobjekte zu realisieren, die auch nachträglich noch die Kontaktaufnahme mit dem Gerät und so die ständige Aktualisierung des Attributwerts ermöglichen. Die aus der Interaktion zwischen Kamera und der Registrierungsmethode des Treibers hervorgehende Attributmenge wird schließlich als Resultatwert an den Lookup Service zurückgegeben. Sie kann in der folgenden Registrierungsphase weiter verwendet werden.

7.3.3 Interne Registrierung des Kamera Treibers

Nach der Attributübertragung stehen dem Lookup Service nun der eigentliche Treiber sowie die Menge der beschreibenden Dienstattribute zur Verfügung. Zum Abschluss des modifizierten Registrierungsvorgangs nutzt der Lookup Service diese Informationen zur Registrierung des Angebots in seiner Laufzeitumgebung. Er tritt dabei als Stellvertreter der Kamera auf und ruft die von ihm hierfür üblicherweise per RMI Mechanismus an den Diensterbringer übergebene Registrierungsmethode *selbst* auf, wobei er Treiber und Attribute der Kamera als Parameter verwendet und eine vom Treiber vorgegebene Registrierungsdauer festlegt.

Zum Abschluss der abgewandelten Prozedur steht also die Anmeldung beim JINI System in hergebrachter Weise. Für den Lookup Service selbst und vor allem für Dienstnehmer sowie übrige Diensterbringer besteht fortan kein Unterschied zwischen dem durch einen herkömmlichen Dienst registrierten Angebot und dem unter Verwendung des abgewandelten Protokolls durch die Kamera als nicht java–fähiger Dienst registrierten Angebot.

7.3.4 Nutzung des Kamera Treibers durch den Klienten

Der in der JINI Umgebung registrierte Treiber der Kamera kann von Dienstnehmern dazu verwendet werden, auf die Ressourcen des Geräts zuzugreifen. Wie aus der Beschreibung des Treibers deutlich wurde, ist es dem Nutzer des Treibers möglich, durch einfache Methodenaufrufe Bilder von der Kamera anzufordern, die in einem für Java üblichen Datenformat vorliegen und einfach weiterverarbeitet werden können.

Der Klient startet dazu eine Suchanfrage beim Lookup Service. In dieser spezifiziert der die Objektklasse `CameraService` als gesuchten Dienst. Auch die Angabe von Dienstattributen ist denkbar, jedoch nicht notwendig, da die im Dienstangebot registrierten Attribute im Fall der Kamera lediglich informativen

Charakter besitzen. Konnte ein entsprechendes Dienstangebot lokalisiert werden, so wird dieses vom Lookup Service mit den darin enthaltenen Attributen und dem Treiber an den Klienten übertragen. Der Klient extrahiert den Treiber aus dem Angebot und kann sodann direkt darauf zugreifen. Durch fortlaufende Aufrufe der `getImage` Methode des Treibers werden kontinuierlich Bilder der Kamera übertragen, die in einem Anwendungsfenster dargestellt werden. Siehe hierzu Abbildung 7.8.

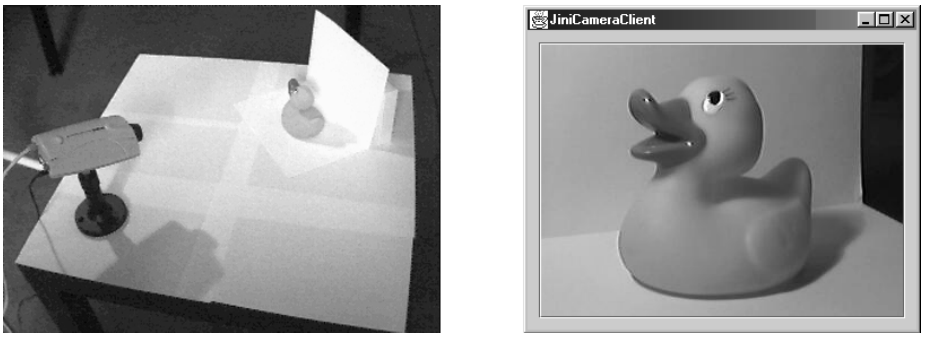


Abbildung 7.8: Darstellung des von der Kamera übertragenen Bildes durch den JINI Dienstnehmer

Bricht die Verbindung der Kamera zum Netz ab oder wird die Kamera abgeschaltet, so scheitert der in den Klienten integrierte Treiber fortan in der Übernahme von Bildern von der Kamera. Die Darstellung ist nun so lange gestört, bis die Registrierungsdauer des Treibers beim Lookup Service und in der Folge auch beim Klienten überschritten wird. Spätestens zu diesem Zeitpunkt ist der Klient gezwungen, eine neue Suchanfrage zu formulieren und an den Lookup Service zu übertragen. Wird die Kamera wieder ans Netzwerk angeschlossen und erneuert diese die Registrierung beim Lookup Service, so wird der Klient hiervon durch den Lookup Service informiert, der Treiber kann wieder übertragen werden und die Darstellung von Bildern erneut beginnen.

7.3.5 Integration in die Systemsoftware der Kamera

Die Integration von JiSDA in die Systemsoftware der Kamera erfolgt in drei Teilkomponenten:

- Realisierung des Multicast Request Protokolls in einer lokalen MRP Engine,
- Behandlung von Anfragen des Lookup Services zur Übernahme von Dienstattributen durch die Diensterbringerschnittstelle,
- Speicherung des Treibers in einem vorab serialisierten Datenblock.

Dabei sind zwecks Speicheroptimierung in diesem Beispiel der für die Übergabe von Attributen verantwortliche Teil der Diensterbringerschnittstelle und die MRP Engine in einem Anwendungsprogramm miteinander kombiniert.

Nach der Systeminitialisierung der Kamera wird die MRP Engine gestartet. Diese beginnt damit, regelmäßig Multicast Requests an die vorgesehene Multicast Adresse des Netzwerks zu senden. Antwortet ein modifizierter Lookup Service auf eines dieser Pakete, so wird das Discovery Protokoll in der bereits geschilderten Weise durchlaufen und dem Lookup Service die Adresse des kamera–eigenen Webservers mitgeteilt. Von dort kann der Lookup Service den Treiber übernehmen und dessen `unmarshalParameters` Methode aufrufen. Diese Methode ruft mittels einer einfachen Socket–Leseoperation die von der Kamera angebotenen Attributwerte bei der Diensterbringerschnittstelle der Kamera ab. Die weitere Registrierung und folgende Nutzung des Treibers erfolgen dann in der oben geschilderten Weise.

Das Konzept der Jini Small Device Adapters ist auf optimale Ausnutzung vorhandener Ressourcen des Diensterbringers ausgelegt. Der gesamte Code des JiSDA Moduls für die Kamera, bestehend aus Treiber, MRP Engine und Attributbehandlung, belegt ca. 43 Kilobytes im Speicher der Kamera. Dabei entfallen auf den serialisierten Treiber etwa 5 kb, die MRP Engine schlägt mit 38 kb zu Buche. Natürlich geben diese Zahlenwerte nur einen groben Hinweis auf tatsächliche Ressourcenanforderungen. Je nach Komplexität des zwischen Treiber und Gerät etablierten Protokolls kann dessen Größe schwanken. Soll der Austausch von Informationen verschlüsselt stattfinden oder ein komplexer Mechanismus zur Ermittlung von Attributwerten Verwendung finden, so sind die Werte nach oben zu verschieben.

7.4 Zusammenfassung

In diesem Kapitel wurde die konkrete Umsetzung des entwickelten Konzepts der opaken Treiber skizziert und seine Funktionsweise anhand eines konkreten Beispiels verdeutlicht.

Dabei wurde zunächst auf die im Zuge der Registrierung neuer Diensterbringer in einer JINI Umgebung verwendeten Protokolle eingegangen. Es wurden die zur Auffindung von Lookup Services notwendigen Multicast Request und Multicast Announcement Protokolle vorgestellt und die in diesem, als Discovery Phase bezeichneten, Schritt ausgetauschten Informationen beschrieben. Im Anschluss wurde beschrieben, wie in einer herkömmlichen JINI Umgebung der nachfolgende Schritt der Dienstregistrierung und der Nutzung von zeitlich begrenzten Bindungen vonstatten geht. Diese Betrachtungen mündeten in die Formulierung von Anforderungen an die konkrete Umsetzung der JINI Small Device Adapters.

Im Anschluss widmete sich das Kapitel dann den Details einer solchen Umsetzung des Konzepts. Es wurde das alternative Lokationsprotokoll beschrieben, das zur Unterstützung nicht java–fähiger Diensterbringer an die Stelle des üblicherweise verwendeten Multicast Request Protokolls tritt. Der Aufbau der Multicast Request Protocol Engine wurde genauso erläutert wie der opake Treiber, der im Gerät gespeichert und zum Zeitpunkt der Kontaktaufnahme mit dem Lookup Service an diesen übertragen wird. Dabei wurde auch auf die Rolle eingegangen, die der modifizierte Lookup Service in der abgewandelten Registrierungsprozedur übernimmt. Es wurde verdeutlicht, dass es seine Aufgabe ist, in dieser Phase als Stellvertreter des eigentlichen Diensterbringers zu agieren und das

Dienstangebot an dessen Stelle zu registrieren. Schließlich wurde in diesem Teil des Kapitels auf den Entwicklungsprozess opaker Treiber eingegangen. Es wurde beschrieben, welche Teilkomponenten der Anwendungsentwickler zu entwerfen hat und wie diese sich in die Softwarearchitektur des modifizierten Gerätes einbetten.

Schließlich wurde das zuvor Gesagte in einem konkreten Beispiel untermauert. Dazu wurde auf die notwendigen Schritte eingegangen, um eine netzwerkfähige Kamera in ein JINI Szenario einzubeziehen. Es wurde auf Funktion und Aufbau des hierzu verwendeten opaken Treibers eingegangen und beschrieben, wie dieser an den Lookup Service übermittelt werden kann. Es wurde weiterhin umrissen, wie der Treiber in den Klienten übernommen wird und schließlich dargestellt, wie dessen Nutzung durch den Klienten erfolgt. So wurden alle wesentlichen Aspekte berührt, die bei der Nutzung von JiSDA zum Tragen kommen.

Kapitel 8

Zusammenfassung und Ausblick

8.1 Zusammenfassung

Ziel dieser Arbeit war es, die Integration mobiler Geräte in verteilte Systemumgebungen zu vereinfachen, indem einerseits die systemtechnische Integration des mobilen Endgeräts vom Prozess der Anwendungsentwicklung entkoppelt wird und andererseits eine automatische *infrastrukturelle* Einbindung des Geräts ermöglicht wird. Dabei wird das mobile Gerät als Nutzer einer vom verteilten System angebotenen Dienstleistung genauso betrachtet wie auch als Anbieter einer solchen.

Wesentliche Beiträge der vorliegenden Arbeit sind hierbei die Einführung einer Abstraktionsebene zwischen Applikationslogik und Präsentationsschicht *laufender* Anwendungen mit dem Ziel der Aufbereitung ihrer Benutzungsschnittstelle für beliebige mobile Zielgeräte sowie der Einsatz opaker Treiber zur Erweiterung einer zuvor auf ressourcenstarke Geräte eingeschränkten Middleware in die mobile Domäne hinein. Das Treiberkonzept ermöglicht es dem mobilen Diensterbringer, die Semantik der eigenen Aufrufschnittstelle vor dem entfernten Dienstnehmer zu verbergen, ohne die dort verwendete Implementationssprache zu beherrschen.

Ausgangspunkt der Arbeit war dabei zunächst die Aufbereitung des thematischen Umfeldes dieser Arbeit. Das Merkmal der Mobilität und seine Behandlung in verteilten Systemen zeigen eine Vielfalt an Ausprägungen, so dass es zunächst sinnvoll erschien, als Grundlage zur Erarbeitung eines *Klassifikationsschemas* eine Aufbereitung bestehender Ansätze vorzunehmen. Das zweite Kapitel umfasst daher zunächst die Darstellung unterschiedlicher Definitionen von Mobilität. Im Ergebnis zeigte sich, dass sowohl die Bewegung des Anwenders als auch die Bewegung des ausführenden Gerätes und letztlich auch die Bewegung des ausgeführten Programmes als Formen einer Mobilität angesehen werden können. Im Rahmen der Zielsetzung dieser Arbeit wurde dabei die Mobilität des Gerätes als Gegenstand der vertieften Betrachtung gewählt, da sie zentral für die *systemtechnische* Konzeption entsprechend verteilter Umgebungen ist.

Im Anschluss wurden typische Merkmale mobiler Geräte herausgearbeitet, um sie als Prädikate zur Aufstellung einer Taxonomie zu nutzen. Zum Abschluss des zweiten Kapitels wurden dann exemplarisch verschiedene konkrete Mobilgeräte beschrieben. Auf diese Weise konnte die praktische Anwendbarkeit der entstandenen Geräteklassen überprüft werden. Neben den anderen angeführten Merkmalen sind es insbesondere der eingeschränkte Nutzungskontext des jeweiligen Geräts und seine geringen Ressourcen, die den Wunsch entstehen lassen, die verarbeiteten Daten vom Mobilgerät zu separieren und dieses lediglich noch als einen austauschbaren *Zugangspfad* zu einem zentralen Informationssystem zu verwenden.

Das dritte Kapitel behandelte im Detail Ansätze, Mobilgeräte in den Kontext eines solchen zentralen Informationssystems zu integrieren und so zu einer effektiven Trennung von Gerät und Daten zu kommen. Die Einteilung der beschriebenen Ansätze erfolgte vor dem Hintergrund des OSI Modells in sieben Kategorien wie der Bitübertragungsschicht, der Sicherungsschicht, der Netzwerkschicht usw. Exemplarisch wurde den meisten der sieben Schichten ein oder mehrere Integrationsansätze zugeordnet und diese detailliert erörtert. Ziel war es hierbei, zunächst die charakteristischen Merkmale der Integration auf der jeweiligen Ebene des OSI Modells herauszuarbeiten.

Ergänzend wurden andere Möglichkeiten der Einordnung diskutiert. Von besonderer Bedeutung für den Ansatz des Autors ist dabei die Anordnung eines Integrationsansatzes in einem Spektrum zwischen Application Awareness, also der durch die Anwendung selbst unterstützten Integration, und Application Transparency, der Integration ohne Unterstützung durch die Anwendung, wie es von Satyanarayanan [SNKP94] vorgeschlagen wurde. Als wesentliche Ergänzung wurde eine entsprechende Erweiterung des Klassifikationssystems vorgeschlagen, um die in dieser Arbeit angestrebte *gerätetransparente* Integration zu ermöglichen. Anhand des so entstandenen erweiterten Klassifikationssystems wurden die zuvor bereits betrachteten Integrationsansätze den sich ergebenden Kategorien zugeordnet. Als ein wichtiges Ergebnis des dritten Kapitels zeigte sich, dass ein Eingreifen auf der Anwendungsebene des OSI Modells ein besonderes Potential für die gerätetransparente Integration aufweist. Auf dieser Ebene kann von einer Vielzahl von technischen Aspekten der Integrationsproblematik abstrahiert und so zu wesentlich flexibleren Integrationsansätzen übergegangen werden.

Ausgehend von dieser Erkenntnis wurde im zentralen vierten Kapitel ein Konzept für einen Integrationsansatz auf eben dieser Anwendungsebene entworfen, wobei der Schwerpunkt zunächst auf die Integration mobiler *Dienstnehmer* gelegt wurde. Existierende Integrationsansätze wurden in drei Kategorien zusammengefasst, mit der Unterscheidung zwischen betriebssystembasierter Integration, applikationsserverbasierter Integration sowie Entwurfszeit–Transformation. Die Vor– und Nachteile der jeweiligen Kategorie wurden ausführlich erörtert, mit dem Ziel, die Anforderungen an einen eigenen Ansatz abzuleiten.

Als zentrale Anforderung an das zu schaffende Integrationskonzept kann dessen maximale Flexibilität formuliert werden. Indem das Spektrum von Mobilgeräten immer breiter und vielfältiger wird, kann ein Ansatz nur noch dann als längerfristig tragfähig bezeichnet werden, wenn er möglichst weite Teile hiervon abdeckt. In besonderem Maße ist für mobile Geräte dabei charakteristisch, dass sie ein oftmals geschlossenes Design aufweisen, das Modifikationen des Gerätes selbst

nur schwer möglich macht. Ein homogenes Systemkonzept muss demnach auf die Merkmale des Gerätetyps und auf seine Fähigkeiten abgestimmt werden können, ohne dass dieses zu Einschränkungen für die Nutzung mit anderen Gerätetypen führt.

Die Einbindung mobiler Dienstnehmer wurde durch die Trennung von Präsentationsschicht und Anwendungslogik mit Hilfe einer abstrakten Zwischenrepräsentation etabliert und unter dem Begriff der Introspektion definiert. Das zugrunde liegende Prinzip beruht auf der Echtzeit–Analyse der grafischen Benutzungsschnittstelle durch eine externe Einheit und die Transformation der gewonnenen Informationen in eine intermediäre Darstellung. Die so gewonnene plattformneutrale Beschreibung der Anwendungsoberfläche kann sodann in einem weiteren Schritt an die Anforderungen des zugreifenden Endgerätes adaptiert und diesem zur Nutzung bereitgestellt werden. In einer zunächst überblicksartigen Beschreibung des resultierenden Ansatzes wurden die funktionalen Anteile identifiziert und umrissen. Darauf folgend wurde im Detail auf jeden der Anteile eingegangen, dieser in seinem theoretischen Aufbau beschrieben und auf existierende Konzepte eingegangen, die zu einer konkreten Umsetzung beitragen.

Im zweiten Teil dieses zentralen Kapitels wurde dann auf die Integration von mobilen Geräten als *Diensterbringer* in einer verteilten Anwendungsumgebung eingegangen, wobei wiederum die Anwendungsschicht als Ausgangspunkt gewählt wurde. Es wurden zunächst mögliche Ansätze identifiziert, wie eine Integration auf dieser Ebene erfolgen kann. Die Verwendung von Middlewaresystemen als Adaptionsschicht zwischen Diensterbringer und Dienstnehmer ist als etablierter Stand der Forschung zu sehen, und so wurde primär darauf abgestellt, welche Möglichkeiten deren Verwendung bietet. Unter den betrachteten Ansätzen zeigte sich die Verwendung von migrierenden Treibern in Systemen wie JINI als sehr flexibel und daher besonders geeignet zur Unterstützung des Ansatzes dieser Arbeit. Als wesentliche Einschränkung des bestehenden Konzepts von JINI wurde die Nutzung von Java als Realisierungssprache für diese Form der *Integration durch Migration* erkannt.

Java bietet zwar durch seine Plattformunabhängigkeit die Möglichkeit zur Verallgemeinerung des Migrationskonzepts, aufgrund seiner vergleichsweise hohen Anforderungen an die Ressourcen des ausführenden Geräts ist seine Verwendung auf mobilen Geräten jedoch derzeit nur sehr eingeschränkt möglich. Bedingt durch seine effiziente Ressourcennutzung ist in diesem Bereich C++ bzw. Assembler als Realisierungssprachen dominant. Da auch zukünftig in diesem Sinne die mobil verfügbaren Technologien jeweils den stationären unterlegen sein werden, wurde ein Konzept entworfen und exemplarisch realisiert, um nicht java–fähige Geräte in eine JINI Umgebung integrieren zu können. Detailliert wurden das in JINI verwendete Integrationsprinzip dargestellt und entsprechende Ansatzpunkte extrahiert, die die Ableitung eines generischen Konzeptes erlauben. Als Ergebnis dieser Analyse wurde das Prinzip des *opaken Treibers* hergeleitet, dieses beschrieben und schließlich mit denkbaren Alternativen verglichen.

Die Tragfähigkeit der zuvor entwickelten Konzepte wurde im Rahmen dieser Arbeit durch die konkrete Umsetzung in einer Integrationsumgebung nachgewiesen. Im zweiten Teil der Arbeit wurde daher näher auf Aspekte dieser Umsetzung eingegangen. Dabei diente das fünfte Kapitel zunächst dem Überblick. Die der Umsetzung zugrunde liegenden Entwurfskriterien sowie deren grundsätzliche

Eigenschaften wurden beschrieben. Anschließend erfolgte die Identifikation der beiden Hauptkomponenten der Gesamtumgebung, die zur Abdeckung der Integration mobiler Dienstnehmer auf der einen Seite und mobiler Diensterbringer auf der anderen dienen. Jeder der Teile wurde in seiner Funktionsweise umrissen, wobei gleichzeitig die funktionalen Module benannt und gegeneinander abgegrenzt wurden, um zu einer klaren Systemarchitektur zu gelangen.

Die sich anschließende Diskussion, ob die Umgebung notwendige Kriterien einer Infrastruktur, einer Middleware bzw. eines Frameworks erfüllt, führte zur Prägung des Begriffs der *Meta–Middleware* zur Einordnung des eigenen Konzepts. Die Bezeichnung der entstandenen Integrationsumgebung als Meta–Middleware soll dabei unterstreichen, dass es sich um ein System handelt, das noch oberhalb eigentlicher Middlewarekonzepte angesiedelt werden kann und ein bestehendes Middlewaresystem überlagern kann. Als eine weitere Möglichkeit der Einordnung wurde schließlich die Erweiterung des gängigen Begriffs der Schichtenarchitektur vorgestellt. Aufgrund der Tatsache, dass sich die Integrationsumgebung als Facette vor einer solchen Schichtenarchitektur darstellen lässt, wurde ihre Bezeichnung als nullte Schicht und die des gesamten Systems als Architektur in ‚0+n' Schichten geprägt.

Nach dieser allgemeinen Einführung der entstandenen integrierenden Systemplattform dienten die folgenden Kapitel dazu, wesentliche Aspekte der Realisierung vorzustellen. Dabei ging das sechste Kapitel zunächst auf die Umsetzung der Integrationsunterstützung für mobile Dienstnehmer ein. Es wurde die entstandene Auszeichnungssprache dargestellt, die Analysekomponente vorgestellt, das Transformationssystem beschrieben und die Komponente zur Ankopplung von Endgeräten erläutert. Das Prinzip der *Integration durch Abstraktion* bzw. Introspektion beruht in hohem Maße auf einer sorgfältigen Verzahnung einer Vielzahl notwendiger Teilschritte. Entsprechend viel Raum wurde daher der Beschreibung von Realisierungsaspekten einer *Sitzungssteuerung* geschenkt, wie sie als elementarer Teil des Steuerungskerns der Gesamtumgebung zu finden ist. Am Ende des Kapitels stand schließlich die Beschreibung der Umsetzung des Push–Konzepts, in dem die vorherrschende Flussrichtung von Informationen umgekehrt und in Richtung des mobilen Endgeräts verlagert wird. Anhand dieses Beispiels wurde abschließend die Flexibilität des Ansatzes des Autors unterstrichen.

Im siebten Kapitel wurde dann auf Realisierungsdetails der Umsetzung des Konzepts der Integration mobiler Diensterbringer durch *Migration* eingegangen. Die verwendeten Protokolle wurden eingehend betrachtet und die zur Umsetzung des Konzepts opaker Treiber notwendigen Anpassungen dargelegt. Im Anschluss wurde dann auf den Aufbau solcher Treiber eingegangen und letztlich exemplarisch anhand eines konkreten Beispiels ihre Verwendung zur Integration nicht java–fähiger Diensterbringer in eine verteilte JINI Umgebung verdeutlicht.

8.2 Ausblick

Im Rahmen dieser Arbeit wurde ein in sich geschlossenes Integrationskonzept zur Einbindung mobiler Diensterbringer sowie Dienstnehmer in verteilte Anwendungsumgebungen entworfen und in seiner konkreten Umsetzbarkeit nach-

gewiesen. Während dabei die wesentlichen Aspekte des Konzeptes ausgearbeitet und realisiert werden konnten, haben sich eine Reihe von Anknüpfungspunkten gezeigt, an denen die Arbeiten fortgeführt und weitere Projekte angeschlossen werden können.

Zum Teil ergeben sich mögliche Erweiterungen in Gebieten, in denen die zur Verfügung stehenden Realisierungstechniken sich erst ansatzweise durchgesetzt haben und die Umsetzung daher nur skizziert werden konnte. So ist die Möglichkeit der Klassifikation von Geräten durch *Geräteprofile* in dieser Arbeit zwar untersucht worden, die konkrete Umsetzung konnte aufgrund der bislang nicht verfügbaren Geräte mit Unterstützung von Standards zur Profilbildung wie etwa CC/PP nur ansatzweise erfolgen. In einem Nachfolgeprojekt der Arbeitsgruppe „Verteilte Systeme“ der Universität Hamburg wird vertieft auf diesen Bereich eingegangen werden und anhand erster Prototypen die Berücksichtigung von Geräteeigenschaften durch Analyse von entsprechenden Profilen in die Integrationsumgebung eingebettet werden.

In diesem Zusammenhang wäre auch eine genauere Untersuchung der Möglichkeit zur weiteren Einbeziehung von Migrationskonzepten in die Unterstützung von mobilen Dienstnehmern zu erwägen. Wird durch das vom Gerät übertragene Geräteprofil beispielsweise die Fähigkeit zur Ausführung von Java–Code signalisiert, könnte die Integrationsumgebung einen entsprechenden Treiber in das Gerät verlagern und so ggf. eine Vorverarbeitung von Benutzerbefehlen durchführen und das Gerät von der ständigen Interaktion mit dem Kern der Integrationsplattform entlasten. Auch wäre es denkbar, dass der vom mobilen Diensterbringer stammende opake Treiber in einem solchen Fall direkt in den mobilen Dienstnehmer übertragen werden könnte, um unter Umgehung der Integrationsumgebung eine direkte Kommunikation zwischen beiden zu erlauben. Entsprechende Arbeiten sind derzeit in der Vorbereitung.

Eine weitere nachfolgende Arbeit könnte sich mit der verstärkten Miniaturisierung der Integrationsumgebung selbst beschäftigen. Im Zusammenhang mit der weiteren Diversifikation der Java Sprachlandschaft wurde kürzlich die *Java 2 Mobile Edition* (J2ME) vorgestellt, die viele der als Motivatoren dieser Arbeit herangezogenen Probleme hinsichtlich zu hoher Ressourcenanforderungen von Java im mobilen Kontext zu lösen verspricht. Es sollte untersucht werden, inwieweit sich die Komponenten der Integrationsumgebung mittels J2ME umsetzen und so selbst auf relativ ressourcenschwache Geräten positionieren lassen. In Verbindung mit Ad–hoc Netzwerktechnologien wie Bluetooth bzw. IEEE 802.11b könnten so höchst dynamische Konstellationen von Dienstnehmern und Diensterbringern zusammengestellt werden, in denen, je nach Ausstattung des Gerätes, Schnittstellen oder ganze Treiber ausgetauscht werden, um die Interaktion zu ermöglichen.

Die entwickelte Zwischenrepräsentation hat sich als genügend umfangreich erwiesen, um den überwiegenden Teil der auf einem Mobilgerät darstellbaren Elemente einer grafischen Schnittstelle zu erfassen bzw. auszudrücken. Die Flexibilität des entstandenen Dialekts wurde nachgewiesen, indem weitere Schlüsselworte integriert werden konnten, ohne dass hierfür grundsätzliche Änderungen am Gesamtkonzept notwendig geworden wären. Lediglich die Introspektionskomponente musste entsprechend angepasst werden. Auch auf der Seite der zu integrierenden Geräte konnte durch bloße Anpassung der Transformationsre-

gelsätze die Einbeziehung weiterer Geräteklassen problemlos ermöglicht werden. Die Erfahrungen waren insgesamt sehr ermutigend und haben dazu geführt, die Entkopplung der Sprache JSML von ihrem ursprünglichen Zweck der bloßen Kodierung zuvor analysierter Schnittstellen als sinnvoll zu erachten. Weiterhin wird untersucht, inwieweit die Border Service Architektur zu einem vollwertigen Entwicklungsrahmenwerk erweitert werden kann, in dem mittels einer Programmierschnittstelle bzw. grafischer Werkzeuge die Erzeugung von Anwendungen ermöglicht wird. Deren Ausgabe basiert dann primär auf JSML und kann durch die JBSA als Applikationsserver flexibel auf beliebigen Zielplattformen verwendet werden.

Auch konnte bisher nur im Ansatz die Leistungsfähigkeit des Konzepts hinsichtlich der Integration beliebiger Kanäle untersucht werden. So gibt es Grund zu der Annahme, dass eine Erweiterung der Externen Kommunikationsadapter um weitere Kanäle mit begrenztem Aufwand möglich wäre. Konkret sind Untersuchungen zur Einbindung gesprochener Sprache angedacht, wie sie unter Verwendung von VoxML, SpeechML oder ähnlichen Repräsentationen realisierbar sind.

Aber auch auf entgegengesetzer Seite der Integrationsumgebung sind Erweiterungen möglich. So könnte anhand einer Introspektionskomponente für das Windows System untersucht werden, welche Probleme der Übergang zu einer auf C++ basierenden GUI Bibliothek bringt, die nicht über die von Java bekannten Möglichkeiten zur Laufzeitanalyse der Schnittstelle verfügt. In diesem Falle müssten andere Wege zur Untersuchung der Anwendungsoberfläche greifen, was sich dank der modularen Architektur im Application Shadow geeignet zusammenfassen lassen sollte.

Schließlich wurden bereits erste, sehr viel versprechende Schritte unternommen, die in prototypischen Arbeiten entstandene Integrationsumgebung aus ihrem akademischen Kontext herauszulösen und einer breiteren Nutzung zuzuführen. So wurde in mehreren industriellen Projekten auf der Basis der realisierten Umgebung in einer Form des *Rapid Prototyping* die Handhabung von Anwendungen auf mobilen Endgeräten getestet. Dieser Bereich sollte in Zukunft zusätzliche Beachtung erhalten, indem die Robustheit und Effizienz der Integrationsumgebung weiter gesteigert werden.

Literaturverzeichnis

[ABK+00] R. Anderson, M. Birbeck, M. Kay, S. Livingstone, B. Loesgen, D. Martin, S. Mohr, N. Ozu, B. Peat, J. Pinnock, P. Stark und K. Williams. *XML professionell*. Programmer To Programmer Series. MITP–Verlag, Bonn, Deutschland, 2000.

[Add96] M.D. Addlesee. The VideoTile. http://www.orl.co.uk/tile.html, 1996.

[AE97] G. Abowd und I. Essa. Ubiquitous and Aware Computing. GVU Newsletter Herbst '97, Georgia Institute of Technology, GVU Center and College of Computing, 1997.

[AGS+93] N. Adams, R. Gold, B.N. Schilit, M.M. Tso und R. Want. An infrared network for mobile computers. In *Proceedings of the USENIX Symposium on Mobile Location–Independent Computing*, Seiten 41–52, Cambridge, MA USA, 1993.

[APB+99] M. Abrams, C. Phanouriou, A. Batongbacal, S. Williams und J. Shuster. UIML: An Appliance Independent XML User Interface Language. In *Proceedings of the 8th International World Wide Web Conference*, Toronto, Kanada, Mai 1999.

[APL+95] E. Ayanoglu, S. Paul, T.F. LaPorta, K.K. Sabnani und R.D. Gitlin. AIRMAIL: A Link–Layer Protocol for Wireless Networks. *ACM Wireless Networks*, 1(1):47–60, 1995.

[AS00] F. Andre und M.T. Segarra. A Generic Approach to Satisfy Adaptability Needs in Mobile Environments. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS '33)*, Maui, HI USA, Januar 2000.

[BAI93] B.R. Badrinath, A. Acharya und T. Imielinski. Impact of Mobility on Distributed Computations. *ACM Operating Systems Review*, 27(2):15–20, April 1993.

[BB97] B.J. Bennington und C.R. Bartel. Wireless Andrew: Experience Building a High Speed, Campus–Wide Wireless Data Network. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '97)*, Budapest, Ungarn, September 1997. ACM/IEEE.

[BBB+94] B. Barringer, T. Burd, F. Burghardt, A. Burstein, A. Chandrakasan, R. Doering, S. Narayanaswamy, T. Pering, B. Richards, T. Truman, J. Rabaey und R. Brodersen. Infopad: A System Design for Portable Multimedia Access. In *Calgary Wireless '94 Conference*, Calgary, Kanada, Juli 1994.

[BBG98] J. Boyer, T. Bray und M. Gordon. Extensible Forms Description Language (XFDL) 4.0. Draft Specification NOTE–XFDL–19980902, Word Wide Web Consortium, September 1998.

[BCG+00] M. Butrico, N. Cohen, J. Givler, A. Mohindra, A. Purakayastha, D.G. Shea, J. Cheng, D. Clare, G. Fisher, R. Scott, Y. Sun, M. Wone und Q. Zondervan. Enterprise Data Access from Mobile Computers: An End–to–End Story. In *Proceedings of the 10th International Workshop on Research Issues in Data Engineering*, Seiten 9–16, San Diego, CA USA, Februar 2000. IEEE Computer Society, IEEE Computer Society Press.

[BCKP95] R. Bagrodia, W.W. Chu, L. Kleinrock und G. Popek. Visions, Issues and Architecture for Nomadic Computing. *IEEE Personal Communications*, Seiten 14–27, Dezember 1995.

[BDL99] J. Bergqvist, P. Dahlberg und F. Ljungberg. Moving Out of the Meeting Room. In S. Boedker, M. Kyng und K. Schmidt, Hrsg., *Proceedings of the 6th European Conference on Computer Supported Cooperative Work, ECSCW '99*, Seiten 81–98, Kopenhagen, Dänemark, September 1999.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte und D. Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note, W3C, 2000.

[Ber96] P.A. Bernstein. Middleware: A Model for Distributed System Services. *Commications of the ACM*, 39(2):86–98, 1996.

[BG99] D. Brickley und R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. W3C Candidate Recommendation CR–rdf–schema–20000327, W3C, März 1999.

[BHJJ97] H.W. Peter Beadle, B. Harper, G.Q. Maguire Jr. und J. Judge. Location Aware Mobile Computing. In *Proceedings of the IEEE/IEE International Conference on Telecommunications (ICT '97)*, Melbourne, Australien, April 1997.

[BI93] B.R. Badrinath und T. Imielinski. Mobile Wireless Computing: Solutions and Challenges in Data Management. Technischer Bericht DCS–TR–296, Dept. of Computer Science, Rutgers University, New Brunswick, NJ USA, 1993.

[Bin00] C. Binnig. Analyse bestehender ausgabeunabhängiger Beschreibungssprachen für GUIs, sowie die prototypische Realisierung einer ausgabeunabhängigen Beschreibung einer vorhandenen GUI. Projektarbeit, Forschungszentrum Karlsruhe, IAI, Karlsruhe, Deutschland, September 2000.

[BKM+97] L. Bass, C. Kasabach, R. Martin, D. Siewiorek, A. Smailagic und J. Stivoric. The Design of a Wearable Computer. In *Proceedings of the 1997 Conference on Computer Human Interaction (HCI '97)*, Atlanta, GA USA, März 1997.

[BKVP97] B. Bakshi, R. Krishna, N. Vaidya und D. Pradhan. Improving performance of TCP over wireless networks. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, Baltimore, MD USA, Mai 1997. IEEE Computer Society, IEEE Computer Society Press.

[BPSK96] H. Balakrishnan, V.N. Padmanabhan, S. Seshan und R.H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *Proceedings of the Annual Technical Conference SIGCOMM '96*, Jgg. 26, Stanford, CA USA, August 1996. ACM.

[Bra87] P.T. Brady. Evaluation of multireject, selective reject, and other protocol enhancements. *IEEE Transactions on Communications*, 35(6):659–666, Juni 1987.

[Bra98] P.J. Braam. The Coda Distributed File System. *Linux Journal*, 4(50):46–51, Juni 1998.

[BRH94] F. Bennett, T. Richardson und A. Harter. Teleporting – Making Applications Mobile. In *Proceedings of the 1st IEEE Workshop on Mobile Computing Systems and Applications (MCSA '94)*, Santa Cruz, CA USA, Dezember 1994.

[Bro52a] F.A. Brockhaus. *Band 10: Sol–Unj.* Der große Brockhaus. F.A. Brockhaus, Wiesbaden, Deutschland, 16. Auflage, 1952.

[Bro52b] F.A. Brockhaus. *Band 5: GP–IZ.* Der große Brockhaus. F.A. Brockhaus, Wiesbaden, Deutschland, 16. Auflage, 1952.

[Bro86a] F.A. Brockhaus. *Band 12: Med–Nen.* dtv Brockhaus Lexikon. Deutscher Taschenbuch Verlag, München, Deutschland, 1986.

[Bro86b] F.A. Brockhaus. *Band 8: Hau–Irt.* dtv Brockhaus Lexikon. Deutscher Taschenbuch Verlag, München, Deutschland, 1986.

[Bro99] R.W. Brodersen. Berkeley Wireless Research Center – Past, Present and Future. Interner Bericht, University of California, EECS Dept., Berkeley, CA USA, 1999.

[BS97] K. Brown und S. Singh. M–TCP: TCP for Mobile Cellular Networks. *ACM Computer Communications Review*, 27(5), Juli 1997.

[BS01] C. Binnig und A. Schmidt. Schichtarbeit – Benutzeroberflächen beschreiben mit UIML. *iX – Magazin für professionelle Informationstechnik*, Seiten 143–147, April 2001.

[C. 00] C. Ogg and C. Costain. I–mode data standard poses challenge to WAP in Europe. Online Article http://www.zdnet.co.uk/itweek/brief/2000/40/network/, ZDWeek IT–Week, Oktober 2000.

[Car97] L. Cardelli. Mobile Computation. In J. Vitek und C. Tschudin, Hrsg., *Mobile Object Systems – Towards the Programmable Internet (MOS '96)*, Lecture Notes in Computer Science No. 1222, Seiten 3–6, Linz, Österreich, April 1997. Springer Verlag.

[Car99] L. Cardelli. Wide Area Computation. In J. Wiedermann, P. van Emde Boas und M. Nielsen, Hrsg., *26th International Colloquium, ICALP '99 Proceedings – Automata, Languagese and Programming*, Lecture Notes in Computer Science No. 1644, Seiten 10–24, Prag, Tschechien, Juli 1999. Springer Verlag.

[CCM99] B.C. Cumberland, G. Carius und A. Muir. *Microsoft Windows NT Server 4.0 Terminal Server Edition Technical Reference*. IT Professional Series. Microsoft Press, Redmond, WA USA, 1999.

[CDK01] G. Colouris, J. Dollimore und T. Kindberg. *Distributed Systems – Concepts and Design*. International Computer Science Series. Pearson Education Ltd., 3. Auflage, 2001.

[CG96] W. Citrin und M.D. Gross. Distributed Architectures for Pen–Based Input and Diagram Recognition. In *Proceedings of ACM Workshop on Advanced Visual Interfaces (AVI '96)*, Gubbio, Italien, Mai 1996.

[Chi95] S. Chiba. A Metaobject Protocol for C++. In *Proceedings of the ACM Conference on Object–Oriented Programming Systems, Languages, and Applications (OOPSLA)*, Jgg. 30 of *SIGPLAN notices*, Seiten 285–299, Austin, TX USA, Oktober 1995.

[CKRH00] S. Campadello, O. Koskimies, K. Raatikainen und H. Helin. Wireless Java RMI. In *Proceedings of the 4th International Enterprise Distributed Objects BIBLIOGRAPHY 55 Computing Conference (EDOC 2000)*, Seiten 114–123, Makuhari, Japan, September 2000. IEEE Computer Society Press.

[Com98] D. Comer. *TCP/IP Illustrated*. Number 1: The Protocols in Addison–Wesley Professional Computing Series. Addison–Wesley, 1998.

[Cox99] D.C. Cox. Wireless Personal Communication: A Perspective. In J.D. Gibson, Hrsg., *The Mobile Communications Handbook*, The Electrical Engineering Handbook Series, Kapitel 15, Seiten 15–1 – 15–49. CRC Press LLC, Dallas, TX USA, 2. Auflage, 1999.

[Cul94] D. Culbert. Low Power Hardware for a High Performance PDA. In *Proceedings of the 1994 IEEE Computer Conference*, San Francisco, CA USA, 1994. IEEE Computer Society Press.

[CWH98] A. Clifford-Winters und R. Hailstone. Windows By Other Means. White Paper 1998/8.1, Data Research DPU, Lidingö, Schweden, August 1998.

[Dal99] M.K. Dalheimer. *Programming with Qt: Writing Portable GUI applications on UNIX and Win32*. O'Reilly & Associates, Sebastopol, CA USA, 1. Auflage, 1999.

[Dat97] DataBeam. A Primer on the T.120 Series Standard. White Paper, Data Beam Corporation, 1997.

[DLS99] P. Dahlberg, F. Ljungberg und J. Sanneblad. Proxy Lady: Mobile Support for Opportunistic Communication. *Scandinavian Journal of Information Systems No. 15*, 1999.

[dLWZ00] E. de Lara, D. S. Wallach und W. Zwaenepoel. Puppeteer: Component–based Adaptation for Mobile Computing. GVU Newsletter TR-00-360, Computer Science Dept., Rice University, Oktober 2000.

[DMCB98] N. Davies, K. Mitchell, K. Cheverst und G. Blair. Developing a Context Sensitive Tourist Guide. In C. Johnson, Hrsg., *Proceedings of the 1st Workshop on Human Computer Interaction with Mobile Devices*, number G98-1 in GIST Technical Report, Glasgow, Schottland, Mai 1998. Dept. of Computing Science, University of Glasgow.

[DPS+94] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M. Theimer und B. Welch. The Bayou Architecture: Support for Data Sharing among Mobile Users. In *Proceedings of the 1994 Workshop on Mobile Computing Systems and Applications*, Seiten 2–7, Santa Cruz, CA USA, Dezember 1994. IEEE Computer Society Press.

[Edw99] W.K. Edwards. *Core Jini*. The Sun Microsystems Press Series. Prentice Hall PTR, Upper Saddle River, NJ USA, 1. Auflage, Juni 1999.

[Eng97] R. Englander. *Developing Java Beans*. O'Reilly & Associates, Sebastopol, CA USA, 1. Auflage, Juni 1997.

[Eng98] Ad Astra Engineering. Jumping Beans White Paper. http://www.JumpingBeans.com, Juni 1998.

[Fan99] A. Fano. What are a Location's „File" and „Edit" Menus? In *Workshop on Situated Interaction in Ubiquitous Computing at CHI 2000*, Den Haag, Niederlande, April 1999.

[Feu97] A. Feuer. *MFC Programming*. Addison–Wesley, 1. Auflage, 1997.

[For99a] WAP Forum. WAP Push Architectural Overview. Technical report, Wireless Application Protocol Forum, 1999.

[For99b] WAP Forum. WAP Push Message Specification. Standard Specification, Wireless Application Protocol Forum, November 1999.

[For99c] WAP Forum. WAP Push OTA Protocol Specification. Standard Specification, Wireless Application Protocol Forum, November 1999.

[For99d] WAP Forum. WAP Push Proxy Gateway Service Specification. Standard Specification, Wireless Application Protocol Forum, November 1999.

[For99e] WAP Forum. WAP Wireless Application Environment Overview. Standard Specification WAP Version 4–Nov–1999, Wireless Application Protocol Forum, November 1999.

[For99f] WAP Forum. WAP Wireless Session Protocol Specification. Standard Specification WAP WSP Version 5–Nov–1999, Wireless Application Protocol Forum, November 1999.

[FS99] J. Flinn und M. Satyanarayanan. Energy–Aware adaptation for mobile applications. In *Proceedings of the 17th ACM Symposium on Operating System Principles*, Kiawah Island Resort, SC USA, Dezember 1999. ACM.

[FZ94] G. Forman und J. Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, Seiten 39–47, April 1994.

[GHJV95] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Design Patterns – Elements of Reusable Object–Oriented Software*. Addison–Wesley Professional Computing Series. Addison–Wesley Longman, Reading, MA USA, 1995.

[Gib99] J.D. Gibson, Hrsg. *The Mobile Communications Handbook*. The Electrical Engineering Handbook Series. CRC Press LLC, Dallas, TX USA, 2. Auflage, 1999.

[Gla99] G. Glass. ObjectSpace's Product Family for State-of–the–Art Distributed Computing. White Paper www.objectspace.com/products/vgrWhitePapers.htm, ObjectSpace Inc., 1999.

[GM98] S.P. Genoway und T.W. Mathers. *Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix Metaframe*. MacMillan, November 1998.

[Gol98] M. Golm. MetaXa and the Future of Reflection. In *Proceedings of the OOPSLA '98 Workshop on Reflective Programming in C++ and Java*, Vancouver, Kanada, Oktober 1998.

[Goo97] D.J. Goodman. *Wireless Personal Communication Systems*. The Addison–Wesley Wireless Communications Series. Addison–Wesley, 1997.

[Gro99] WAP Forum UAProf Working Group. Wireless Application Group User Agent Profile Specification. Standard Specification WAG UAPROF Version 10–Nov–1999, Wireless Application Protocol Forum, November 1999.

[GS99] A. Gokhale und D.C. Schmidt. Techniques for Optimizing CORBA Middleware for Distributed Embedded Systems. In *Proceedings of Infocom '99*, New York, NY USA, März 1999.

[Hal00a] R. Hall. Exchanging Data via XML: Portal–To–Go at a Glance. White Paper, Oracle Inc., Redwood Shores, CA USA, Februar 2000.

[Hal00b] R. Hall. Exchanging Data via XML: Source to XML, XML to Target. White Paper, Oracle Inc., Redwood Shores, CA USA, Februar 2000.

[Har96a] J. Harmer. Preliminary Requirements for Mobile–API, W1D13. OnTheMove Public Deliverable AC034/BT/WP1/DS/P/009/a1, British Telecom, April 1996.

[Har96b] J. Harmer. The OnTheMove Project. In *Proceedings of the 3rd International Workshop on Mobile Multimedia Communications*, Princeton, NJ USA, September 1996. Plenum Publishers.

[Har00] E.R. Harold. *Java Network Programming.* O'Reilly & Associates, Sebastopol, CA USA, 2. Auflage, August 2000.

[HCC99] M. Haahr, R. Cunningham und V. Cahill. Supporting CORBA Applications in a Mobile Environment. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '99)*, Seattle, WA USA, August 1999. ACM/IEEE.

[Heu92] A. Heuer. *Objektorientierte Datenbanken: Konzepte, Modelle, Systeme.* Addison–Wesley, Bonn, Deutschland, 1. Auflage, Juli 1992.

[HHC+99] A. Helal, B. Haskell, J.L. Carter, R. Brice, D. Woelk und M. Rusinkiewicz. *Any Time, Anywhere Computing.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA USA, 1999.

[HHS+99] A. Harter, A. Hopper, P. Steggles, A. Ward und P. Webster. The Anatomy of a Context–Aware Application. In *Proceedings of the 5th Annual ACM /IEEE International Conference on Mobile Computing and Networking (Mobicom '99)*, Seattle, WA USA, August 1999.

[HKL+99] F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel und M. Schwehm. Next Century Challenges: Nexus – An Open Global Infrastructure for Spatial–Aware Applications. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '99)*, Seattle, WA USA, August 1999. ACM/IEEE.

[HKR99] H. Hjelmeroos, P. Ketola und K.-J. Raiha. Coping with consistency under multiple design constraints: the case of the Nokia 9000 WWW browser. In *Proceedings of the 2nd Workshop on Human Computer Interaction with Mobile Devices*, Seiten 51–57, Edinburgh, Schottland, 1999.

[HS99] M. Halkia und N. Streitz. The role of roomware and sensing technology for supporting narratives in ubiquitous computing environments. In *Workshop on Situated Interaction in Ubiquitous Computing at CHI 2000*, Den Haag, Niederlande, April 1999.

[HTK98] J. Herstad, D. Van Thanh und S. Kristoffersen. Wireless Markup Language as a Framework for Interaction with Mobile Computing Communication Devices. In C. Johnson, Hrsg., *Proceedings of the 1st Workshop on Human Computer Interaction with Mobile Devices*, number G98-1 in GIST Technical Report, Glasgow, Schottland, Mai 1998. Dept. of Computing Science, University of Glasgow.

[Hum00] A.P.J. Hum. Fabric Area Network – An Ubiquitous Wireless Communications Infrastructure for Intelligent Ware. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL USA, Juli 2000. International Institute of Informatics and Systemics.

[HV99] M. Henning und S. Vinoski. *Advanced CORBA Programming with C++*. Addison–Wesley Professional Computing Series. Addison–Wesley Longman, Februar 1999.

[IB94] T. Imielinski und B.R. Badrinath. Mobile Wireless Computing: Challenges in Data Management. *Communications of the ACM*, 37(10), Oktober 1994.

[IDJ91] J. Ioannidis, D. Duchamp und G.Q. Maguire Jr. IP–based Protocols for Mobile Internetworking. In *Proceedings of the 1991 ACM SIGCOMM Symposium on Communications Architectures & Protocols*, Seiten 235–245, Zürich, Schweiz, September 1991. ACM.

[IK96] T. Imielinski und H.F. Korth, Hrsg. *Mobile Computing*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA USA, 1996.

[Jac88] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of the SIGCOMM '88 Conference*, Seiten 314–329, Stanford, CA USA, August 1988. ACM.

[JHE99a] J. Jing, A. Helal und A. Elmagarmid. Client–Server Computing in Mobile Environments. *CSURV: Computing Surveys*, 31, 1999.

[JHE99b] J. Jing, A. Helal und A. Elmagarmid. Client–Server Computing in Mobile Environments. *ACM Transactions On Database Systems*, Preliminary Version, 1999.

[JW01] D. Jones und R. Woundy. Addition of Device Class to Agent Options. Internet Draft draft–ietf–dhc–agentoptions–device–class–01, IETF Dynamic Host Control Working Group, Februar 2001.

[JWDH95] A. Joshi, S. Weerawarana, T.T. Drashansky und E. Houstis. SciencePad: An Intelligent Electronic Notepad for Ubiquitous

Scientific Computing. In *Proceedings of the International Conference on Intelligent Information Management Systems (IASTED '95)*, Washington, DC USA, Dezember 1995. Purdue University, Dept. of Computer Science.

[JWW+95] A. Joshi, S. Weerawarana, R. Weerasinghe, T. Drashansky, N. Ramakrishnan und E. Houstis. A Survey of Mobile Computing Technologies and Applications. Technischer Bericht TR-95-050, Dept. of Computer Sciences, Purdue University, Oktober 1995.

[KAA+99] C.D. Kidd, G.D. Abowd, C.G. Atkeson, I.A. Essa, B. MacIntyre, E. Mynatt und T.E. Starner. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Proceedings of the 2nd International Workshop on Cooperative Buildings (CoBuild '99)*, Pittsburgh, PA USA, Oktober 1999.

[KAC+96] A.M. Keller, T. Ahamad, M. Clary, O. Densmore, S. Gadol, W. Huang, R. Razavi und R. Pang. *Mobile Computing*, Kapitel 24: The Diana Approach to Mobile Computing, Seiten 651–679. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA USA, 1996.

[Kar93] P. Karn. The Qualcomm CDMA digital cellular system. In *Proceedings of the USENIX Mobile and Location–Independent Computing Symposium*, Seiten 35–39, Cambridge, MA USA, August 1993. USENIX.

[Kat94] R.H. Katz. Adaption and Mobility in Wireless Information Systems. *IEEE Personal Communications*, 1(1):6–17, 1994.

[Ker92] H. Kerner. *Rechnernetze nach OSI.* Addison–Wesley, Bonn, Deutschland, 1992.

[Kle95] L. Kleinrock. Nomadic Computing – An Opportunity. *ACM Communication Review*, 25(1):36–40, Januar 1995.

[KLP99] A. Küpper und C. Linnhoff-Popien. Invoking Computational Objects in Mobile Devices. In *Proceedings of the 2nd IFIP W.G. 6.1 International Working Conference on Distributed Applications and Interoperable Systems*, Helsinki, Finnland, Juni 1999. Kluwer Academic Press.

[Kly00] G. Klyne. CC/PP Attribute Vocabularies. W3C Working Draft WD–CCPP–vocab–20000721, W3C, Juli 2000.

[KRR98a] E. Kovacs, K. Röhrle und M. Reich. Integrating an Agent System into the Mobile Middleware. In K. Rothermel und F. Hohl, Hrsg., *Proceedings of the 2nd International Workshop on Mobile Agents (MA '98)*, Lecture Notes in Computer Science No. 1477, Stuttgart, Deutschland, September 1998. Springer Verlag.

[KRR98b] E. Kovacs, K. Röhrle und M. Reich. Mobile Agents OnTheMove – Integrating an Agent System into the Mobile Middleware. In *Proceedings of the ACTS Mobile Summit 1998*, Rhodos, Griechenland, Juni 1998. ACTS.

[KT01] M. Krutwig und R. Tolksdorf. *WML und WMLScript.* xml.bibliothek. dpunkt.verlag, Heidelberg, Deutschland, 2001.

[LAB+00] S. Li, R. Ashri, M. Buurmeijer, E. Hol, B. Flenner, J. Scheuring und A. Schneider. *Professional JINI.* Programmer To Programmer Series. Wrox Press, Birmingham, Großbritannien, 2000.

[Lam01] R. Lamprecht, Hrsg. *Zukunft mobile Kommunikation – Wirklichkeit und Vision einer technischen Revolution.* Frankfurter Allgemeine Buch, Frankfurt, Deutschland, März 2001.

[LBT99] H. Lei, M. Blount und C. Tait. DataX: an Approach to Ubiquitous Database Access. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, (WMCSA '99)*, New Orleans, LA USA, Februar 1999.

[LP97] H. Lei und C. Perkins. Ad–hoc Networking with Mobile IP. In *Proceedings of the 2nd European Personal Mobile Communications Conference 1997*, Seiten 197–202, Bonn, Deutschland, Oktober 1997. ITG.

[LS99] O. Lassila und R.R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation REC–rdf–syntax–19990222, W3C, Februar 1999.

[Maa97] H. Maaß. Location–Aware Mobile Applications based on Directory Services. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom '97)*, Bukarest, Ungarn, September 1997. ACM/IEEE.

[Man98] S. Mann. Wearable Computing as a Means for Personal Empowerment. In *Proceedings of the 1st International Conference on Wearable Computing (ICWC '98)*, Fairfax, VA USA, Mai 1998.

[May94] C. Mayers. A Technical Introduction to ANSA. Folienpräsentation, Mai 1994.

[May96] C. Mayers. ANSA Phase III: Introduction to ANSA. Folienpräsentation im Rahmen der ANSAworks '96 Konferenz, April 1996.

[Mer96] M. Merz. *Elektronische Dienstemärkte: Modelle und Mechanismen zur Unterstützung von Handelstransaktionen in offenen verteilten Systemen.* Dissertation, Universität Hamburg, Fachbereich Informatik, Hamburg, Deutschland, September 1996.

[Mes94] G. Meszaros. *Pattern: Half–Object+ Protocol (HOPP)*, Seiten 129–132. In: Pattern Language of Program Design. Addison–Wesley, 1994.

[MJ96] K. Müller-Jones. *Koordinierte Dienstnutzung in offenen verteilten Dienstemärkten.* Dissertation, Universität Hamburg, FB Informatik, Hamburg, Deutschland, Dezember 1996.

[Mül96] S. Müller. Entwurf eines Kooperationskonzeptes für Dienstvermittlungskomponenten in offenen verteilten Systemen. Diplomarbeit, Universität Hamburg, FB Informatik, Hamburg, Deutschland, Juni 1996.

[MMFR96] M. Mathis, J. Mahdavi, S. Floyd und A. Romanow. TCP Selective Acknowledgement Options. Request for Comment 2018, PSC, LBNL und Sun Microsystems, Oktober 1996.

[MMJLT96] S. Müller, K. Müller-Jones, W. Lamersdorf und T. Tu. Global Trader Cooperation in Open Service Markets. In O. Spaniol, C. Linnhoff-Popien und B. Meyer, Hrsg., *Proceedings of the International Workshop on Trends in Distributed Systems (TreDS '96)*, Lecture Notes in Computer Science No. 1161, Seiten 214–227, Berlin, Deutschland, Oktober 1996. Springer Verlag.

[Mot99] Motorola. VoxML 1.1 Language Reference. Reference Document, Internet and Conectivity Solutions Division, April 1999.

[MP92] M. Mouly und M.-B. Pautet. *The GSM System for Mobile Communications*. Cell&Sys, Palaiseau, Frankreich, 1992.

[MP97] J. Meggers und A. Park. Initial Architecture of MASE v2.0, D26. OnTheMove Public Deliverable AC034/I4/WP2/DR/P/026/b1, RWTH Aachen, i4, Juni 1997.

[MS93] A. Myles und D. Skellern. Comparison of Mobile Host Protocols for IP. *Journal of Internetworking Research and Experience*, 4(4), 1993.

[MSGW98] S. Murer, P. Schnorf, E. Gamma und A. Weinand. *Eine realistische Applikationsarchitektur für Multi–Tier Java–basierte Clients in der Praxis*, Kapitel 9. In: Java in der Praxis. d.Punkt Verlag, Heidelberg, Deutschland, 1998.

[MW00a] S. Müller-Wilken. Gateway to Heaven – WAP Gateways erklärt. *iX – Magazin für professionelle Informationstechnik*, Seiten 136–141, Mai 2000.

[MW00b] S. Müller-Wilken. Momentaufnahme – 8 WAP Terminals im Test. *iX – Magazin für professionelle Informationstechnik*, Seiten 64–70, August 2000.

[MWHL00] S. Müller-Wilken, D. Hinz und W. Lamersdorf. XML and JINI – On using XML and the ‚Java Border Service Architecture‘ to integrate mobile devices into the Java Intelligent Network Infrastructure. In *Proceedings of the 2000 XTech Conference*, San Jose, CA USA, März 2000. Graphic Communications Association.

[MWL00a] S. Müller-Wilken und W. Lamersdorf. Enhancing JINI to Support non–Java Applicances. In *Proceedings of the 4th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL USA, Juli 2000. International Institute of Informatics and Systemics.

[MWL00b] S. Müller-Wilken und W. Lamersdorf. JBSA: An Infrastructure for Seamless Mobile Systems Integration. In C. Linnhoff-Popien und H.-G. Hegering, Hrsg., *Proceedings of the 3rd International IFIP/GI Workshop Conference, Towards a Universal Service Market (USM '2000)*, Lecture Notes in Computer Science No. 1890, Seiten 164–175, Berlin, Deutschland, September 2000. Springer Verlag.

[MWR98] S. Müller-Wilken und F. Ramme. Advanced Service Provisioning in future telecommunication networks – a first pre–study. Interner Bericht, Ericsson Eurolab, Herzogenrath, Deutschland, September 1998.

[MWWL00] S. Müller-Wilken, F. Wienberg und W. Lamersdorf. On integrating mobile devices into a workflow management scenario. In A.M. Tjoa, R.R. Wagner und A. Al-Zobaidie, Hrsg., *Proceedings of the 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, Seiten 186–192, London, Großbritannien, Oktober 2000. IEEE Computer Society.

[Nak96] T. Nakajima. Adaptability and Mobility in Mobile Computing Environment. In *13th Conference Proceedings JSSST*, September 1996.

[NHO00] M. Nilsson, J. Hjelm und H. Ohto. Composite Capabilities/Preference Profiles: Requirements and Architecture. W3C Working Draft WD–CCPP–ra–20000721, W3C, Juli 2000.

[Nil00] M. Nilsson. Composite Capabilities/Preference Profiles: Terminology and Abbreviations. W3C Working Draft WD–CCPP–ta–20000721, W3C, Juli 2000.

[NLL99] H. Frystyk Nielsen, P. Leach und S. Lawrence. HTTP Extension Framework. Internet Draft draft–frystyk–http–extensions–03, W3C, März 1999.

[Nob00] B.D. Noble. System Support for Mobile, Adaptive Applications. *IEEE Personal Communications*, 7(1):44–49, Februar 2000.

[NPS95] B.D. Noble, M. Price und M. Satyanarayanan. A Programming Interface for Application–Aware Adaptation in Mobile Computing. *Computing Systems*, 8(4):345–363, Herbst 1995.

[NSN+97] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn und K.R. Walker. Agile Application–Aware Adaptation for Mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, Saint–Malo, Frankreich, Oktober 1997.

[Nye92] A. Nye. *Xlib Programming Manual.* X Window System Series. O'Reilly & Associates, Sebastopol, CA USA, 3. Auflage, Juli 1992.

[OH99] H. Otho und J. Hjelm. CC/PP exchange protocol based on HTTP Extension Framework. W3C Note NOTE–CCPPexchange–19990624, W3C, April 1999.

[Oli92] Olivetti. Intelligence At Work. The I.T. Manager Collection Vol. IV, Olivetti, DEGW, Northcroft und Ove Arup Partners, 1992.

[Ora00] OracleMobile. OracleMobile Online Studio: A Hosted Solution for Building and Testing Wireless Applications. http://www.oraclemobile.com, Dezember 2000.

[OSI94] ITU–T Rec. X.200 / ISO 7498/1 – OSI Basic Reference Model. International Standards Organisation, 1994.

[OW00] S. Oaks und H. Wong. *JINI in a Nutshell*. O'Reilly & Associates, Sebastopol, CA USA, 1. Auflage, März 2000.

[Pac00] Hewlett Packard. JetSend Getting Started Guide. Folienpräsentation, April 2000.

[PBC+99] A. Paepcke, M. Baldonado, C.-C.K. Chang, S. Cousins und H. Garcia-Molina. Building the InfoBus: A Review of Technical Choices in the Stanford Digital Library Project. *Computer*, 32(2):80–87, Februrar 1999.

[Per96] C.E. Perkins. IP Mobility Support. Request for Comment 2002, IBM, Oktober 1996.

[Per98] C.E. Perkins. *Mobile IP: Design Principles and Practices*. Addison–Wesley Wireless Communication Series. Addison–Wesley Longman, Reading, MA USA, März 1998.

[PGJH90] G.J. Popek, R.G. Guy, T.W. Page Jr. und J.S. Heidemann. Replication in Ficus Distributed File Systems. In *Proceedings of the Workshop on Management of Replicated Data*, Seiten 20–25, Los Angeles, CA USA, April 1990. IEEE.

[PH97] R.W. Picard und J. Healey. Affective Wearables. *Personal Technologies*, 1(1):231–249, 1997.

[PJ96] C.E. Perkins und D.B. Johnson. Mobility Support in IPv6. In *Proceedings of the 2nd Annual International Conference on Mobile Computing and Networking (Mobicom '96)*, White Plains, NY USA, November 1996. ACM.

[PJ00] C.E. Perkins und D.B. Johnson. Mobility Support in IPv6. Internet Draft ipv6–12, IETF Mobile IP Working Group, April 2000.

[PM95] A. Paulraj und T. Moorthi. The Evolution of Mobile Communications. *IETE Technical Review*, 12(5&6):353–358, November/Dezember 1995.

[PR99] M. Panayi und D. Roy. Magic of Today: Tomorrow's Technology – Wearables for Kids. In *Workshop on Situated Interaction in Ubiquitous Computing at CHI 2000*, Den Haag, Niederlande, April 1999.

[PR00] A. Puder und K. Römer. *MICO: An Open Source CORBA Implementation*. dpunkt.verlag, Heidelberg, Deutschland, 3. Auflage, Mai 2000.

[Pro99] The Mozilla Project. Introduction to a XUL Document. http://www.mozilla.org/xpfe/xptoolkit/xulintro.html, Juni 1999.

[PSGP99] T.-L. Pham, G. Schneider, S. Goose und A. Pizano. Composite Device Computing Environment: A Framework for Augmenting the PDA Using Surrounding Resources. In *Workshop on Situated Interaction in Ubiquitous Computing at CHI 2000*, Den Haag, Niederlande, April 1999.

[RAA+98] K. Raatikainen, R. Agrawal, R. Akers, , M. Anagnostou, J. Biswas, Z. Canela, Y. Ismailov, H. Laamanen, M. Larsson, M. Liljeberg, L. Marchand, L. Oberlander, S. Palazzio, R. Radhakrishna, P.S. Rahkila, M. Ranganath, P. Runyan, F. Tam und W. Weiguo. White Paper on Wireless Access and Terminal Mobility in CORBA. OMG White Paper telecom/98-02-09, OMG Telecom Domain Task Force, November 1998.

[RASS97] M. Ranganathan, A. Acharya, S.D. Sharma und J. Saltz. Network–aware Mobile Programs. In *Proceedings of the 1997 USENIX Annual Technical Conference*, Anaheim, CA USA, 1997.

[Rat98] D.H. Ratner. *Roam: A Scalable Replication System for Mobile and Distributed Computing*. Dissertation, UCLA Computer Science Dept., Los Angeles, CA USA, Januar 1998.

[Ray97] D. Raychaudhuri. R&D Challenges in Mobile & Wireless Networking. In *Proceedings of the NSF Wireless and Mobile Communications Workshop*, Airlie, VA USA, März 1997.

[RB97] A. Rakotonirainy und A. Bond. Specification of Mobile Transparency. In J. Rolia, J. Slonim und J. Botsford, Hrsg., *Proceedings of the IFIP/IEEE International Conference on ODP and Distributed Platforms*, Seiten 277–287, Toronto, Kanada, Mai 1997. Chapman & Hall.

[RBMH94] T. Richardson, F. Bennett, G. Mapp und A. Hopper. Teleporting in an X window system environment. *IEEE Personal Communications*, 3(1):6–12, Drittes Quartal 1994.

[RCDD98] T. Rodden, K. Chervest, N. Davies und A. Dix. Exploiting Context in HCI Design for Mobile Systems. In C. Johnson, Hrsg., *Proceedings of the 1st Workshop on Human Computer Interaction with Mobile Devices*, number G98–1 in GIST Technical Report, Glasgow, Schottland, Mai 1998. Dept. of Computing Science, University of Glasgow.

[RDLH99] J. Redström, P. Dahlberg, P. Ljungstrand und L.E. Holmquist. Designing for Local Interaction. In *Proceedings of the 1st International Workshop on Managing Interaction in Smart Environments (MANSE '99)*, Dublin, Irland, Dezember 1999. Springer Verlag.

[RF99] K. Ramakrishnan und S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. Request for Comment 2481, The Internet Society, Januar 1999.

[RHDS99] F. Reynolds, J. Hjelm, S. Dawkins und S. Singhal. Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation. W3C Note NOTE–CCPP–19990727, W3C, Juli 1999.

[RKF93] W. Rosenberry, D. Kennedy und G. Fischer. *Understanding DCE*. O'Reilly & Associates, Sebastopol, CA USA, 2. Auflage, 1993.

[RMW99] B.J. Rhodes, N. Minar und J. Weaver. Wearable Computing Meets Ubiquitous Computing: Reaping the best of both worlds. In *Proceedings of The 3rd International Symposium on Wearable Computers (ISWC '99)*, San Francisco, CA USA, Oktober 1999.

[RSC+99] M. Roman, A. Singhai, D. Carvalho, C. Hess und R.H. Campbell. Integrating PDAs into Distributed Systems: 2k and PalmORB. In H.-W. Gellersen, Hrsg., *Handheld and Ubiquitous Computing – 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*, Lecture Notes in Computer Science No. 1707, Heidelberg, Deutschland, September 1999. Springer Verlag.

[RSFWH98] T. Richardson, Q. Stafford-Fraser, K.R. Wood und A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, Januar/Februar 1998.

[Sat95] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *Proceedings of the 14th ACM SIGACT–SIGOPS Symposium on Principles of Distributed Computing (PODC '95)*, Ottawa, Kanada, August 1995. ACM.

[Sch93] A. Schill. *DCE: Das OSF Distributed Computing Environment*. Springer Verlag, 1993.

[Sch00] J. Schiller. *Mobilkommunikation – Techniken für das allgegenwärtige Internet*. Networking and Communication Series. Addison–Wesley, München, Deutschland, 2000.

[Sie96] J. Siegel, Hrsg. *CORBA Fundamentals and Programming*. John Wiley & Sons, New York, NY USA, 1996.

[SJ98] F. Stajano und A. Jones. The Thinnest of Clients: Controlling it all via Cellphone. *ACM Mobile Computing and Communications Review*, 2(4), Oktober 1998.

[SKM+93] M. Satyanarayanan, J.J. Kistler, L.B. Mummert, M.R. Ebling, P. Kumar und Q. Lu. Experience with Disconnected Operation in a Mobile Computing Environment. In *USENIX Mobile and Location–Independent Computing Symposium*, Seiten 11–28, Cambridge, MA USA, August 1993. Carnegie Mellon University.

[SM99] J.T. Sinclair und M. Merkow. *Thin Clients Clearly Explained*. Academic Press, 1. Auflage, 1999.

[SMR+97] T. Starner, S. Mann, B. Rhodes, J. Levine, J. Healey, D. Kirsch, R.W. Picard und A. Pentland. Augmented Reality Through Wearable Computing. M.I.T. Media Laboratory Perceptional Computing Section / Technical Report 397, Massachusetts Institute of Technology, Cambridge, MA USA, 1997.

[SNKP94] M. Satyanarayanan, B. Noble, P. Kumar und M. Price. Application–Aware Adaptation for Mobile Computing. In *Proceedings of the 6th ACM SIGOPS European Workshop*, Dagstuhl, Deutschland, September 1994. Carnegie Mellon University.

[SRBW99] M. Schläger, B. Rathke, S. Bodenstein und A. Wolisz. Advocating a Remote Socket Architecture for Internet Access using Wireless LANs. *Mobile Networks and Applications, Special Issue on Wireless Internet and Intranet Access*, 1999.

[SRC81] J.H. Saltzer, D.P. Reed und D.D. Clark. End–To–End Arguments in System Design. In *Proceedings of the 2nd International Conference on Distributed Computing Systems*, Seiten 509–512, Paris, Frankreich, April 1981.

[SRC84] J.H. Saltzer, D.P. Reed und D.D. Clark. End–To–End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[SRC98] J.H. Saltzer, D.P. Reed und D.D. Clark. Commentaries on Active Networking and End–To–End Arguments. *IEEE Network*, 12(3):66–71, Mai/Juni 1998.

[SS98] N. Sawhney und C. Schmandt. Speaking and Listening on the Run: Design for Wearable Audio Computing. In *Proceedings of ISWC '98, International Symposium on Wearable Computing*, Pittsburg, PA USA, Oktober 1998. IEEE.

[SS99] A. Smailagic und D. Siewiorek. User–Centered Interdisciplinary Concurrent System Design. *ACM Mobile Computing and Communications Review*, 3(3):43–52, 1999.

[SSR+99] B. Schiele, T. Starner, B. Rhodes, B. Clarkson und A. Pentland. *Situation Aware Computing with Wearable Computers*, Kapitel 21. Lawrence Erlbaum Press, Mahwah, NJ USA, 1999.

[Sta87] W. Stallings. *Handbook of Computer Communications Standards*. Number 2: Local Network Standards in The William Stallings Series of Books on Computer and Data Communications Technology. Howard W. Sams & Company, Indianapolis, IN USA, 1987.

[Sta99] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison–Wesley, 3. Auflage, 1999.

[Ste90] W.R. Stevens. *UNIX Network Programming*. Prentice Hall Software Series. Prentice Hall PTR, 1990.

[Ste92] W.R. Stevens. *Advanced Programming in the UNIX Environment*. Addison–Wesley Professional Computing Series. Addison–Wesley, Juni 1992.

[Ste94] W.R. Stevens. *TCP/IP Illustrated*. Number 1: The Protocols in Addison–Wesley Professional Computing Series. Addison–Wesley, 1994.

[Sun99a] Sun Microsystems. JavaSpaces Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun99b] Sun Microsystems. JINI Architecture Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun99c] Sun Microsystems. JINI Device Architecture Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun99d] Sun Microsystems. JINI Discovery and Join Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun99e] Sun Microsystems. JINI Discovery Utilities Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun99f] Sun Microsystems. JINI Lookup Service Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 1999.

[Sun00] Sun Microsystems. Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices. KVM White Paper, SUN Microsystems Inc., Palo Alto, CA USA, 2000.

[SW94] W.R. Smith und R.V. Welland. A Model for Address–Oriented Software and Hardware. In *Proceedings of the 25th Hawaii International Conference on System Sciences (HICSS '25), Vol.1*, Seiten 720–729, San Francisco, CA USA, 1994. IEEE Computer Society Press.

[Tan96] A.S. Tanenbaum. *Computer Networks*. Prentice–Hall International Inc., Upper Saddle River, NJ USA, 3. Auflage, 1996.

[Tec97] IONA Technologies. IONA Technologies IIOP Engine Establishes True Consumer Device Connectivity. Pressemitteilung, http://www.iona.com/Press/Press/iiopengine.html, März 1997.

[Tho01] K.B. Thompson. JINI Surrogate Architecture Specification. Technischer Bericht, Sun Microsystems Inc., Palo Alto, CA USA, 2001.

[TYT91] F. Teraoka, Y. Yokote und M. Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of the Conference on Communications Architecture & Protocols (SIGCOMM '91)*, Seiten 209–220, Zürich, Schweiz, September 1991. ACM.

[WB97] G. Welling und B.R. Badrinath. A Framework for Environment Aware Mobile Applications. In *Proceedings of the 17th International Conference on Distributed Computing Systems (ICDCS '97)*, Seiten 384–391, Maryland, MA USA, Mai 1997.

[WBWW90] R. Wirfs-Brock, B. Wilkerson und L. Wiener. *Designing Object–Oriented Software*. Prentice Hall, 1990.

[Web01] M. Weber. *Verteilte Systeme*. Spektrum–Hochschultaschenbuch. Spektrum Akademischer Verlag, Heidelberg, Deutschland, März 2001.

[Wei91] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[Wei93] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):74–83, Juli 1993.

[Wie01] F. Wienberg. *Informations– und prozessorientierte Modellierung verteilter Systeme auf der Basis von Feature–Structure–Netzen*. Dissertation, Universität Hamburg, Fachbereich Informatik, Hamburg, Deutschland, Mai 2001.

[Wol99] A. Wolisz. Mobility in Multimedia Communication. In *Proceedings of the 2nd WAKI–Symposium „Verteilte multimediale Anwendungen und diensteintegrierende Kommunikationsnetze“*, Flensburg, Deutschland, September 1999.

[WRB+96] K.R. Wood, T. Richardson, F. Bennet, A. Harter und A. Hopper. Global Teleporting with Java: Towards ubiquitous personalised computing. Technischer Bericht 96.2, Olivetti Research Laboratories, Cambridge, Großbritannien, März 1996.

[WSA+95] R. Want, B.N. Schilit, A. Adams, R. Gold, K. Petersen, D. Goldberg, J. Ellis und M. Weiser. The ParcTab Ubiquitous Computing Experiment. Technischer Bericht CSL–95–1, Xerox Palo Alto Research Center, Palo Alto, CA USA, März 1995.

[WSA+96] R. Want, B.N. Schilit, N.I. Adams, R. Gold, K. Petersen, D. Goldberg, J.R. Ellis und M. Weiser. *Mobile Computing*, Kapitel 2, Seiten 45–101. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA USA, 1996.

[WSW+94] R. Welland, G. Seitz, L.-W. Wang, L. Dyer, T. Harrington und D. Culbert. The Newton Operating System. In *Proceedings of the 1994 IEEE Computer Conference*, San Francisco, CA USA, 1994. IEEE Computer Society Press.

[WWDS96] M. Weiser, B. Welch, A. Demers und S. Shenker. *Mobile Computing*, Kapitel 17, Seiten 449–470. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA USA, 1996.

[XP99] G. Xylomenos und G.C. Polyzos. Link Layer Support for Quality of Service on Wireless Internet Links. *IEEE Personal Communications*, 6(5):52–60, 1999.

[You90] D.A. Young. *The X Window System: Programming and Applications with Xt, OSF/Motif Edition.* Prentice Hall, 1990.

[You95] D.A. Young. *Object–Oriented Programming With C++ and OSF/Motif.* Prentice Hall PTR, Upper Saddle River, NJ USA, 2. Auflage, 1995.

[Zor00] M. Zorzi. *The Communications Handbook*, Kapitel 2.4 Mobile and Wireless Telecommunications Networks, Seiten 2–40 – 2–65. CRC Press LLC., 2000.

Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorstehende Arbeit eigenständig und ohne fremde Hilfe angefertigt und mich dabei ausschließlich der angegebenen Quellen und Hilfsmittel bedient habe.

Lütjensee, den 12. April 2002.

Stefan Müller–Wilken