Strukturierung petrinetzbasierter Multiagentenanwendungen am Beispiel verteilter Softwareentwicklungsprozesse

Dissertation

zur Erlangung des akademischen Grades
Dr. rer. nat
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

eingereicht beim Fachbereich Informatik von

Kolja Markwardt

aus Hamburg

Januar 2012

Betreuer der Arbeit:	Dr. Daniel Moldt Prof. Dr. Rüdiger Valk
Gutachter:	Prof. Dr. Horst Oberquelle
Vorsitz Promotionsprüfungsausschuss:	Prof. Dr. Bernd Wolfinger
Datum der Disputation:	4. September 2012

Danksagung

Ich danke an dieser Stelle den Leuten, die mich im Verlauf der Entstehung dieser Arbeit unterstützt haben und ohne die diese Arbeit nicht möglich gewesen wäre. Zunächst sind dies Dr. Daniel Moldt und Professor Dr. Rüdiger Valk, die die Arbeit betreut haben, sowie Professor Dr. Horst Oberquelle für die Begutachtung und Professor Dr. Bernd Wolfinger als Vorsitzender des Promotionsprüfungsausschusses.

Desweiteren gilt mein Dank meiner Familie, die die ganze Zeit über für mich da war, insbesondere Vanessa für die stetige Ermutigung und Unterstützung, meinem Vater für gute Ratschläge und Finian und Thea dafür, dass sie da waren und mir die Kraft gegeben haben, weiterzumachen.

Ich danke allen Mitgliedern des Arbeitsbereichs TGI für viele wertvolle und anregende Diskussionen, Zusammenarbeit und Hilfen, wann immer ich sie benötigt habe. Ohne die fruchtbaren Gespräche mit Christine, Lawrence, Michael, Thomas, Matthias, Sven, Jan und allen anderen wäre diese Arbeit nicht möglich gewesen.

Aber insbesondere die teilweise recht intensiven Diskussionen mit Daniel waren es, die mich immer und immer wieder verwirrt und durcheinander zurückgelassen haben, die aber am Ende diese Arbeit mehr geformt haben als alles andere. Ohne diese stundenlangen Gespräche wäre die Arbeit nicht das geworden, was sie ist. Daniel hat mir immer wieder gezeigt, was denkbar ist hinter dem, was ich bisher gedacht habe.

Und schließlich möchte ich mich bei meinen Freunden bedanken, die mich die ganzen Jahre über ausgehalten haben, aber insbesondere Verena, die mich oft ermutigt und aufgebaut hat und mir geholfen hat, das Ziel weiterhin zu sehen und nicht aufzugeben, sowie meinen Kollegen außerhalb der Uni. Während der Jahre bei S&R, Azri, Bitmanager und Kühne+Nagel habe ich vieles gelernt und viele wertvolle Erfahrungen gesammelt, von denen sich so manche auch in dieser Arbeit niedergeschlagen hat.

Zusammenfassung

Die Entwicklung komplexer, verteilter Anwendungen in heterogenen Umgebungen ist eine zentrale Herausforderung der Informatik. Für die Modellierung und Implementierung nebenläufiger, verteilter Anwendungen bieten Petrinetze eine gute Kombination aus intuitiver Darstellung und klarer Semantik. Um die Beherrschbarkeit von Netzmodellen für große Systeme zu gewährleisten, ist eine geeignete Strukturierung der Modelle erforderlich.

Dafür hat sich die agentenorientierte Softwareentwicklung als erfolgreiches Paradigma erwiesen. Multiagentensysteme bieten die Möglichkeit, individuell handelnde Akteure zu modellieren, die in einem verteilten System interagieren. Beim Entwurf agentenorientierter Software ist die Frage zu beantworten, aus welchen Arten von Agenten eine Anwendung besteht und wie diese miteinander in Beziehung stehen. Als Beitrag zur Softwareentwicklung mit Petrinetzen werden daher in dieser Arbeit Strukturierungsmöglichkeiten für petrinetzbasierte Multiagentenanwendungen untersucht. Struktur und Prozess werden als orthogonale Dimensionen des Systementwurfs betrachtet und mit der agentenorientierten Softwareentwicklung verbunden. Für die Strukturdimension wird in dieser Arbeit das HeRA-System (HElper and Resource Agents) entworfen, das eine Modellierung der Gegenstände und Funktionen des Anwendungsbereichs ermöglicht. HeRA erlaubt die Definition von Hilfsmitteln und Ressourcen, mit denen die Arbeitsplätze von Benutzern des Systems flexibel konfiguriert werden können. Durch die Aufteilung in untereinander verbundene Agentenplattformen wird die Struktur des Gesamtsystems definiert. Die Prozessinfrastruktur für Agentenanwendungen (Pia) unterstützt die prozessorientierte Sichtweise. Mit diesem agentenorientierten Workflow Management System können die Abläufe in Multiagentensystemen modelliert und ihre Ausführung gesteuert werden. Damit wird es möglich, die Abläufe in einer Organisation und zwischen verschiedenen, miteinander interagierenden Organisationen zu formalisieren und zu unterstützen. In dieser Arbeit werden gezielt Erweiterungen an Pia vorgenommen, um eine Verbindung mit dem HeRA-System zu erreichen.

Mit dem Potato-System (Process-Oriented Tool Agents for Team Organization) werden die beiden Strukturierungsdimensionen zu einem neuen System verbunden. Dieses konzeptionelle Framework bietet die Möglichkeit, sowohl eine gegenstandsorientierte, als auch eine prozessorientierte Sichtweise für die Entwicklung von Multiagentenanwendungen in einem gemeinsamen System zu vereinen.

In dieser Arbeit wird als motivierendes Beispiel die Unterstützung verteilter Softwareentwicklung betrachtet. Potato bietet Ansätze für den Entwurf einer Entwicklungsumgebung zur Unterstützung geographisch verteilter Akteure bei der Softwareentwicklung. Die Anforderungen an eine solche verteilte Entwicklungsumgebung werden aufgestellt. Das Potato-System und seine Bestandteile werden mit Hilfe des Paose-Ansatzes (Petrinetz-, Agenten- und Objektorientierte Software-Entwicklung) entworfen. Durch Prototypen wird die Anwendbarkeit für die Softwareentwicklung gezeigt.

Abstract

The development of complex, distributed applications in heterogenous environments is an important challenge of computer science. Petri nets offer a good combination of intuitive modeling and clear semantics for the modeling and implementation of distributed, concurrent applications. To manage net models for large-scale systems however, it is important to find a suitable structuring of the models involved. Agent-oriented software engineering has been proven to be a successful paradigm of software engineering in this context. Multiagent systems allow the modeling of autonomous actors interacting in a larger system context.

For the design of agent-oriented applications a central question is the choice of agent types and their interaction patterns. Within the field of Petri net-based software engineering, further means of structuring multiagent applications are examined in this work. The two orthogonal dimensions of structure and process are considered for system design in general and agent-oriented software engineering in particular.

The structural dimension is addressed in this work with the HeRA-System (HElper and Resource Agents), which models functions and resources of the application domain as agents. HeRA allows the definition of helper and resource agents that populate the workspace of a user and enable him to flexibly assemble the functionality he needs. Using interconnected agent platforms in a distributed multiagent system serves to define the system structure as a whole.

Pia (Process Infrastructure for Agents) supports the process-oriented view. Processes in multiagent systems can be modelled and controlled in this agent-oriented workflow management system, in order to formalize and support the execution of processes within organizations as well as between different, cooperating organizations. In this work additional extensions to Pia are implemented in order to facilitate an integration with the HeRA-system.

Both these structural dimensions are combined within the Potato-system (Process-Oriented Tool Agents for Team Organization). As a conceptional framework Potato offers an integrated approach to designing multiagent applications using an artifact-based approach together with a process-oriented one.

The motivation and example used in this work is an application for the support of distributed software development. Potato offers concepts for the design of an integrated development environment to support the collaborative work of geographically distributed actors. Requirements for such a distributed development environment are developed and an agent-oriented design for its implementation is created using the Paose (Petri net and Agent-Oriented Software Engineering) approach. The applicability is shown through a set of prototypes.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die weltweite Zusammenarbeit verschiedener Unternehmen untereinander wird immer wichtiger für die effiziente Abwicklung aller Geschäftsprozesse. Das hat zur Folge, dass zunehmend auch die IT-Infrastrukturen stärker zusammenwachsen müssen. Geschäftsprozesse zwischen Unternehmen werden über Serviceorientierte Architekturen und Workflow Management Systeme (WfMS) miteinander verzahnt. Zusätzlich zu einer funktionsorientierten Betrachtung wird die Unterstützung der einzelnen Akteure innerhalb der Wertschöpfungskette und deren Interaktionen durch verteilte Softwaresysteme immer wichtiger. Diese Systeme stellen neue Anforderungen an die Prozesse der Softwareerstellung.

An der Entstehung, dem Betrieb und der weiteren Entwicklung einer Software sind in der Regel eine Reihe verschiedener Organisationen oder Organisationseinheiten beteiligt, die sich auf verschiedene Arten koordinieren müssen, um gemeinsam eine Software bereitstellen zu können. Teile der Entwicklungsarbeit werden zum Beispiel in andere Länder ausgelagert, in denen Arbeitskraft kostengünstiger ist. Durch Outsourcing und Off-/Nearshoring können sich allerdings auch Probleme ergeben, etwa durch erhöhten Koordinations- und Steuerungsaufwand oder durch Missverständnisse aufgrund von Kommunikationsdefiziten.

Ausgehend von diesen Überlegungen untersucht diese Arbeit Möglichkeiten, die Unterstützung verteilter Softwareentwicklungsprozesse zu verbessern. Am Arbeitsbereich „Theoretische Grundlagen der Informatik“ der Universität Hamburg werden Verfahren der Softwareentwicklung mit Petrinetzen untersucht. Vergangene Arbeiten haben Petrinetze erfolgreich für die agentenorientierte Softwareentwicklung eingesetzt.

Petrinetze sind sehr gut dazu geeignet, verteilte, nebenläufige Systeme zu modellieren. Die Verwendung von Referenznetzen bietet zusätzlich einen sehr ausdrucksstarken Formalismus, der von der Modellierung bis zur Implementation verwendet werden kann.

Für die Unterstützung verteilt und autonom arbeitender Akteure wird die Verwendung von agentenorientierter Software vorgeschlagen. Die Agentenori-

entierung ist geeignet, verteilte Systeme aus miteinander interagierenden, autonomen Akteuren abzubilden. Zusätzlich zeichnen sich Multiagentensysteme durch ein hohes Maß an Flexibilität und Adaptivität gegenüber Veränderungen aus.

Im PAOSE-Ansatz[1] war aber bislang die Frage, welche Arten von Agenten in einer Agentenanwendung verwendet werden, und wie die Interaktionen zwischen diesen Arten implementiert werden, noch nicht behandelt worden. Die vorliegende Arbeit baut auf den bisherigen Arbeiten auf und untersucht Muster für das Zusammenspiel der Agenten in Anwendungen, um Multiagentensysteme effektiver für die Konstruktion von Unterstützungssystemen nutzen zu können.

Um Multiagentensysteme für die Entwicklung konkreter Anwendungen zu verwenden, werden in dieser Arbeit zwei verschiedene, orthogonale Ansätze für die Struktur von Multiagentenanwendungen vorgestellt und miteinander verbunden. Zum einen wird eine anwendungsorientierte Sichtweise verwendet, die Gegenstände des Anwendungsbereiches als Agenten modelliert und die vom Autor unter anderem in [LEHMANN und MARKWARDT, 2004, MARKWARDT und MOLDT, 2010] beschrieben wird. Zum anderen wird eine Prozessorientierung angestrebt, die sich mit den Abläufen innerhalb des Systems befasst, und diese in der Art eines Workflow Management Systems organisiert [REESE et al., 2005, REESE, 2010]. Zudem wird die Ebene der Projektorganisation in der agentenorientierten Softwareentwicklung betrachtet. Hierfür wird der oben genannte PAOSE-Ansatz [CABAC, 2010] verwendet.

Bei der Entwicklung von Multiagentensystemen erfolgte in der Vergangenheit meist eine prozessorientierte übergeordnete Strukturierung des Gesamtsystems. Dabei waren aber keine detaillierteren Muster für die Auswahl und die Ausrichtung der Prozesse und der im System vorkommenden Agenten vorhanden, an denen sich die Entwicklung orientieren konnte. Ein Ansatz, der in dieser Arbeit verfolgt wird, ist das HERA-System (HElper and Resource Agents), das Helfer und Ressourcen als spezielle Agententypen zur Systemstrukturierung etabliert.

Einen prozessorientierten Ansatz liefert [REESE, 2010] mit einer Prozessinfrastruktur für Agentenanwendungen (PIA). Reese fordert zwar eine Weiterentwicklung der Prozesssicht hin zu einer abstrakteren Sicht, die Agenten und Prozesse vereinheitlicht, es fehlt aber die differenzierte Ausgestaltung der Einheiten ihrer letzten Entwicklungsstufe. In dieser Arbeit wird konzeptionell eine gezielte Erweiterung der vorgenannten Arbeiten vorgestellt, indem die Ansätze von HERA und PIA im POTATO-System (Process-Oriented Tool Agents for Team Organization) vereint werden.

In vergangenen Lehreprojekten wurde eine agentenorientierte Projektorganisation erfolgreich eingesetzt, um agentenorientierte Software zu entwickeln. Die Studenten wurden dabei als „Agenten“ aufgefasst, die jeweils einen be-

[1] Petrinetz-, Agenten- und Objektorientierte Software-Entwicklung

stimmten Verantwortungsbereich in der entwickelten Software übernommen haben, also beispielsweise eine bestimmte Agentenrolle oder eine Interaktion. Bei der Entwicklung mussten sie dann selbstständig mit anderen Studenten interagieren, um auftretende Konflikte aufzulösen (vgl. [Moldt, 2005]). Aufgrund der Größe und der organisatorisch bedingten Verteilung der bis zu 50 beteiligten Personen lag ein gutes Fallbeispiel eines verteilten Softwareentwicklungsprozesses vor.

Die Vision und das Anwendungsbeispiel in dieser Arbeit ist die Erstellung einer verteilten Entwicklungsumgebung (vIDE), die verschiedene Parteien bei der gemeinsamen Arbeit an einem Softwareprojekt unterstützen soll. Für diese vIDE werden Eigenschaften und Anforderungen aufgestellt und ein Vorschlag entwickelt, wie eine solche Umgebung softwaretechnisch umgesetzt werden kann. Im Rahmen der Arbeit wird die Umsetzung prototypisch gezeigt.

Kapitel 2

Motivation

Die Motivation für diese Arbeit ergibt sich unter anderem durch die persönliche Erfahrung in einem internationalen Softwareprojekt. Der Autor hat in Indien die Entwicklung einer webbasierten Anwendung für eine weltweit operierende Organisation geleitet. Dabei war die Abstimmung mit weiteren, international verteilten Projektteilnehmern erforderlich. Im Laufe des Projektes wurden vier verschiedene Betrachtungsebenen deutlich, die für die Modellierung von Entwicklungsprozessen für Softwaresysteme relevant sind.

Die erste Ebene bildet die verteilte Organisation selbst. Hier sind verschiedene Akteure, Funktionen und Prozesse zu finden, die durch Software unterstützt werden müssen. Auf der zweiten Ebene findet man die Anwendungssoftware, die die Vorgänge in der Organisation unterstützt und die an den in der Organisation vorgefundenen Strukturen ausgerichtet wird.

Als dritte Ebene lässt sich der Prozess der Entwicklung dieser Anwendungssoftware und die damit befasste Entwicklerorganisation definieren. Für diese Organisation kann es sich als sinnvoll erweisen, sich wiederum an den Strukturen zu orientieren, die in der Software vorgefunden werden. Als vierte Ebene wird nun die Softwareentwicklung wiederum als neuer Anwendungskontext betrachtet. Die Unterstützung der Entwicklungsprozesse folgt dann wiederum der Struktur der Entwicklerorganisation und auch die verwendete Softwareentwicklungsumgebung wird an die bei der Entwicklerorganisation vorgefundenen Strukturen angepasst. Dieses Ziel der Harmonisierung von Organisationen und ihren Softwaresystemen wird auch als Business/IT-Alignment bezeichnet [Baumöl, 2006, S. 314] und ausführlicher in Abschnitt 2.1 diskutiert.

Im Folgenden wird zunächst der Begriff des Business/IT-Alignment näher ausgeführt und mit der verteilten Entwicklungsumgebung in Zusammenhang gesetzt. Anschließend wird skizziert, was im weiteren Verlauf der Arbeit unter einer verteilten Entwicklungsumgebung verstanden wird, und wie diese die Arbeit in verteilten Softwareprojekten unterstützen kann. Es wird ein Überblick über die verschiedenen softwaretechnischen Dimensionen gegeben, die die Architektur des Potato-Systems beeinflussen. Schließlich folgt eine Übersicht über das weitere Vorgehen der Arbeit.

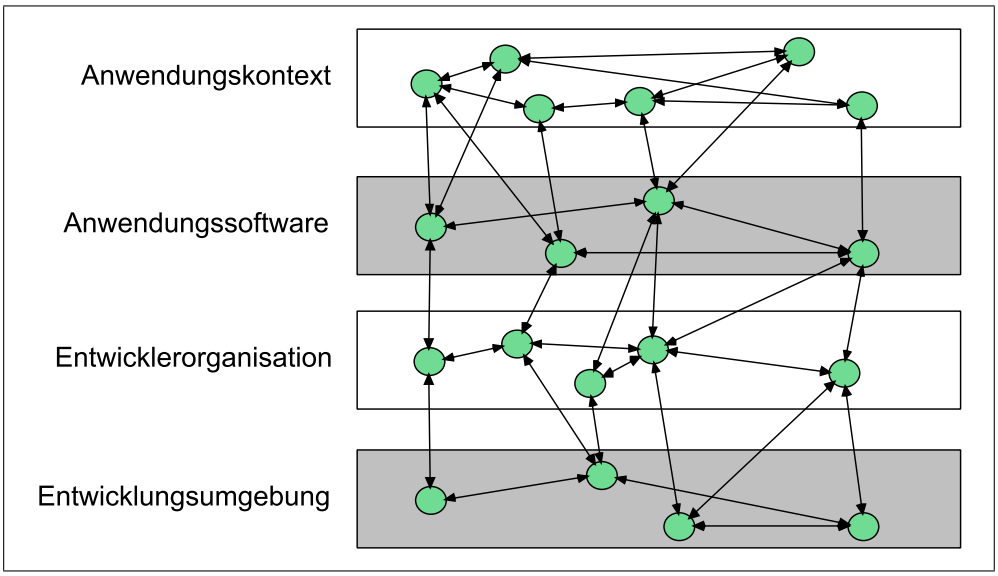

Abbildung 2.1: Ebenen des Business/IT-Alignment

2.1 Business/IT-Alignment

Ein wichtiges Ziel beim Entwurf von IT-Infrastrukturen in Organisationen ist es, die Softwarelandschaften mit den Zielen der Organisation in Einklang zu bringen. Das wird auch als Business/IT-Alignment bezeichnet [Baumöl, 2006, S. 314]. „Die Gestaltungsorientierung spielt sowohl für die unternehmerische Praxis als auch für die GPM-Forschung eine zentrale Rolle. In beiden Zusammenhängen sind das Verstehen und die optimale Gestaltung des Zusammenspiels zwischen Organisationsstrukturen und Informationssystemen von großer Bedeutung.“[Houy et al., 2011] Dafür ist es von Vorteil, wenn Softwareanwendungen in ihrer Struktur an die Gegebenheiten der Anwendungsdomäne angepasst sind. Dadurch können Änderungen im Anwendungskontext leichter in die Software übernommen werden.

Abbildung 2.1 zeigt die verschiedenen Ebenen des Business/IT-Alignment, die in dieser Arbeit betrachtet werden. Im oberen Bereich ist der Anwendungskontext einer Organisation dargestellt. Dieser besteht aus verschiedenen, miteinander in Beziehung stehenden Systemkomponenten. Im Falle einer verteilten Organisation sind dies beispielsweise verschiedene Unterorganisationen, die miteinander vernetzt sind und über verschiedene Beziehungsmuster miteinander interagieren.

Die zweite Ebene stellt die Anwendungssoftware dar, die zur Unterstützung des Anwendungskontextes verwendet wird. Die verschiedenen Bestandteile der Software korrespondieren mit bestimmten Elementen des Anwendungskontextes. Dies ist keine 1:1-Abbildung, dennoch wird eine hohe Ähnlichkeit der Struktur angestrebt. In dieser Arbeit wird insbesondere für Organisationen, in denen verteilt agierende Akteure miteinander kooperieren müssen, um ge-

meinsame Ziele zu erreichen und die ein hohes Maß an Flexibilität und Adaptivität erfordern, eine agentenorientierte Struktur vorgeschlagen. Die Akteure des Anwendungskontextes werden durch Softwareagenten abgebildet, die Orte, an denen sich diese befinden, durch Agentenplattformen. Dadurch lassen sich die Interaktionen der Akteure durch Agenteninteraktionen in einem Multiagentensystem abbilden.

Die dritte Ebene stellt die Entwicklerorganisation dar, die für die Entwicklung der Anwendungssoftware verantwortlich ist. Hier kann es ebenfalls sinnvoll sein, wenn die Strukturen einer Software sich in der Struktur, die zu ihrer Erstellung verwendet wird, widerspiegeln. Mit anderen Worten, die Organisation der Softwareentwicklung orientiert sich an der Struktur der zu entwickelnden Software.

Wird also eine agentenorientierte Software entwickelt, liegt es nahe, auch das Entwicklungsteam entlang der identifizierten agentenorientierten Konzepte auszurichten. Die Aufteilung der Arbeit erfolgt dann anhand der verschiedenen Agenten/Rollen bzw. anhand der identifizierten Interaktionen. Diese Art der Organisation hat sich in verschiedenen Lehreprojekten als sinnvoll erwiesen und bildet auch die Grundlage der Team Organisation im PAOSE-Ansatz (siehe Abschnitt 3.3).

In der Abbildung findet sich daher eine dritte Ebene, die die Entwicklerorganisation zeigt, die sich wiederum an den Einheiten in der Anwendungssoftware orientiert. Dies bedeutet nun nicht, dass ein Entwickler direkt für ein bestimmtes Element innerhalb des Anwendungskontextes verantwortlich ist. Vielmehr werden hier ähnliche Strukturen vorgefunden, die jeweils eine agentenorientierte Modellierung nahelegen.

Als vierte Ebene wird nun die Entwicklungsumgebung betrachtet. Die Entwicklerorganisation bildet hier wiederum den Anwendungskontext, für den eine geeignete Softwareunterstützung gefordert wird. Auch für die Software zur Unterstützung verteilter Softwareentwicklung bedeutet das, dass auch und gerade die Software, die die (agentenorientierte, verteilte) Softwareentwicklung unterstützt, ebenfalls verteilt und agentenorientiert implementiert werden sollte. Auf diese Weise ergibt sich eine Hierarchie der verschiedenen Ebenen und ihrer Abhängigkeiten:

- Der Anwendungskontext lässt sich betrachten als Agenten und ihre Interaktionen.
- Die Anwendungssoftware unterstützt den Anwendungskontext durch agentenorientierte Strukturen.
- Die Entwicklung der Anwendungssoftware erfolgt durch ein agentenorientiert strukturiertes Entwicklungsteam.
- Die Entwicklung wird durch eine agentenorientierte Entwicklungsumgebung unterstützt.

Üblicherweise werden bei der Untersuchung des Business/IT-Alignments nur die Beziehungen zwischen einem Unternehmen und seiner IT Infrastruktur untersucht. Die Betrachtung wird hier nun auf die Ebenen der Softwareentwicklung für eine Organisation und die dafür wiederum eingesetzte Software ausgeweitet. Zusätzlich wird statt der unternehmenszentrierten Sichtweise eine Fokussierung auf die Anwendung vorgenommen. In dieser Arbeit wird daher statt von Business/IT-Alignment von Application/IT-Alignment oder Ausrichtung von Anwendung und Informationstechnologie gesprochen. Zur Unterstützung des Application/IT-Alignment wird eine verteilte Entwicklungsumgebung vorgeschlagen, die im Folgenden näher beschrieben wird.

2.2 Verteilte Softwareentwicklungsumgebung

Organisationsübergreifende Zusammenarbeit ist ein bestimmendes Thema der heutigen Arbeitswelt. Ein Beispiel hierfür ist das oben erwähnte Projekt, bei dem der Autor ein Entwicklungsteam in Indien geleitet hat. Anforderungen wurden mit dem Auftraggeber in den Niederlanden und Deutschland abgestimmt, technische Fragen mit amerikanischen und osteuropäischen Kollegen geklärt und in Indien umgesetzt.

Für diese Art verteilter Projekte ist es wichtig, softwaretechnische Lösungen anzubieten, die die Zusammenarbeit effektiv unterstützen können. Softwareentwicklung ist oft ein Prozess, der sich im Zusammenspiel zwischen verschiedenen Organisationen abspielt. In größeren, weltweit tätigen Organisationen können dies zum Teil auch verschiedene Organisationseinheiten der gleichen Organisation sein. Die verschiedenen Parteien, die an diesem Prozess beteiligt sind, benötigen Software zur Unterstützung ihrer Zusammenarbeit. Anhand typischer Benutzungsszenarien wird im Folgenden verdeutlicht, wie die verschiedenen beteiligten Parteien von einem solchen System profitieren können.

Das Anwendungsbeispiel dieser Arbeit ist die Erstellung einer verteilten Softwareentwicklungsumgebung. Dieser Abschnitt erklärt, was unter einer solchen Umgebung zu verstehen ist, indem die Benutzung aus der Sicht verschiedener Prozessbeteiligter beschrieben wird. Damit wird verdeutlicht, welche Vorteile durch eine solche Umgebung realisiert werden können und welche Art von Funktionen im Fokus der Aufmerksamkeit stehen.

Im Rahmen dieser Arbeit wird die konkrete Entwicklungsumgebung sowohl als Ziel verwendet, als auch als Kontext, in dem Beispielagenten implementiert werden. Im Folgenden wird beschrieben, auf welche Art verschiedene an der Entwicklung beteiligte Funktionsbereiche durch die vorgesehene verteilte IDE unterstützen werden können. Teile dieser IDE werden im Rahmen der Arbeit prototypisch implementiert.

2.2.1 Unterstützung des Projektmanagements

Eine ideale verteilte IDE unterstützt die Projektorganisation verteilter Softwareprojekte auf zwei Ebenen. Die Aufbauorganisation wird durch die Verbindung der beteiligten Parteien in einem gemeinsamen System und Strukturierung in geeignete Plattformen unterstützt, die Ablauforganisation durch eine Prozessinfrastruktur, die die Abläufe im Projekt steuert.

Die Funktion des Projektmanagement kann durch dedizierte Projektmanager erbracht werden, zum Teil kann das System aber bereits so gestaltet werden, dass die Systemteilnehmer sich in bestimmten Bereichen selbst managen, indem sie beispielsweise direkt über bestimmte Aspekte verhandeln. Im Folgenden wird zum Teil von Projektmanagern geredet, wenn die Rolle gemeint ist.

Die verschiedenen, an einem Projekt beteiligten Personen, Organisationen und Organisationseinheiten werden durch die verteilte IDE in einem gemeinsamen System zusammengebracht, in dem sie miteinander interagieren können. Organisationen oder Organisationseinheiten werden durch eigene Plattformen repräsentiert, auf denen sich ihre Mitglieder befinden und die Teil eines Gesamtsystems sind.

Dabei können diese Plattformen wiederum verschachtelt sein, um zum Beispiel Organisationseinheiten als Teil einer Organisation zu beschreiben. Wie bei MULAN handelt es sich um eine „relative Relationierung“ der Einheiten.

Zusätzlich lassen sich aber neue Organisationseinheiten, etwa für Projektteams, formieren, die auf Kollaborationsplattformen miteinander zusammenarbeiten können. Diese Kollaborationsplattformen sind logische Räume, in denen Interaktion, Kooperation und der Zugriff auf gemeinsame Ressourcen ermöglicht werden. Auf diese Weise können für spezifische Projekte schnell neue Kooperationen aufgesetzt werden. Die Nachbildung bestehender Kooperationsbeziehungen in der Entwicklungsumgebung unterstützt das Application/IT-Alignment (mehr dazu in Abschnitt 2.1).

Die Ablauforganisation von Projekten wird unterstützt durch eine Prozessinfrastruktur, die die im Projekt angewendeten Arbeitsprozesse unterstützt und steuert. Für verschiedene Vorgehensmodelle lassen sich Abläufe definieren, die dann in konkreten Projekten als Vorlage für die verwendeten Softwareentwicklungsprozesse dienen. Diese Vorgaben müssen dann an die individuellen Gegebenheiten und Erfordernisse angepasst werden.

Hiervon profitiert das Projektmanagement, da Projektmanager im System Zugriff auf die benötigten Informationen bezüglich ihrer Projekte bekommen, vom Ressourceneinsatz über den Projektfortschritt bis hin zu den im Projekt erstellten Dokumenten. Dadurch ist ein effektives Monitoring möglich. Auch die Steuerung und Anpassung der Abläufe, Projektrollen und Ressourcenzuordnungen wird ermöglicht.

Darüber hinaus ist es Aufgabe des Projektmanagers, dafür zu sorgen, dass die Teammitglieder über die notwendigen Ressourcen und Informationen ver-

fügen, um ihre Aufgaben ausführen zu können. Hierbei kann das System ihn unterstützen, indem es die Ressourcen verwaltet, die zu einem Prozess benötigt werden und den Prozessteilnehmern die benötigten Dokumente zur Verfügung stellt.

Konflikte um Ressourcen in verteilten Umgebungen können durch Verhandlung zwischen den verschiedenen beteiligten Parteien gelöst werden. Diese können dann gemeinsam einen geänderten Ablauf definieren, der allen Bedürfnissen Rechnung trägt.

2.2.2 Unterstützung der Entwicklerarbeit

Eine Entwicklungsumgebung unterstützt den Entwicklungsprozess, indem sie die erforderlichen Hilfsmittel und Ressourcen zur Verfügung stellt, die für die jeweiligen Aufgaben benötigt werden. Es wäre wünschenswert, eine verteilte Entwicklungsumgebung zu haben, die es erlaubt, Hilfsmittel und Ressourcen zwischen Benutzern auszutauschen, um Arbeitsergebnisse weiterzureichen oder gemeinsam zu bearbeiten.

Zur optimalen Arbeitsunterstützung kann sich der Entwickler seinen persönlichen Arbeitsplatz mit den Werkzeugen einrichten, die er benötigt und diese nach Wunsch konfigurieren. Wenn er sich an einem anderen physikalischen Arbeitsplatz befindet, kann er sich seine aktuelle Arbeitsumgebung wieder herstellen lassen.

Die Werkzeugunterstützung umfasst den gesamten Entwicklungsprozess, von der Anforderungsermittlung über den Entwurf und die Implementierung bis zur Qualitätssicherung und Wartung. In jedem dieser Schritte gibt es bestimmte Tätigkeiten und Dokumente, für die spezifische Hilfsmittel benötigt werden und die von verschiedenen Parteien erstellt, bearbeitet und weiterverwendet werden. Durch Kollaborationswerkzeuge kann darüber hinaus die Zusammenarbeit zwischen den Prozessbeteiligten verbessert werden, etwa indem auftretende Fragen schnell beantwortet oder auch Dokumente gemeinsam bearbeitet werden können.

Der Grad der Unterstützung reicht von einfachen Editoren über spezialisierte Modellierungs- und Debuggingwerkzeuge bis hin zu intelligenten Assistenten und autonomen Prozessen, die als eigenständige Akteure Teile des Entwicklungsprozesses ausführen. Ein Beispiel dafür ist die Unterstützung der Qualitätssicherung (QS) durch automatisierte Tests. Nach jeder Codeänderung werden automatische Testsuiten ausgeführt, um sicherzustellen, dass nicht versehentlich ungewollte Seiteneffekte eingebaut wurden. Dies wird als „Continuous Integration“ bezeichnet und zum Beispiel durch Werkzeuge wie JUnit [JUnit, 2011], Selenium [Selenium, 2011], CruiseControl [CruiseControl, 2011] oder Hudson [Hudson, 2011] unterstützt.

Zusätzlich wird der Entwicklungsprozess durch die in der Prozessinfrastruktur hinterlegten Abläufe unterstützt. Jeder Beteiligte hat dadurch stets einen Überblick über seine aktuellen Aufgaben. Der Gesamtprozess eines Projektes

lässt sich in verschiedene Teilprozesse unterteilen. Die am Projekt beteiligten Organisationen können ihre eigenen lokalen Prozesse selbst verwalten. Diese Subprozesse bilden dann einen Teil des Gesamtprozesses, der unter Umständen nur lokal im Detail sichtbar ist.

Ein anderes Beispiel sind die oben erwähnten autonomen Testprozesse. Die Test-Tools können als eigenständige Akteure im System auftreten und neue Prozesse starten. Wird in einem automatisierten Test ein Fehler gefunden, kann automatisch ein Eintrag in einem Bugtracking-Werkzeug vorgenommen werden, wie etwa Jira [Jira, 2011a] oder Bugzilla [Bugzilla, 2011]. Zusätzlich wird ein spezieller Prozess für die Behebung des Fehlers gestartet.

2.2.3 Unterstützung der Kundeneinbindung

Um die Bedürfnisse eines Kunden bei der Softwareentwicklung besser berücksichtigen zu können, sieht die verteilte IDE den Kunden bzw. die Kundenorganisation als wichtigen Teil des Entwicklungsprozesses an. Auf welche Art diese Einbindung geschieht, ist abhängig von den verwendeten Vorgehensmodellen und Entwicklungsprozessen.

Mitarbeiter der Kundenorganisation können direkt in den Entwicklungsprozess eingebunden werden. Dafür wird definiert, auf welche Teile der Anwendung sie zugreifen können und an welchen Aktivitäten sie beteiligt werden. Diese Beteiligung kann auf verschiedene Arten erfolgen.

Aufgaben im Prozess können komplett an Vertreter der Kundenorganisation übertragen werden, beispielsweise die Durchführung von Tests oder die Abnahme fertiger Module. Neue Prozesse können durch den Kunden gestartet werden, etwa die Beauftragung einer Änderung in der Software oder ein Prozess zur Behebung eines konkreten Fehlers. Schließlich können Aktivitäten durch Kollaboration verschiedener Beteiligter bearbeitet werden, zum Beispiel in Form von kollaborativer Modellierung oder durch virtuelle Meetings. Hierfür sind verschiedene Hilfsmittel erforderlich, die den Zugriff auf die erforderlichen Funktionen und Ressourcen ermöglichen.

Die Art der Einbindung der Kundenorganisation in die Entwicklung ist stark abhängig von den beteiligten Organisationen. Die Definition festgelegter Interaktionsprozesse erlaubt eine klar nachvollziehbare und konstruktive Zusammenarbeit. Schnelle Feedback-Zyklen erlauben eine bessere Kundeneinbindung und damit Software, die sich enger an den Bedürfnissen des Kunden orientiert.

2.3 Perspektiven

Die verschiedenen am Softwareentwicklungsprozess beteiligten Organisationen und Funktionsbereiche stellen eine Reihe unterschiedlicher Anforderungen an die Entwicklungsumgebung. Um diese miteinander in Einklang zu bringen,

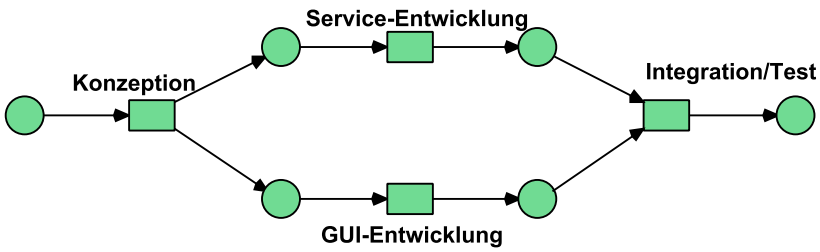

Abbildung 2.2: Beispiel-Entwicklungsprozess

werden für die Entwicklung der Systemarchitektur verschiedene Perspektiven betrachtet.

Im Folgenden werden drei Perspektiven auf die Systementwicklung beleuchtet, die in dieser Arbeit betrachtet werden: Agenten, Workflows, sowie Ressourcen und Helfer. Eine Synthese aus diesen drei Perspektiven bildet den konzeptionellen Rahmen der Arbeit und dient als Grundlage für die Architektur der Entwicklungsumgebung.

2.3.1 Agenten

Betrachtet man die verschiedenen Akteure, die am verteilten Entwicklungsprozess beteiligt sind, so sieht man eine Reihe autonom arbeitender Akteure, die jeweils ihre eigenen Ressourcen, Ziele und Pläne haben. Sie interagieren miteinander, um ihre eigenen Ziele zu erreichen, die gemeinsam einem Gesamtzweck dienen. Das entspricht recht gut der Beschreibung eines Multiagentensystems.

Daher bietet sich eine Implementierung der Entwicklungsumgebung als Multiagentensystem an, wie sie im Paose-System grundsätzlich für Anwendungen verfolgt wird. Als Leitbild hierfür soll die Metapher des „Multiagentensystems von Entwicklern“ [Cabac, 2007] verwendet werden. Die am Entwicklungsprozess beteiligten Parteien können als Agenten innerhalb eines Multiagentensystems aufgefasst werden, die miteinander interagieren.

Diesem Leitbild folgend lässt sich dann auch die Entwicklungsumgebung als Multiagentensystem verstehen, modellieren und konzipieren. Die Beteiligten werden jeweils durch einen Agenten im System dargestellt, die verschiedenen Orte durch Agentenplattformen, die miteinander in Verbindung stehen. Die Interaktionen lassen sich durch Agenteninteraktionen modellieren.

Als Beispiel soll hier ein einfaches Entwicklungsmodell für ein Softwaremodul verwendet werden, dass auch im weiteren Verlauf zur Veranschaulichung verwendet wird (Abb. 2.2). Bei diesem Prozess wird zunächst durch einen Softwarearchitekten ein Konzept für das zu entwickelnde Modul erstellt. Dieses Konzept enthält Service- und GUI-Bestandteile, die dann jeweils von einem Entwickler separat implementiert werden. Sind die beiden Einzelkomponenten fertig gestellt, werden sie durch einen Integrator zusammengeführt. Um den

Aspekt der Verteilung einzubringen, soll davon ausgegangen werden, dass Architekt und Integrator sich an einem Ort befinden, die beiden Entwickler aber an zwei anderen Orten (bzw. in anderen Organisationen).

In der Agentenbetrachtung ist dies ein Multiagentensystem mit vier beteiligten Agenten auf drei Agentenplattformen. Das Gesamtsystem umfasst möglicherweise noch weitere Plattformen, die weitere Agenten beherbergen können. Die Interaktion wird proaktiv durch den Architektenagenten gestartet (bzw. ein eventueller externer Auslöser für seine Aktivität ist im betrachteten Modell nicht sichtbar). Im Falle eines MULAN-Agenten (vgl. Abschnitt 3.2.3) würde er hierfür ein Protokoll für Modulentwicklung starten.

Er führt zunächst eine lokale Bearbeitung durch, um das Konzept zu erstellen. Anschließend sendet er Nachrichten an die beiden Entwickleragenten, in denen er sie auffordert, ihren jeweiligen Teil der Entwicklung zu leisten und ihnen das Konzept zukommen lässt.

Die beiden Entwickleragenten starten daraufhin ihre Protokolle für diese Interaktion, verarbeiten die eingegangene Nachricht und arbeiten dann nebenläufig an ihrem jeweiligen Teil der Implementierung. Ein Teil dieser Arbeit kann von den Agenten lokal durchgeführt werden, ohne mit anderen Agenten interagieren zu müssen, häufig bestehen aber Abhängigkeiten zwischen verschiedenen Teilbereichen einer Software. Die GUI-Entwicklung ist beispielsweise auf bestimmte Servicemethoden angewiesen, die sich schlecht durch Testversionen ersetzen lassen, oder es werden Anforderungen für weitere Methoden deutlich, die die Spezifikation nicht vorgesehen hat.

Obwohl die Aktivitäten unabhängig und nebenläufig modelliert sind, kann es also zum Teil starke Interaktionsbedarfe geben. Wenn Kommunikationswege zwischen den Entwicklern außerhalb des Systems existieren, kann die Koordinierung über diese erfolgen (das Gespräch an der Kaffeemaschine), und muss nicht explizit in der Software modelliert werden.

Zur nachhaltigen Modellierung der Entwicklungstätigkeiten sollten jedoch möglichst viele Kommunikationswege modelliert und durch das System erfasst werden (im Rahmen einer sinnvollen Kosten/Nutzen-Abwägung). Twitter, Blogs, Wikis und Foren sind erste technische Realisierungen solcher Abbildungen. Durch die hier verfolgte agentenorientierte Architektur lassen sich diese dann mittels HERA einbetten.

Thema hier ist aber eine verteilte Entwicklung, bei der die unmittelbare Kommunikation nicht immer gegeben ist. Grund hierfür können organisationale Grenzen oder auch schlicht unterschiedliche Zeitzonen sein. Die Entwicklung der verschiedenen Komponenten kann daher nicht durch eine einfache lokale Aktivität dargestellt werden, wie dies für den Architekten angenommen wurde, sondern durch eine explizit im System modellierte Interaktion zwischen den beiden Agenten.

Das Gleiche gilt für die Integration und den Test, die im Anschluss durch den Integrator durchgeführt werden sollen. Auch diese Aktivität ist auf den ersten Blick unabhängig und lokal durchzuführen. In der Praxis ergeben sich

aber oft Nachfragen, Klärungsbedarfe und (in diesem einfachen Modell nicht dargestellte) Korrekturschleifen, die die Interaktion aller vier Agenten erfordern kann.

2.3.2 Workflow

Eine andere mögliche Sichtweise auf das beschriebene Vorgehen besteht darin, die Abläufe zwischen den beteiligten Parteien in den Vordergrund der Betrachtung zu stellen. Die Prozesse bei der Softwareentwicklung können sehr komplex werden, insbesondere wenn viele verschiedene Parteien/Organisationen beteiligt sind. Anstatt die einzelnen Akteure und ihre Interaktionen zu betrachten, wird ein Prozessmodell gebildet, das im Einzelfall instanziiert und ausgeführt wird.

Im oben genannten Beispiel würde also ein Workflow zur Modulentwicklung definiert werden, der die Aktivitäten „Design“, „Service-Entwicklung“, „GUI-Entwicklung“ und „Integration“ enthält. Abb. 2.2 zeigt den Entwicklungsprozess als Workflownetz. Diese Aktivitäten sind bestimmten Rollen zugewiesen und mit bestimmten Ressourcen verbunden.

Um nun ein konkretes Modul zu implementieren, wird eine Instanz dieses Workflows gestartet und mit den erforderlichen Parametern versehen. Die Definition sieht vor, dass die erste Aktivität durch einen Workflow-Teilnehmer ausgeführt wird, der die Rolle „Architekt“ innehat. Der entsprechende Teilnehmer wird ermittelt und die Aufgabe zur Konzeption in seine Worklist eingetragen. Hier können natürlich unterschiedliche Verfahren zur Bestimmung des richtigen Teilnehmers, Push oder Pull der Aufgaben etc. verwendet werden.

Der Softwarearchitekt erhält nun diese Aufgabe und bearbeitet sie mit den ihm zur Verfügung stehenden Mitteln. Anschließend bestätigt er den Abschluss der Aufgabe beim Workflowmanagementsystem. Zusätzlich fügt er das erstellte Konzept als Ressource zum System hinzu, damit es im weiteren Prozessverlauf verwendet werden kann.

Anschließend ermittelt das WfMS die Folgeaufgaben zur Umsetzung des Konzepts und die hierfür vorgesehenen Teilnehmer. Auch diese erhalten ihre entsprechenden Aufgaben über ihre Worklisten zugewiesen. Im Gegensatz zum Agentensystem ist hier also eine Entkoppelung der Prozessbeteiligten zu beobachten, da die Koordination nicht mehr direkt, sondern über das WfMS erfolgt. Der Architekt muss nun nicht mehr wissen, an wen er das Konzept senden muss, er muss nicht einmal wissen, ob im nächsten Schritt bereits die Entwicklung steht, oder ob zunächst beispielsweise das Konzept zur Abstimmung an den Kunden gesandt wird. Das WfMS hilft dabei, diese Details zu isolieren und damit leichter änderbar zu machen, insbesondere, wenn es eine dynamische Anpassung der Prozesse zur Laufzeit ermöglicht (vgl. [Aalst, 2001, Reichert und Dadam, 1997]).

2.3.3 Ressourcen und Helfer

Die dritte Perspektive, die hier betrachtet wird, betrifft die Arbeitsgegenstände, die im Verlauf der Arbeit verwendet werden. Dies umfasst einerseits die Ressourcen, die im Laufe des Prozesses erstellt und bearbeitet werden, andererseits die Hilfsmittel, mit denen auf diese Ressourcen zugegriffen und mit denen sie bearbeitet werden.

Der Softwarearchitekt erstellt ein Konzept für das neue Softwaremodul. Das Konzept umfasst beschreibende Texte, Modelle, Grafiken etc. Um das Konzept zu erstellen, kann er spezialisierte Software verwenden, etwa CASE-Tools, oder auch eine Kombination verschiedener Programme, wie Textverarbeitungsprogramme, Grafikprogramme, spezialisierte Programme für bestimmte Teilaufgaben wie etwa UML-Diagramme etc.

Am Ende steht ein Dokument, das an die Entwickler weitergegeben werden kann. Dieses Dokument dient diesen nun als Grundlage für die Erstellung ihres Codes. Die Entwickler verwenden ihrerseits Werkzeuge für ihre Aufgaben, z.B. eine IDE, die die Bearbeitung und Kompilierung des Programmcodes erlaubt und erzeugen ihrerseits neue Ressourcen, zum Beispiel Programmcode und Dokumentation.

Bei dieser Sichtweise, die sich unter anderem am Werkzeug und Material Ansatz (WAM) [Züllighoven, 2004] orientiert, erfolgt die Koordination der verschiedenen erforderlichen Tätigkeiten über die Ressourcen. Die Softwareentwickler wissen, was von ihnen erwartet wird, weil sie ein Konzept vorgelegt bekommen, der Integrator weiß, dass er mit der Integration beginnen kann, weil er die fertigen Teilprogramme bekommt. Eine explizite Aufgabensteuerung erfolgt aber in der Regel nicht.

Diese Sichtweise orientiert sich stärker an dem, was ein Anwender in seiner täglichen Arbeit vorfindet, Gegenstände, mit denen er umgeht, Dokumente, die er erzeugt und weitergibt. In der Regel muss der Anwender aber selber wissen, an wen er eine Ressource weitergeben muss, damit diese weiter bearbeitet werden kann. Dadurch ergeben sich mitunter Schwierigkeiten in Systemen, die eine Vielzahl von Beteiligten enthalten und bei denen nicht immer klar ist, wer an einem Prozess beteiligt ist.

Im WAM-Ansatz werden hierfür spezielle Koordinationsmaterialien verwendet, wie etwa Projektakten oder Laufzettel. Der WAM-Ansatz liefert insbesondere eine wesentlich verfeinerte Modellierung objektorientierter Systeme, indem weitere Klassifikationsmerkmale der Objekte die Kommunikation der beteiligten Entwickler unterstützen.

2.3.4 Agenten und Workflow

Der nächste Schritt in der Betrachtung besteht nun darin, die verschiedenen Konzepte miteinander zu kombinieren. Eine Integration von Agenten- und Workflowkonzepten findet sich in [Reese, 2010]. Dort wird ein Vorgehen zur

Anwendungsentwicklung skizziert, bei dem Agenten zur Strukturierung verwendet werden und ein Workflow Management System die Prozesse steuert. Dieses WfMS kann ebenfalls durch Agenten implementiert sein.

Statt eines Multiagentensystems, in dem die Agenten zu jedem Zeitpunkt selbst entscheiden, was sie tun, liegt nun eine Prozesssteuerung vor, die die Abläufe zwischen den Agenten steuert. Dies kombiniert die autonome Problemlösungsfähigkeit eines Agenten mit der Strukturierung und Nachvollziehbarkeit, die ein WfMS mit sich bringt. Agenten können die Prozesssteuerung nutzen, um ihre Abläufe zu steuern, die eigenen Entscheidungskomponenten wählen aber weiterhin aus, was der Agent tatsächlich tut und ob er die vom WfMS angebotenen Aufgaben annehmen möchte.

Im Beispiel sieht das dann wie folgt aus: Zusätzlich zu den Agenten für den Softwarearchitekten, die Entwickler und den Integrator gibt es im System einen oder mehrere Agenten, die die Rolle des WfMS einnehmen. In der Prozessinfrastruktur, die in [Reese, 2010] vorgestellt und die im weiteren Verlauf der Arbeit näher ausgeführt wird, sind hierfür eine Reihe verschiedener Agenten zuständig, die in Interaktion miteinander die Dienstleistung eines WfMS erbringen. Der Einfachheit halber wird hier zunächst das WfMS als ein einzelner Agent betrachtet.

Der Prozess beginnt damit, dass eine Anfrage an das WfMS gesendet wird, eine Instanz des Modulentwicklungsworkflows zu starten. Im WfMS wird dann eine Instanz des Workflows gestartet und, wie im Falle des einfachen WfMS, der Workflowteilnehmer für die erste Aufgabe ermittelt. In diesem Fall ist dies allerdings ein Agent. Der Agent des Softwarearchitekten wird über eine Agentennachricht darüber unterrichtet, dass eine Aufgabe für ihn im System ansteht.

Der Softwarearchitekt erstellt wieder das Konzept und sendet es an das WfMS. Dies ähnelt sehr dem Vorgehen in einem normalen WfMS, außer dass Workflow-Teilnehmer und WfMS als Agenten implementiert sind und entsprechend über Agentennachrichten kommunizieren. Damit können durch Benutzer ausgeführte Aufgaben im Workflow genauso behandelt werden wie automatisierte Aufgaben, da beide aus Sicht des WfMS durch einen Agenten behandelt werden. Die inhärente Verteilung von Multiagentensystemen erleichtert auch die Verteilung der Prozesse auf verschiedene Organisationen.

2.3.5 Agenten und Helfer

Bei der Betrachtung der Interaktion von Agenten wurde der Aspekt, wie die Agenten ihre Aufgaben ausführen, bisher außer Acht gelassen. Eine Kombination mit der werkzeugorientierten Sichtweise hilft, diese Lücke zu schließen.

Es gibt verschiedene Möglichkeiten, Ressourcen/Hilfsmittel und Agenten zu kombinieren. Eine Möglichkeit ist, das Grundmodell des Agenten (in diesem Fall Mulan) explizit um Konzepte wie Ressourcen und Hilfsmittel zu erweitern. Zurzeit kennt ein Mulan-Agent Protokolle, die sein Verhalten be-

schreiben, Entscheidungskomponenten, die Planungs- und Entscheidungsprozesse übernehmen und eine Wissensbasis, in der das Wissen des Agenten vorgehalten wird.

Als Erweiterung wäre es möglich, Ressourcen in die Wissensbasis einzufügen oder über eine ähnliche Schnittstelle verfügbar zu machen, so dass der Agent von verschiedenen Stellen aus auf die ihm eigenen Ressourcen zugreifen kann. Hilfsmittel erweitern den Agenten um neue Handlungsoptionen und lassen sich über eine ähnliche Schnittstelle wie Protokolle und Entscheidungskomponenten einbinden. Auch kann die Agentenplattform Ressourcen und Hilfsmittel verwalten und sie Agenten bei Bedarf über bestimmte Zugriffsprotokolle verfügbar machen.

Eine andere Möglichkeit besteht darin, Hilfsmittel und Ressourcen explizit als eigene Agenten zu modellieren, die mit anderen Agenten (etwa einem Benutzeragenten) interagieren und diesem so Zugriff auf die durch sie repräsentierten Inhalte und Handlungsoptionen ermöglichen. Hilfsmittel werden dann zu aktiven „Helfern“, die mehr oder weniger autonom den Benutzer unterstützen. Ressourcenagenten können selbstständig ihre Inhalte verwalten und zum Beispiel bestimmte Benutzungsmuster durchsetzen. Ein Kollaborationsmaterial wie ein Laufzettel könnte sich auch selbstständig um seine Bearbeitung bemühen.

Die zuerst genannte Lösung würde stark in das Grundmodell von Mulan eingreifen, während die Implementierung als separate Agenten sich im Rahmen der vorhandenen Architektur umsetzen lässt und zudem Helfer und Ressourcen als vollwertige Entitäten modelliert. Dabei werden spezielle Agenten verwendet, die Helfer und Ressourcen repräsentieren [Lehmann und Markwardt, 2004, Lehmann et al., 2005, Markwardt und Moldt, 2010]. Diese Agenten können flexibel in den Agenten eines Benutzers eingebunden werden und erweitern diesen dann um neue Funktionalität. Gleichzeitig behalten sie aber die Autonomie und Flexibilität eines Agenten.

Ein weiterer Vorteil dieser Umsetzung ist der Austausch von Gegenständen zwischen Benutzern des Systems. Dies lässt sich mit vorhandenen Mitteln durch Agentenmigration bzw. durch den Austausch von Referenzen erreichen. Insbesondere verteilte Materialien, die gleichzeitig von mehreren Parteien genutzt werden sollen, lassen sich so gut darstellen, während das bei Ressourcen innerhalb der Wissensbasis größere Probleme darstellen würde.

Im Beispiel würde dann der Softwarearchitekt einen Konzeptionshelfer verwenden, um das Softwarekonzept zu erstellen. Dieser Helfer ist ein Agent, der in Kooperation mit dem Benutzeragenten des Architekten die Erstellung des Konzeptes unterstützt. Dafür erweitert er den Benutzeragenten beispielsweise um Werkzeuge wie einen Diagrammeditor oder eine Textverarbeitung. Im Verlauf der Bearbeitung wird ein neuer Agent erzeugt, der die erstellte Ressource repräsentiert, der Ressourcenagent.

Diese Ressource kann dann an den nächsten Agenten weitergeschickt werden, der mit seinen Helfern eine weitere Bearbeitung vornehmen kann. Denkbar

ist eine Migration des Ressourcenagenten zwischen den verschiedenen Agentenplattformen im System oder der Zugriff über Nachrichtenkommunikation, je nachdem welche Dienste eines Ressourcenagenten benötigt werden.

Die Konzept-Ressource im Beispiel kann von den beiden Entwicklern gleichzeitig verwendet werden, da beide nur einen lesenden Zugriff darauf benötigen. In anderen Fällen kann das Material dafür sorgen, dass der Zugriff im wechselseitigen Ausschluss erfolgt. Für die Entwicklung der GUI und der Services können die Softwareentwickler unterschiedliche Helfer benutzen, die ihre jeweilige Arbeitsweise am Besten unterstützen.

Innerhalb des Systems können Helfer und Ressourcen zwischen den Agenten ausgetauscht werden. Neue Arten von Helfern und Ressourcen können als neue Agenten in das System eingebracht werden. Um das zu vereinfachen, wurde bereits in [Lehmann und Markwardt, 2004, Lehmann et al., 2005] unter dem Namen „Werkzeugfabrik“ ein Agent vorgeschlagen, der dazu dient, neue Helferagenten zu erzeugen und der alle in einem (Teil-)System bekannten Helfertypen kennt.

2.3.6 Workflow und Hilfsmittel

Ein anderer Aspekt ist die Kombination eines WfMS mit Werkzeug- und Materialaspekten. Auf der einen Seite werden Hilfsmittel verwendet, um auf das WfMS zuzugreifen, beispielsweise eine Worklistenverwaltung oder ein Editor zum Erstellen neuer Workflowdefinitionen. Auf der anderen Seite tauchen Hilfsmittel und Ressourcen aber auch als Objekte innerhalb des Workflows auf. Aktivitäten erfordern Hilfsmittel, um sie auszuführen. Sie erlauben die Erstellung und Bearbeitung von Ressourcen, die am Ende ein gewünschtes Endprodukt des Workflows ergeben.

Wenn der Softwareerstellungsprozess gestartet wird, erhält der Softwarearchitekt über seine Worklist, die er mit einem entsprechenden Hilfsmittel ansehen und in der er Aufgaben auswählen kann, die Aufgabe zur Konzeptionierung des neuen Moduls. Um dies zu tun, stellt ihm das System das passende Hilfsmittel zur Verfügung, mit dem er das Konzept erstellt. Das Werkzeug ermöglicht die Bearbeitung von Texten, Diagrammen und Modellen und erzeugt eine neue Ressource, die nach Abschluss der Aufgabe im WfMS gespeichert wird.

Anschließend, wenn die Softwareentwickler ihre Aufgaben ausführen, bekommen sie nicht nur das vorher erstellte Konzept als Ressource zur Verfügung gestellt, sondern auch passende Werkzeuge, um die Implementierung vorzunehmen. Die verschiedenen Parteien müssen sich also nicht mehr selbst darum kümmern, was ihre Aufgaben sind oder womit sie diese bearbeiten. Die zu einem Prozess gehörenden Ressourcen werden an einer Stelle, im Workflow, zentral gespeichert. Dadurch wird die Entwicklung kontrollierbar und übersichtlich.

2.3.7 Agenten, Workflow und Helfer

Der letzte Schritt besteht nun darin, alle drei Konzepte zu einem Gesamtkonzept zusammenzuführen. In diesem Gesamtkonzept, das hier POTATO genannt wird (Process Oriented Tool Agents for Team Organization), werden die verschiedenen am Prozess beteiligten Nutzer durch Agenten in einem Multiagentensystem repräsentiert. Die Prozesse innerhalb des Systems werden durch ein agentenorientiertes Workflow Management System gesteuert. Der Zugriff der Agenten auf das WfMS und die Bearbeitung der Aufgaben erfolgt durch Helferagenten.

Das Beispiel stellt sich dann in diesem System wie folgt dar: Der WfMS-Agent erhält eine Nachricht mit der Anforderung, einen neuen Prozess zur Erstellung eines Softwaremoduls zu starten und die erforderlichen Parameter zur Initialisierung des Prozesses.

Er startet einen Workflow-Agenten, der die neue Workflowinstanz repräsentiert. Dies ist eine Art spezieller Ressourcenagent, der zum einen den aktuellen Zustand des Prozesses (zum Beispiel in Form eines Workflow-Petrinetzes) enthält, zum anderen Informationen zu den für die verschiedenen Aktivitäten zu verwendenden Helfer- und Ressourcentypen und zum dritten die im Prozess erzeugten und bearbeiteten Ressourcen.

Der Workflow-Agent wird durch das WfMS bearbeitet, welches daraus unter anderem die aktuellen Aufgabenlisten generiert und den entsprechenden Agenten zusendet. Die Benutzer wiederum sind über spezielle Workflow-Helferagenten mit dem WfMS verbunden, die auch die Aufgabenlisten entgegennehmen und dem Benutzer vorlegen. Benutzer können Aufgaben zur Bearbeitung annehmen. Daraufhin werden die nötigen Ressourcen und Helferagenten für die Bearbeitung zusammengestellt und dem Agenten zugestellt.

Der Workflow-Agent erzeugt einen Aktivitätsagenten, einen speziellen Helferagenten für die Erzeugung von Softwarekonzepten. Dieser ist vorkonfiguriert mit Informationen, die der Softwarearchitekt für seine Aufgabe benötigt. Im Laufe der Bearbeitung erzeugt er einen neuen Ressourcenagenten für das Softwarekonzept, das er nach Abschluss der Aufgabe in den Workflow-Agenten integriert.

Die Softwareentwickler erhalten anschließend ihre jeweiligen Aktivitätsagenten, die sie in der Umsetzung ihrer Implementationsaufgaben unterstützen. Im Gegensatz zu den Helferagenten, die sie im einfachen Modell verwenden, sind diese spezialisiert für die Ausführung der zugehörigen Workflowaufgabe. Beispielsweise ist das Softwarekonzept bereits integriert, so dass der Entwickler unmittelbar mit der Arbeit beginnen kann.

2.4 Vorgehen

Dieser Abschnitt verdeutlicht das Vorgehen, das für den weiteren Verlauf dieser Arbeit gewählt wird. In Kapitel 3 werden die Grundlagen erläutert, auf denen die Arbeit aufbaut. Dies betrifft einerseits die fachlichen Hintergründe in den Bereichen Softwareentwicklung und Softwareentwicklungsumgebungen, Computer Supported Cooperative Work und Workflowmanagementsysteme. Andererseits werden die technischen Grundlagen beleuchtet, die der Modellierung und Implementierung des Prototypen zugrundeliegen.

Für den Ansatz der Petrinetzbasierten Agenten- und Objektorientierten SoftwareEntwicklung (PAOSE) werden Petrinetze und ihr Einsatz für die Softwareentwicklung, sowie der Ansatz der Agentenorientierten Softwareentwicklung beschrieben. Für die Erstellung von Softwareentwicklungswerkzeugen wird auf den Werkzeug- und Materialansatz (WAM) Bezug genommen. Die zugrundeliegende Entwicklungsmethodik wird beschrieben und ein Ansatz entwickelt, wie diese Prinzipien in Agentensystemen aufgegriffen werden können.

Kapitel 4 widmet sich dem Begriff der Verteilten Softwareentwicklung. Dafür wird zunächst auf den allgemeinen Begriff des Software Engineering eingegangen, da die dort vorgefundenen Prinzipien und Qualitätsmerkmale ebenso für die verteilte Softwareentwicklung gelten.

Verteilte Software, sowie verteilte Softwareentwicklung werden erläutert und ein Überblick sowie eine Klassifikation und Bewertungskriterien gegeben für existierende Alternativen in diesem Bereich. Abschließend wird die Verwendung von Agenten und Multiagentensystemen für die verteilte Softwareentwicklung motiviert.

Die komplette Entwicklung einer Softwareentwicklungsumgebung liegt außerhalb dessen, was diese Arbeit leisten kann. Der Fokus der Konzeption und insbesondere der prototypischen Implementierung wird daher auf eine Zwischenstufe gelegt. Das POTATO-System (Process-Oriented Tool Agents for Team Organization) bildet eine Plattform für verteilte Agentenanwendungen, die auf eine Prozesssteuerung und Werkzeugunterstützung zurückgreifen können, um ihren Zweck zu erfüllen. Die Entwicklungsumgebung wird als Beispielanwendung für diese Plattform betrachtet. Einzelne Agenten, die auf dieser Plattform aufsetzen und die für die Softwareentwicklung verwendet werden können, werden als Beispiele zur Verdeutlichung der Konzepte angeführt.

In Kapitel 5 wird das Agentensystem POTATO vorgestellt. Dazu werden zunächst Anforderungen aufgestellt, die an die Verteilte Entwicklungsumgebung gestellt werden, ausgehend von diesen werden dann verschiedene Architekturmodelle diskutiert, bevor ein Vorschlag für eine Architektur entwickelt wird.

Die zentralen Säulen, auf denen die Architektur beruht, sind das agentenorientierte Unterstützungssystem HERA, das in Kapitel 6 beschrieben wird, sowie die Prozessinfrastruktur für Agentensysteme (PIA), die Thema in Kapitel 7 ist.

Im Anschluss an die Beschreibung der generellen Architektur von POTATO wird daher zunächst das HERA-System vorgestellt. Anforderungen an ein agentenorientiertes System zur Modellierung und Implementierung von Helfereinheiten und Ressourcen werden aufgestellt, aus diesen ein Entwurf entwickelt und Prototypen zur Umsetzung vorgestellt.

Anschließend wird die Prozessinfrastruktur PIA vorgestellt. Auch hier werden zunächst die Anforderungen entwickelt und ein Entwurf vorgestellt. Zusätzlich wird hierbei die Integration der Helfer- und Ressourcenagenten aus dem HERA-System beschrieben.

Kapitel 8 widmet sich dann den verschiedenen Prototypen, die implementiert worden sind. Es wird auf verschiedene Designentscheidungen eingegangen und deutlich gemacht, wie der PAOSE-Entwicklungsansatz genutzt wurde, um die Entwicklung zu organisieren.

Am Ende folgt eine Schlussbetrachtung mit einer Zusammenfassung der Ergebnisse und einem Ausblick auf weitere Fragestellungen und weiterführende Themengebiete.

Kapitel 3

Grundlagen

Dieses Kapitel stellt die Grundlagen vor, auf die im Rest der Arbeit Bezug genommen wird und auf denen das POTATO-System aufbaut. Dafür werden die benötigten Konzepte im Bereich der Petrinetze, Workflowmanagementsysteme, Computer Supported Cooperative Work (CSCW) und Multiagentensysteme vorgestellt.

Darüber hinaus wird der PAOSE-Ansatz zur agentenorientierten Softwareentwicklung mit Petrinetzen vorgestellt, sowie der Werkzeug- und Materialansatz, der eine Inspirationsquelle für die Entwicklung der agentenbasierten Unterstützungsumgebung HERA darstellt. Abschließend wird die im Verlauf der Arbeit verwendete Terminologie zur Referenz zusammengefasst.

3.1 Petrinetze

Für die Modellierung und prototypische Implementierung der verteilten Entwicklungsumgebung werden in dieser Arbeit Petrinetze verwendet. Petrinetze sind eine Familie von Formalismen zur Modellierung nebenläufiger Systeme, für die eine Vielzahl von Modellierungs- und Analysemethoden existieren.

Als erstes werden der Basisformalismus für Petrinetze (P/T-Netze) vorgestellt und die grundlegende Funktionsweise von Petrinetzen erläutert. Eine spezielle Unterklasse von Petrinetzen, die Workflownetze, werden verwendet, um Geschäftsprozesse und andere strukturierte Abläufe zu modellieren und zu verifizieren. Workflownetze werden in dieser Arbeit verwendet, um die Beschreibungen von Workflowprozessen innerhalb der Prozessinfrastruktur zu verifizieren (siehe Kapitel 7).

Für die Modellierung und Implementierung der Entwicklungsumgebung wird der Formalismus der Referenznetze verwendet [KUMMER, 2002]. In Referenznetzen können Netze selbst wieder als Marken in anderen Netzen verwendet werden, womit eine große Ausdrucksmächtigkeit für die Modellierung komplexer Systeme gegeben ist.

3.1.1 P/T-Netze

Petrinetze bestehen aus drei Typen von Netzelementen, bezeichnet als *Stellen* (auch *Plätzen*) und *Transitionen*, sowie verbindenden *Kanten*. Dabei gibt es einen Dualismus von Bedingungen, Zuständen oder Eigenschaften auf der einen Seite und Aktivitäten, Ereignissen oder Handlungen auf der anderen Seite. Die ersteren werden durch Stellen repräsentiert und in graphischen Darstellungen meist durch Kreise dargestellt, letztere als Transitionen und durch Rechtecke oder fette Striche dargestellt.

Stellen und Transitionen sind verbunden durch Kanten, dargestellt durch Pfeile. Diese Kanten verlaufen stets von einer Stelle zu einer Transition oder von einer Transition zu einer Stelle, niemals zwischen zwei gleichen Typen von Netzelementen. Kanten stellen einen Kausalzusammenhang zwischen Stellen und Transitionen dar.

Eine Kante von einer Stelle zu einer Transition bedeutet, dass der entsprechende Zustand Vorbedingung ist für das Ausführen der Aktivität, die durch die Transition dargestellt wird. Umgekehrt bedeutet eine Kante von einer Transition zu einer Stelle, dass das Ausführen dieser Aktion die zur Stelle gehörige Bedingung erfüllt.

Die Stellen, von denen aus Kanten zu einer Transition t führen, werden als *Vorstellen*, *Vorbedingungen* oder *Vorplätze* von t bezeichnet, alle Vorstellen einer Transition bilden den *Vorbereich*, kurz bezeichnet als ${}^\bullet t$. Der *Nachbereich*, also die Menge aller Stellen, zu denen von t aus eine Kante führt, wird mit $t^\bullet$ bezeichnet.

Die folgenden Definitionen stammen aus [Starke, 1990, S. 21ff]:

Definition 1 (Netz)
Ein Netz ist ein Tripel $N = (P, T, F)$, wobei

- P die Menge der Stellen und
- T die Menge der Transition bezeichnen.
- $P \cap T = \emptyset$
- $F \subset (P \times T) \cup (T \times P)$ ist eine binäre Relation, die *Flussrelation*, mit $dom(F) \cup cod(F) = P \cup T$, das heißt, jedes Netzelement besitzt mindestens eine ein- oder ausgehende Kante.

Eine Stelle kann eine beliebige Anzahl von Marken enthalten, die das Vorhandensein bestimmter Ressourcen oder die Erfüllung einer Bedingung darstellen. Eine Ressource kann mehrfach vorhanden sein, bei einer Bedingung macht das im Allgemeinen keinen Sinn.

Mit *Markierung* wird ein Zustand eines Petrinetzes bezeichnet. Er ergibt sich aus den Markenanzahlen, die sich in allen Stellen befinden. Sie kann beispielsweise als Tupel von natürlichen Zahlen dargestellt werden, bei dem jede

Komponente der Markenanzahl einer Stelle entspricht oder als Multimenge über die Stellenmenge.

Für das Schalten einer Transition kann es erforderlich sein, dass in einer Stelle mehrere Marken liegen, oder es können beim Schalten mehrere Marken in eine Nachstelle gelegt werden. Dies wird durch die Angabe einer *Vielfachheit* an einer Kante dargestellt.

Definition 2 (P/T-Netz)
Ein *Platz/Transitions-Netz* (P/T-Netz), oder auch *Petrinetz* ist ein Tupel $N = (P, T, F, V, m_0)$ mit

- (P,T,F) ist ein Netz
- $V : F \to \mathbb{N}$ ist eine Abbildung, die jeder Kante $f \in F$ eine positive natürliche Zahl V(f) zuordnet, die *Vielfachheit* der Kante
- Ist $V(f) = 1$ für alle $f \in F$, heißt N *gewöhnlich.*
- $m_0 \in BAG(P)$[1] ist die *Anfangsmarkierung* des Netzes

Sind in einer Markierung in allen Stellen des Vorbereichs einer Transition mindestens so viele Marken vorhanden, wie die Vielfachheit der Kante von der Vorstelle angibt, so ist diese Transition *aktiviert* und kann schalten oder feuern. Dabei werden aus jeder Stelle im Vorbereich der Transition soviele Marken abgezogen wie es der Vielfachheit der entsprechenden Kante entspricht und dafür in jede Stelle in ihrem Nachbereich so viele Marken hinzugefügt wie die Anschriften der ausgehenden Kanten angeben.

Sei $N = (P, T, F, V, m_0)$ ein Petrinetz. $t \in T$, $m, m' \in BAG(P)$

- Die Abbildungen t^+, t^- und Δt seien wie folgt definiert:

  $t^+(p) := V(t, p)$, falls $p \in t^\bullet$, 0 sonst
  $t^-(p) := V(p, t)$, falls $p \in {}^\bullet t$, 0 sonst
  $\Delta t(p) = t^+(p) - t^-(p)$ ist die Wirkung der Transition t.

- t heißt aktiviert in m, wenn $t^- \leq m$.
- Die *Schaltrelation* $\to$ ist definiert als: $m \xrightarrow{t} m'$ genau dann, wenn $t^- \leq m$ und $m' = m + \Delta t$
- Die transitive, reflexive Hülle der Schaltrelation heißt *Erreichbarkeitsrelation.* Zu einem Netz heißt $R_N(m_0) = \{m \in BAG(P) | m_0 \xrightarrow{*} m\}$ die *Erreichbarkeitsmenge.*

[1] BAG(P) ist die Menge der Multimengen über P

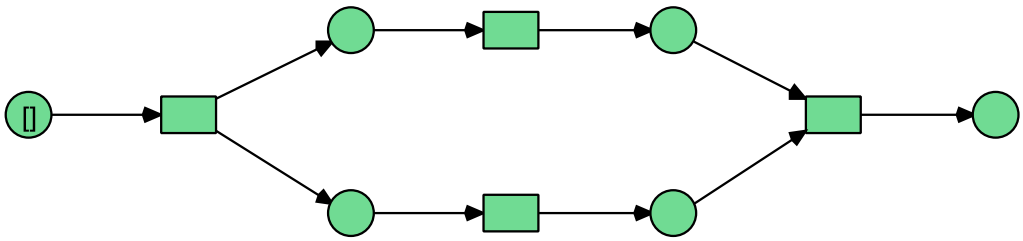

Abbildung 3.1: Ein Workflow-Petrinetz

3.1.2 Workflownetze

Ein wichtiger Anwendungsbereich von Petrinetzen ist die Modellierung von Geschäftsprozessen oder Workflows. Um Workflows zu verifizieren, werden in [Aalst, 1997] Workflownetze, eine spezielle Unterklasse von Petrinetzen, definiert.

Workflownetze zeichnen sich dadurch aus, dass sie stets mit einer Stelle beginnen, die als einzige in der Anfangsmarkierung mit einer Marke belegt ist, und mit einer Stelle enden. Dies sind jeweils die einzige Quelle und Senke des Netzes, alle anderen Stellen und Transitionen liegen auf Pfaden zwischen dieser Anfangs- und Endstelle. Teilweise werden anstatt einer Anfangs- und Endstelle auch Transitionen verwendet, die den Start und das Beenden des Workflows als Aktion darstellen. Abbildung 3.1 zeigt ein einfaches Beispiel für ein Workflownetz mit zwei nebenläufigen Aufgaben.

Diese Art der Darstellung abstrahiert von einer Reihe von Eigenschaften eines Workflows, beispielsweise von der Zeit, die ein Vorgang dauert, von benötigten Ressourcen und bestimmten Triggerbedingungen. Sie erlaubt aber eine Überprüfung von Workflowbeschreibungen auf ihre Fehlerfreiheit, Soundness genannt [Aalst, 1998].

3.1.2.1 Soundness

Mit Soundness wird eine Korrektheitsbedingung für Workflownetze bezeichnet. Diese Eigenschaft stellt sicher, dass ein Workflow zu jedem Zeitpunkt beendet werden kann und dass nach Abschluss keine offenen Aktivitäten mehr vorhanden sind. Die Soundness-Eigenschaft lässt sich mit petrinetzbasierten Analysemethoden (beispielsweise mit dem Woflan-Werkzeug) entscheiden [Verbeek et al., 2001, Aalst, 2011].

Damit ein Workflownetz sound ist, müssen drei Bedingungen erfüllt sein:

- Der Workflow kann zu jedem Zeitpunkt terminieren, d.h. von jeder erreichbaren Markierung des Netzes ist eine Endmarkierung erreichbar, d.h. eine Markierung, die eine Marke in der Endstelle enthält.

- Die Termination ist eindeutig, d.h. die einzig erreichbare Endmarkierung ist die Markierung, in der genau eine Marke in der Endstelle liegt. Dies bedeutet auch, dass keine weiteren Marken im Netz „liegen bleiben“ können.

- Jede Transition des Netzes kann in mindestens einer möglichen Schaltfolge schalten, es gibt also keine überflüssigen Aktivitäten, die nie zum Einsatz kommen können.

Diese Eigenschaften können auf einfache Weise überprüft werden, indem das sogenannte „kurzgeschlossene“ Netz konstruiert wird. Dafür wird die Endstelle durch eine neue Transition mit der Anfangsstelle verbunden. Dieses neue Netz ist genau dann lebendig und beschränkt, wenn das Ausgangsnetz sound ist. Ausführlichere Diskussionen und der Beweis hierzu finden sich in [Verbeek et al., 2001].

Die Soundness-Eigenschaft ist eine Mindestanforderung für Workflownetze, die Verklemmungen und unerwünschtes Verhalten vermeiden hilft. Sie kann effektiv automatisiert überprüft werden. Trotz Einhaltung der Soundness kann natürlich ein Workflow trotzdem fehlerhaft sein, beispielsweise indem Ressourcen spezifiziert werden, die im System nicht vorhanden sind oder auch indem ein Verhalten spezifiziert wird, dass so nicht erwünscht ist.

3.1.2.2 Workflownetze in der verteilten Entwicklungsumgebung

Workflownetze spielen in der Architektur der Entwicklungsumgebung an mehreren Stellen eine Rolle. Zum einen können die Protokolle[2] der Agenten für einzelne Abläufe als Workflownetze dargestellt werden (vgl. Abschnitt 3.2.3 und [Lehmann, 2003, Lehmann und Moldt, 2004]), zum anderen werden in der Prozessinfrastruktur Workflowprozesse verwendet, um die Bearbeitung von Projekten zu organisieren. Die dort verwendeten Netze sind Referenznetze, die eine workflowartige Struktur besitzen. Zu Analysezwecken können sie aber auf Workflownetze reduziert werden, um z.B. die Soundness-Eigenschaft zu testen.

3.1.3 Referenznetze

Es gibt eine Reihe von Erweiterungen des grundlegenden Modells von Petrinetzen. Diese erlauben es, Modelle einfacher zu formulieren oder ihnen eine höhere Ausdruckskraft zu verleihen. Gefärbte Netze[Jensen, 1986] beispielsweise erlauben es, innerhalb von Netzen verschiedene Sorten (Farben) von Marken zu verwenden.

[2]Gemeint sind hier Agentenprotokolle, die das Verhalten eines Agenten beschreiben. Die Agentenprotokolle mehrerer Agenten beschreiben eine Interaktion, die als Workflow interpretiert werden kann. Aufgrund der synchronen Kanäle sind dies aber keine echten Workflownetze.

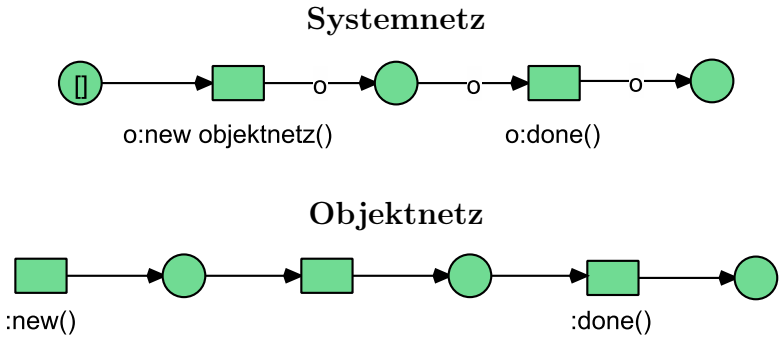

Abbildung 3.2: System- und Objektnetz in einem Referenznetzsystem

Referenznetze [Kummer, 2002] erlauben es, Transitionen innerhalb eines gefärbten Netzes und sogar zwischen verschiedenen Netzen miteinander zu synchronisieren. Sie schalten dann gemeinsam wie eine einzige Transition. Darüber hinaus können Referenzen zu anderen Netzen als Marken in einem Netz verwendet werden. Dadurch wird das Konzept der Netze in Netzen [Valk, 1998, Valk, 2004] realisiert. Das erlaubt eine Abstraktion und Verfeinerung durch Unternetze, die Teilaufgaben übernehmen und die Erstellung komplexer Softwarearchitekturen mit Netzen.

Abbildung 3.2 zeigt ein Beispiel für ein Referenznetzsystem. Wenn das obere Netz (das Systemnetz) schaltet, wird eine neue Instanz des Objektnetzes erzeugt und als Referenz in der Stelle im Systemnetz abgelegt. Gleichzeitig schaltet dabei im neuen Objektnetz die `:new()`-Transition. Ebenso schalten die beiden mit `:done()` beschrifteten Transitionen zeitgleich. Auf diese Weise können auch Informationen zwischen verschiedenen Netzen durch parametrisierte synchrone Kanäle ausgetauscht werden.

Durch die Möglichkeit, verschiedenste Typen von Marken, inklusive Referenzen auf andere Netze zu verwenden, sowie Netzelemente mit Anschriften in der Programmiersprache Java [Java, 2011] zu versehen, stellen Referenznetze ein mächtiges Modellierungswerkzeug dar. Die Software Renew [Renew, 2007] erlaubt die Erstellung und Ausführung von Referenznetzen. Die im Rahmen dieser Arbeit erstellten Modelle und Prototypen sind in weiten Teilen in Renew sowohl als Modellierungs- wie auch als Implementierungswerkzeug entwickelt worden.

3.2 Agenten und Multiagentensysteme

Das Paradigma der agentenorientierten Softwareentwicklung stellt eine Art der Strukturierung und Implementierung von Softwaresystemen dar, die die Interaktionen zwischen autonom agierenden Einheiten (Agenten) innerhalb eines Systems (Multiagentensystem) in den Vordergrund stellt. Die Grundbegriffe dieses Ansatzes werden hier vorgestellt. Außerdem wird das mit Hilfe von

Referenznetzen implementierte Multiagentensystem MULAN [KÖHLER et al., 2001] vorgestellt, auf dem das POTATO-System aufbaut.

3.2.1 Agenten

Obwohl der Begriff des Agenten zentral ist für einen agentenorientierten Ansatz der Softwareentwicklung, gibt es unterschiedliche Ansichten darüber, was einen Agenten ausmacht. [WOOLDRIDGE, 2000, S. 28ff] gibt Autonomie als zentrales Merkmal eines Agenten an.

Weitere Eigenschaften, die ein Agent haben kann, sind die Fähigkeit zu lernen, die Möglichkeit, eine Umwelt wahrzunehmen und zu beeinflussen, sowie die Interaktion mit anderen Agenten oder menschlichen Benutzern eines Computersystems. Handlungen des Agenten können als Reaktion auf einen wahrgenommenen Zustand der Umwelt erfolgen oder proaktiv vom Agenten angestoßen werden, um seine Ziele zu erreichen.

[WOOLDRIDGE und JENNINGS, 1995] unterscheiden eine starke und eine schwache Agentendefinition. Nach der schwachen Definition zeichnet sich ein Agent aus durch Autonomie, die Fähigkeit in sozialen Kontexten zu interagieren, auf Reize der Umgebung zu reagieren und von sich aus, proaktiv, Handlungen anzustoßen.

Die starke Definition fügt Eigenschaften wie Mobilität, Aufrichtigkeit, Wohlverhalten und Rationalität hinzu. Im Kontext dieser Arbeit interessieren vor allem die Eigenschaften der schwachen Agentendefinition. Für die Verwendung von Agenten als Werkzeuge und insbesondere als Ressourcen ist in vielen Fällen auch die Eigenschaft der Autonomie stark eingeschränkt.

3.2.2 Multiagentensysteme

Ein Multiagentensystem (MAS) ist ein System, in dem mehrere Agenten miteinander in Interaktion treten können. Diese können sich dafür auf verschiedenen Agentenplattformen befinden. Eine konkrete Architektur für ein Multiagentensystem wird in Abschnitt 3.2.3 angegeben. Die Aufgabe eines Multiagentensystems ist es, eine Infrastruktur für die Kommunikation und Interaktion der Agenten untereinander zur Verfügung zu stellen [HUHNS und STEPHENS, 2000, S. 81].

Mehrere Agenten können innerhalb eines Multiagentensystems miteinander interagieren. Je nach individueller Zielsetzung können die Agenten versuchen, ein gemeinsames Ziel zu lösen, etwa im Rahmen des verteilten Problemlösens oder persönliche Ziele zu erreichen, etwa durch Verhandlungen auf einem virtuellen Marktplatz.

[JENNINGS et al., 1998, S. 17] nennen als wesentliche Merkmale eines MAS:

- Jeder Agent besitzt nur unvollkommene Information und ist in seinen Möglichkeiten eingeschränkt.

- Es gibt keine globale Steuerung des Systems.
- Die Daten sind verteilt.
- Die Berechnung erfolgt asynchron.

Ein Multiagentensystem kann aus mehreren Plattformen bestehen, die (logische oder physikalische) Orte repräsentieren. Plattformen können darüber hinaus den auf ihnen befindlichen Agenten bestimmte Dienste zur Verfügung stellen, wie etwa Kommunikation, Mobilität oder den Zugriff auf bestimmte Ressourcen.

Die Architektur eines Multiagentensystems sagt noch nichts darüber aus, welche Arten von Agenten darin verwendet werden und wie diese miteinander in Verbindung stehen. In dieser Arbeit wird mit dem POTATO-System ein Konzept zur Strukturierung von Multiagentensystemen aufgestellt, das die Entwicklung komplexer, interaktiver Systeme verbessert.

3.2.3 Mulan

Der Aufbau der in dieser Arbeit verwendeten Multiagentensysteme wird durch die von Rölke auf der Basis von Referenznetzen entwickelte MULAN-Referenzarchitektur beschrieben[RÖLKE, 1999, KÖHLER et al., 2001]. Eine Erweiterung der Referenzarchitektur um technische Realisierungsmöglichkeiten stellt CAPA dar (siehe unten).[3] Der Aufbau von MULAN wird im Folgenden beschrieben.

3.2.3.1 Einbettung in bestehende Standards

Die Federation for Intelligent Physical Agents (FIPA [FIPA, 2011b]) ist eine nicht gewinnorientierte Organisation, die zum Ziel hat, Standards für agentenorientierte Systeme zu schaffen. Dazu wurden verschiedene Spezifikationen erarbeitet, die alle Bereiche von Multiagentensystemen umfassen, mit dem Ziel, die Interoperabilität verschiedener agentenbasierter Applikationen zu ermöglichen [NAGI, 2001].

Diese Standards umfassen unter anderem die Bereiche der Architektur von Agentensystemen, das Agentenmanagement und die Kommunikation [FIPA, 2005]. MULAN bildet diese Standardarchitektur mit Hilfe von Referenznetzen nach. Weitere Arbeiten zur Anpassung von MULAN an die FIPA-Standards finden sich in [DUVIGNEAU et al., 2003] unter dem Namen CAPA. Durch diese Erweiterungen ist es möglich, auch verschiedene Agentenplattformen miteinander zu kombinieren und zum Beispiel Agenten auf einer CAPA-Plattform mit Agenten interagieren zu lassen, die auf einer Jadex-Plattform [POKAHR et al., 2003] ausgeführt werden.

[3] Mehr zu Petrinetzen in Abschnitt 3.1, insbesondere zu Referenznetzen in Abschnitt 3.1.3

3.2.3.2 Struktur von Mulan

Mulan besteht aus einer Reihe von Referenznetzen, die verschiedene Teile des Systems darstellen. Über die Verwendung des Konzepts der „Netze in Netzen“ werden Hierarchien zwischen den verschiedenen Ebenen hergestellt. Einen Überblick über den Aufbau zeigt Abb. 3.3. Die Vergrößerungsschritte zeigen jeweils ein Netz einer tieferen Hierarchiestufe als Marke innerhalb des übergeordneten Netzes.

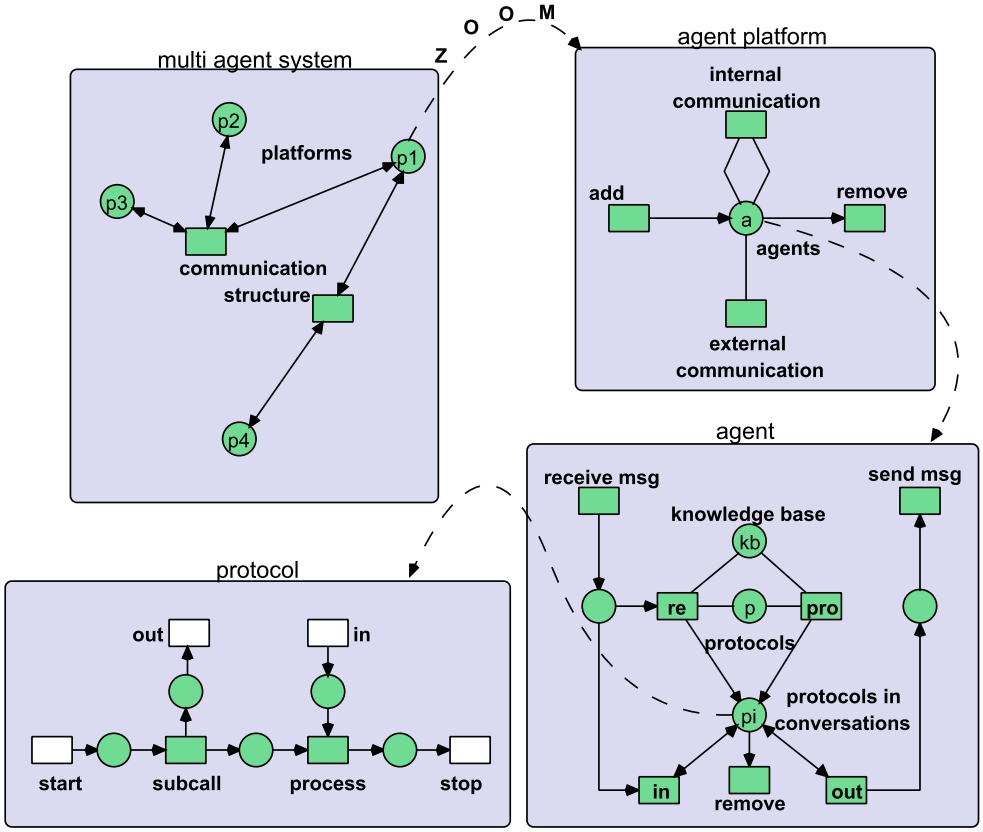

Abbildung 3.3: Abstrakte Architektur des Multiagentensystems Mulan (Aus: [Rölke, 2004, S. 157])

Ein Multiagentensystem als oberste Abstraktionsebene kann aus einer oder mehreren Agentenplattformen bestehen, die miteinander in Verbindung stehen. Eine Plattform wird üblicherweise auf einem Rechner ausgeführt, so dass ein Multiagentensystem als ein verteiltes System betrachtet werden kann. Eine Plattform ist damit auch eine Abstraktion für einen Ort.

Die Hauptaufgabe von Plattformen ist es, bestimmte Dienste für die darauf befindlichen Agenten anzubieten. Dies umfasst hauptsächlich das Erzeugen und Löschen von Agenten sowie die Kommunikation von Agenten innerhalb

der Plattform oder auf anderen Plattformen innerhalb des MAS. Die Agenten auf einer Plattform werden durch Agentennetze implementiert.

Agenten sind wiederum abgegrenzte Einheiten innerhalb der Plattform. Ein Agent besitzt eine Wissensbasis, in der sein lokales Wissen abgelegt ist. Außerdem kann er Protokolle instanziieren, um Handlungen auszuführen. Ein Protokoll, oder Agentenprotokoll, beschreibt dabei jeweils den Anteil, den ein bestimmter Agent an einer Interaktion hat.

Agentenprotokolle sind abgeschlossene Aktionen, die aufgerufen werden (proaktiv oder reaktiv) und einen bestimmten Vorgang bearbeiten. Zusätzlich besitzt ein Agent sogenannte Decision Components (DCs), die für den Agenten Entscheidungen treffen und neue Protokolle starten können (siehe dazu [Cabac, 2010, S. 60]).

Eine Decision Component hat im Allgemeinen kein klar definiertes Ende, sondern läuft so lange, wie der Agent aktiv ist. Sie kann über spezielle Kanäle von Protokollen angesprochen werden, um Entscheidungen zu treffen. Während Agentenprotokolle einfache Reaktionen abbilden, dienen Decision Components als Planungs- und Steuerungskomponenten.

Mulan beinhaltet noch eine Reihe weiterer Netze, auf die hier aber nicht speziell eingegangen wird. Sie sind für die Betrachtung hier nicht wichtig, da sie hauptsächlich Hilfsfunktionen ausführen, wie z.B. das Instanziieren anderer Netze. Auch der Bereich der Verwaltung von Agenten wird hier ausgespart. Es wird lediglich die Interaktion zwischen den Agenten beschrieben.

3.2.3.3 Zusammenspiel der Komponenten

Die Kommunikation zwischen den verschiedenen Ebenen des Multiagentensystems erfolgt über synchrone Kanäle. Jeweils eine Transition im übergeordneten Netz und eine im untergeordneten werden miteinander synchronisiert und schalten gemeinsam, wobei Parameter sowohl von oben nach unten als auch von unten nach oben übergeben werden können. Wenn ein Agentenprotokoll vorsieht, eine Nachricht an einen anderen Agenten zu versenden, wird eine Ausgabetransition im Protokollnetz mit der `:out()`-Transition des Agenten synchronisiert, um die Nachricht an den Agenten zu übergeben. Dieser wiederum synchronisiert sich mit der entsprechenden Transition der Plattform, um die Nachricht zu versenden.

Befindet sich der Adressat der Nachricht auf der gleichen Plattform, wird die Nachricht dann innerhalb der Plattform weitergegeben, andernfalls wird sie eine weitere Ebene nach oben gegeben und innerhalb des Multiagentensystems an die Zielplattform gesendet. Der Empfang der Nachricht passiert dann ebenfalls über synchrone Kanäle, absteigend bis in das angesprochene Agentenprotokoll auf der Empfängerseite.

An einer Interaktion in Mulan sind mehrere Verhaltensprotokolle verschiedener Agenten beteiligt. Ein Agent startet die Interaktion, indem er eine neue Instanz eines Protokolls erzeugt. Dies kann beim Start des Agenten proaktiv

erfolgen, reaktiv angeregt durch ein anderes Protokoll oder auf Impuls einer Decision Component. Auch Benutzereingaben werden über eine Decision Component in den Agenten eingebracht, so dass auch hierdurch proaktiv ein Protokoll gestartet werden kann. Eine neu erzeugte Protokollinstanz synchronisiert sich mit dem Agenten über eine `:start()`-Transition. Anschließend kann das Protokoll die Startnachricht entgegennehmen, mit der es erzeugt worden ist.

Befinden sich die Agenten auf unterschiedlichen Plattformen, wird die Nachricht an eine andere Plattform innerhalb des Multiagentensystems weitergeleitet. Die Plattform des Empfängers gibt die Nachricht wiederum über den synchronen Kanal `:in()` an den Empfängeragenten weiter. Dieser startet ein Protokoll, um die Nachricht zu behandeln. An dieses Protokoll wird dann die Nachricht weitergegeben.

Innerhalb einer Konversation zwischen mehreren Agenten kann über eine `Konversations-ID` eine bereits existierende Protokollinstanz direkt angesprochen werden. Dann wird die Nachricht direkt an das entsprechende Protokoll weitergegeben, ohne dass eine neue Netzinstanz erzeugt wird. Wenn ein Protokoll beendet ist, schaltet im Protokoll die Transition `:stop()` synchron mit einer entsprechenden Stelle innerhalb des Agenten, die dann dafür sorgt, dass die Protokollinstanz beendet und die Referenz darauf im Agenten entfernt wird.

3.2.3.4 Schachtelung von Einheiten

Die verteilte Entwicklungsumgebung dient dazu, die Arbeit der beteiligten Parteien aus verschiedenen Organisationen oder Organisationseinheiten zu unterstützen. Daher muss sie in der Lage sein, die Organisationsstrukturen, die dort zu finden sind, in der Software abzubilden. Dafür wird das Konzept von Einheiten auf Plattformen verwendet. Plattformen sind ein logischer oder physikalischer Ort, auf dem sich verschiedene Einheiten ähnlicher Art befinden können. Dieses Konzept lässt sich auf verschiedenen Ebenen wiederfinden:

Der Referenznetzsimulator RENEW stellt eine Plattform dar, in der verschiedene Einheiten als Netze ausgeführt werden können. Der Simulator dient als Plattform für einen oder mehrere MULAN-Plattformagenten, die darin ausgeführt werden. Diese Plattformagenten dienen ihrerseits als Plattform für weitere Agenten, die darin enthalten sind.

Jeder Agent stellt seinerseits eine Art Plattform dar für die Protokolle, die in ihm ausgeführt werden, wenn auch eine spezielle. Da der Plattformagent seinerseits auch wieder als Agent betrachtet werden kann, können diese Ebenen konzeptionell zusammengelegt werden, so dass theoretisch Agenten beliebig ineinander geschachtelt werden können.

Abb. 3.4 verdeutlicht dieses Prinzip. Zwei Agentenplattformen beherbergen eine Reihe von Agenten. Die Plattformen stellen verschiedene Dienste zur Verfügung, die von den Agenten genutzt werden (einfacher Pfeil); jeweils einer dieser Dienste auf Plattform 1 und 2 wird durch mehrere Agenten der Platt-

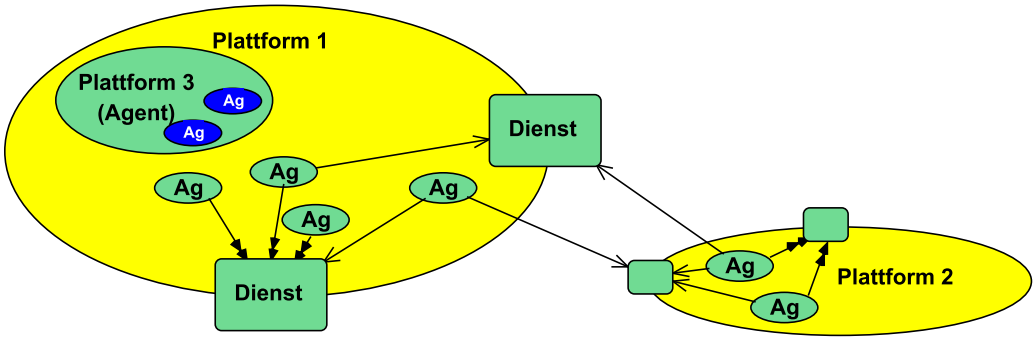

Abbildung 3.4: Plattformen und Agenten als geschachtelte Einheiten

form zur Verfügung gestellt (Doppelpfeil), ein anderer Dienst wird durch die Plattform selbst angeboten. Ein Agent auf Plattform 1 dient seinerseits wieder als Plattform (Plattform 3) für (logisch) darin geschachtelte Agenten, hier in schwarz dargestellt.

Die verteilte Softwareentwicklungsumgebung besteht aus mehreren, über eine Kommunikationsinfrastruktur gekoppelte Plattformen/Agenten, die miteinander interagieren können.

Plattformdienste Plattformen unterscheiden sich darin, welche Art von Diensten sie den Einheiten, die sich auf ihnen befinden, zur Verfügung stellen. Plattformen können den Agenten, die sich darauf befinden, Dienste anbieten, z.B. Kommunikationsdienste für Agenten untereinander und zu Agenten auf anderen Plattformen. Protokolle können die Wissensbasis und Entscheidungskomponenten als Dienste nutzen. Eine Plattform kann aber auch nach außen hin Dienste anbieten, die durch die auf der Plattform befindlichen Agenten bereitgestellt werden.

Ein Beispiel hierfür ist der Dienst der Workflowabwicklung, durch den ein Agent/eine Plattform mittels der darauf befindlichen Agenten/Protokolle zum Workflow Management System (WfMS) wird. Die Dienste der Workflowinstanziierung, Zuweisung von Aufgaben, Prüfung von Rollen und Rechten etc. werden von unterschiedlichen Agenten innerhalb der Plattform erbracht, die dadurch nach außen hin den Dienst eines WfMS anbieten kann.

Helfer und Ressourcen Die Aufgabe einer Entwicklungsumgebung ist es, den oder die Benutzer in der Ausführung ihrer Tätigkeiten zur Entwicklung von Software zu unterstützen. Aus diesem Grund stellt sie eine Reihe von Helfern zur Verfügung.

Auch die Helfer sind als Agenten in einer Umgebung implementiert. Die Arbeitsumgebung eines Benutzers stellt konzeptionell eine Plattform dar, in der durch die sich darauf befindenden Helferagenten verschiedene Dienste erbracht werden. Ressourcen, über die ein Benutzer verfügt, befinden sich als Ressour-

cenagenten auf seiner Plattform. Ein Helfer, der eine Ressource bearbeitet, kann dann über die Plattform darauf zugreifen. Für einen exklusiven Zugriff ist es auch möglich, die Ressource in den Helferagenten, der dann ebenfalls als Plattform agiert, zu verschieben.

3.2.4 Agentenorientierte Softwareentwicklungsmethodiken

Die Agentenorientierte Softwareentwicklung (AOSE) sieht sich als Nachfolger objektorientierter Methoden. Eine Zerlegung der Anwendungsmodelle erfolgt anhand der beteiligten Akteure (Agenten) und ihrer Interaktionen. Üblicherweise findet man in agentenorientiert implementierten Systemen eine Reihe von Agenten, die sich auf verschiedenen Plattformen befinden können. Diese bilden zusammen ein Multiagentensystem und damit ein spezielles verteiltes System.

Auf diese Weise liegt der Fokus von Entwurf und Entwicklung der Software auf der Abbildung der verschiedenen Akteure und ihrer Beziehungen und Interaktionen miteinander. Insbesondere bei Systemen, die organisationale Beziehungen und Abläufe modellieren und unterstützen sollen, hilft dieser Blickwinkel, ein akkurateres Modell zu finden.

Für die Entwicklung agentenorientierter Software sind eine Reihe von Ansätzen entwickelt worden. Einige ausgewählte Ansätze werden im Folgenden vorgestellt, um Gemeinsamkeiten und Unterschiede zum hier verwendeten Paose-Ansatz herauszuarbeiten.

3.2.4.1 Tropos

Tropos [Bresciani et al., 2004] ist eine Methodologie zur Entwicklung von agentenorientierten Softwaresystemen. Ziel ist es dabei, agentenorientierte Konzepte, wie etwa Ziele und Pläne, in allen Phasen der Softwareentwicklung als Leitbild zu verwenden. Tropos basiert auf der BDI-Architektur[4].

Konzepte Die Konzepte, die im Tropos-Ansatz verwendet werden, sind zum größten Teil aus BDI-Architekturen bekannt. Akteure dienen als Oberbegriff für Rollen und Agenten, Agenten verfolgen Ziele, indem sie Pläne erzeugen und ausführen. Ressourcen beschreiben Informationen oder physische Güter, über die ein Agent verfügt, Abhängigkeiten können bezüglich Ressourcen oder Plänen zwischen Agenten bestehen. Schließlich verfügt ein Agent über Fähigkeiten und Überzeugungen, mit deren Hilfe er seine Pläne konstruiert.

[4]BDI steht für Belief-Desire-Intention und beschreibt ein Agentenmodell, dass das Verhalten von Agenten über die Aspekte Beliefs (Annahmen über die Umwelt), Desires (Ziele) und Intentions (Pläne) beschreibt. Entwickelt wurde dieses Modell von [Bratman, 1987].

Anforderungsermittlung Der Tropos-Ansatz legt Wert darauf, bereits in den frühen Phasen der Softwareerstellung, also schon während der Analyse und der Anforderungsermittlung, agentenorientiert vorzugehen. In der Anforderungsermittlungsphase werden die verschiedenen am System beteiligten Parteien identifiziert und als soziale Akteure modelliert. Die Abhängigkeiten zwischen diesen Akteuren in Zielen, Ressourcen und Plänen werden erfasst. Anschließend wird das konzeptuelle Modell um einen neuen Akteur erweitert, der das umgebende System darstellt, um die Abhängigkeiten zur Umgebung zu erfassen.

Entwurf In der Entwurfsphase werden die beschriebenen Anforderungen in eine Spezifikation umgesetzt. Es werden eine Architektur entwickelt, Akteure als Subsysteme definiert und Daten- und Kontrollstrukturen für die Abhängigkeiten der Akteure untereinander beschrieben.

Implementierung In der Implementationsphase wird die Spezifikation dann in ein lauffähiges Agentensystem umgesetzt. Die Methodologie gibt keine spezifische Implementierungsumgebung vor, Besonderheiten eines verwendeten Frameworks müssen daher im Entwurf berücksichtigt werden, um später eine reibungslose Implementierung zu ermöglichen.

3.2.4.2 Prometheus

Das Ziel der Prometheus-Methodologie [Padgham und Winikoff, 2002] ist die Unterstützung praktischer Softwareentwicklung mit agentenorientierten Methoden. Auch dieser Ansatz geht von BDI-Agenten aus, für die Entwicklung steht das JACK-Framework [Jack, 2011] zur Verfügung. Prometheus besteht aus drei Phasen, die in verschiedenen Iterationen ineinander greifen. Abb. 3.5 zeigt diese Ebenen und die darin verwendeten Artefakte, sowie ihre Beziehungen untereinander.

Systemspezifikation In der Systemspezifikation wird festgehalten, was die verschiedenen Funktionsbestandteile des zu erstellenden Systems sind. Für diese Funktionen wird jeweils festgehalten, welche Eingaben (Beobachtungen der Umwelt eines Agenten) sie beeinflussen, welche Aktionen sie auslösen können und welche sonstigen Daten dafür relevant sind.

Architekturentwurf Im Architekturentwurf wird festgelegt, welche Agenten im System existieren sollen und welche Funktionalitäten sie jeweils erfüllen sollen. Wichtige Kriterien hierfür sind die klassischen Modularisierungskriterien eines hohen Zusammenhangs zwischen den Funktionalitäten innerhalb eines Agenten sowie einer möglichst losen Kopplung zwischen den verschiedenen Agenten.

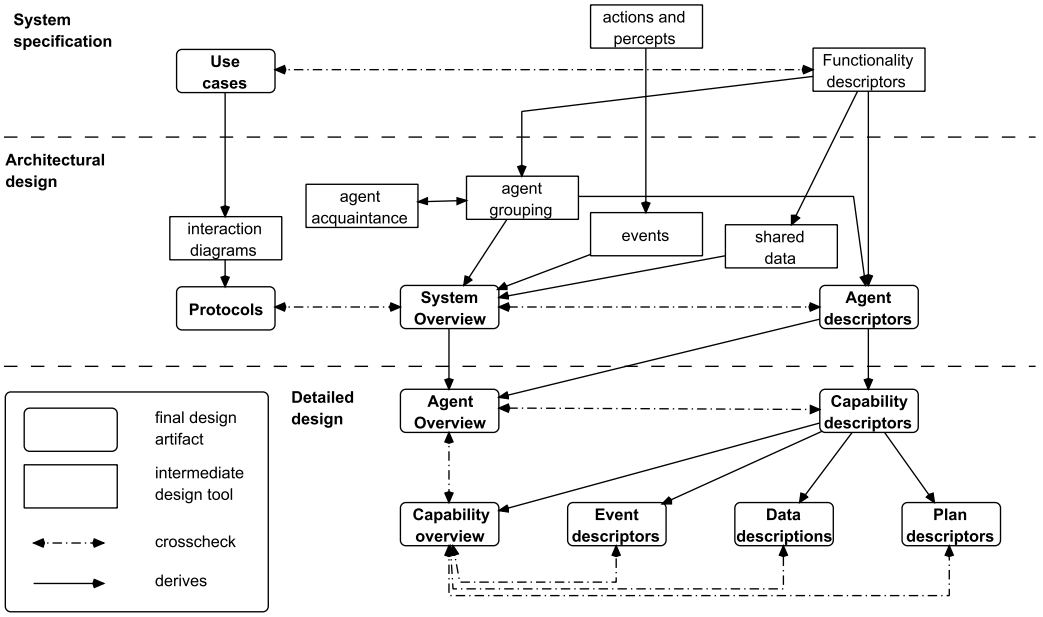

Abbildung 3.5: Phasen des Prometheus Entwicklungsprozesses (Grafik nach [Padgham und Winikoff, 2002, S. 176])

Nachdem die Agenten festgelegt sind, werden diese näher beschrieben. Wieviele Agenten eines bestimmten Typs enthält das System, wie sieht der Lebenszyklus des Agenten aus, wie wird der Agent erzeugt und zerstört? Welche Informationen hält der Agent? Auf welche Ereignisse reagiert er?

Ebenso müssen die benötigten Daten- und Nachrichtenformate, Ereignisse und gemeinsam verwendeten Ressourcen definiert werden. All dies wird in einem Systemüberblicks-Diagramm festgehalten. Als letzter Schritt werden die Interaktionen mit Hilfe von AUML-Interaktionsdiagrammen[5] entworfen und detailliert als Interaktionsprotokolle beschrieben.

Detailentwurf Im Detailentwurf wird der innere Aufbau der Agenten festgelegt. Die verschiedenen Ziele, Pläne und Ereignisse (im Falle eines BDI-Agenten) werden ausformuliert, Module gebildet und in Agentenüberblicks-Diagrammen festgehalten. Diese können wiederum weiter untergliedert werden in detaillierte Beschreibungen der einzelnen Ziele, Pläne und Ereignisse. Parallel zum Entwicklungsprozess wird ein Data Dictionary geführt, in dem die verwendeten Begriffe festgehalten werden, um eine einheitlich verwendete Ontologie sicherzustellen.

[5] Agent-UML: Eine Erweiterung der UML-Modelle um agentenspezifische Modellierungsmöglichkeiten [FIPA, 2011a]

Implementierung Für die Implementierung wird das JACK-Framework empfohlen, das die hier verwendeten Designdokumente unterstützt und an verschiedenen Stellen Konsistenzprüfungen der Modelle erlaubt.

3.2.4.3 Gaia

Von Wooldridge et al. stammt die Gaia-Methodik zur Entwicklung von Multiagentensystemen [Wooldridge et al., 2000, Zambonelli et al., 2003]. Der Ansatz umfasst den kompletten Prozess zur Entwicklung einer Multiagentenanwendung, von der anfänglichen Anforderungsermittlung bis zur Implementierung. Sowohl die Makroebene (organisationale Aspekte), als auch die Mikroebene (Prozesse innerhalb der einzelnen Agenten) werden dabei betrachtet.

Gaia ist nicht auf ein bestimmtes Werkzeug oder eine bestimmte Agentenplattform ausgelegt, sondern beschreibt verschiedene Phasen, die dann jeweils auf die individuellen Bedürfnisse angepasst werden müssen. Abb. 3.6 zeigt die verschiedenen Phasen der Entwicklung und die Dokumente, die jeweils erzeugt werden. Die Anforderungsermittlung sowie die tatsächliche Implementierung sind dabei nicht Teil der Gaia-Methodik im engeren Sinne.

Analyse Der erste Schritt besteht darin, die Ziele zu formulieren, die das Multiagentensystem erfüllen soll. Wenn es sich dabei um eine komplexe Organisation handelt, kann es sinnvoll sein, zunächst eine Aufteilung in überschaubarere Suborganisationen vorzunehmen.

Anschließend wird die Umgebung, in der sich die Agenten bewegen, in Form eines Umgebungsmodells beschrieben. Agenten werden teilweise beschrieben als Entitäten, die eine Umgebung wahrnehmen und beeinflussen können [Wooldridge und Jennings, 1995]. Daher ist es sinnvoll, diese Umgebung durch die Eigenschaften zu spezifizieren, die Agenten davon wahrnehmen können und die Operationen, die sie darin vornehmen können, um sie zu beeinflussen.

Die Agenten, die später das Multiagentensystem bilden, werden nun in einem vorläufigen Rollenmodell festgehalten. Es wird allerdings nur oberflächlich festgehalten, welche Rollen von außen beobachtet werden können, ohne diese näher zu spezifizieren.

Ebenso werden die Interaktionen zwischen den Rollen, die zu diesem Zeitpunkt ersichtlich sind, in einem groben Interaktionsmodell festgehalten. Die Interaktionen werden in Protokollbeschreibungen festgehalten, die die beteiligten Rollen, zwischen diesen ausgetauschte Informationen und eine grobe Beschreibung beinhalten. Die genaue Ausformulierung der Kommunikationsprotokolle erfolgt dann in späteren Phasen.

Die Rahmenbedingungen für die Agenten innerhalb des Systems werden durch organisationale Regeln beschrieben. Diese geben Bedingungen und Invarianten an, die das System als Ganzes einhalten muss, zusätzlich zu Regeln,

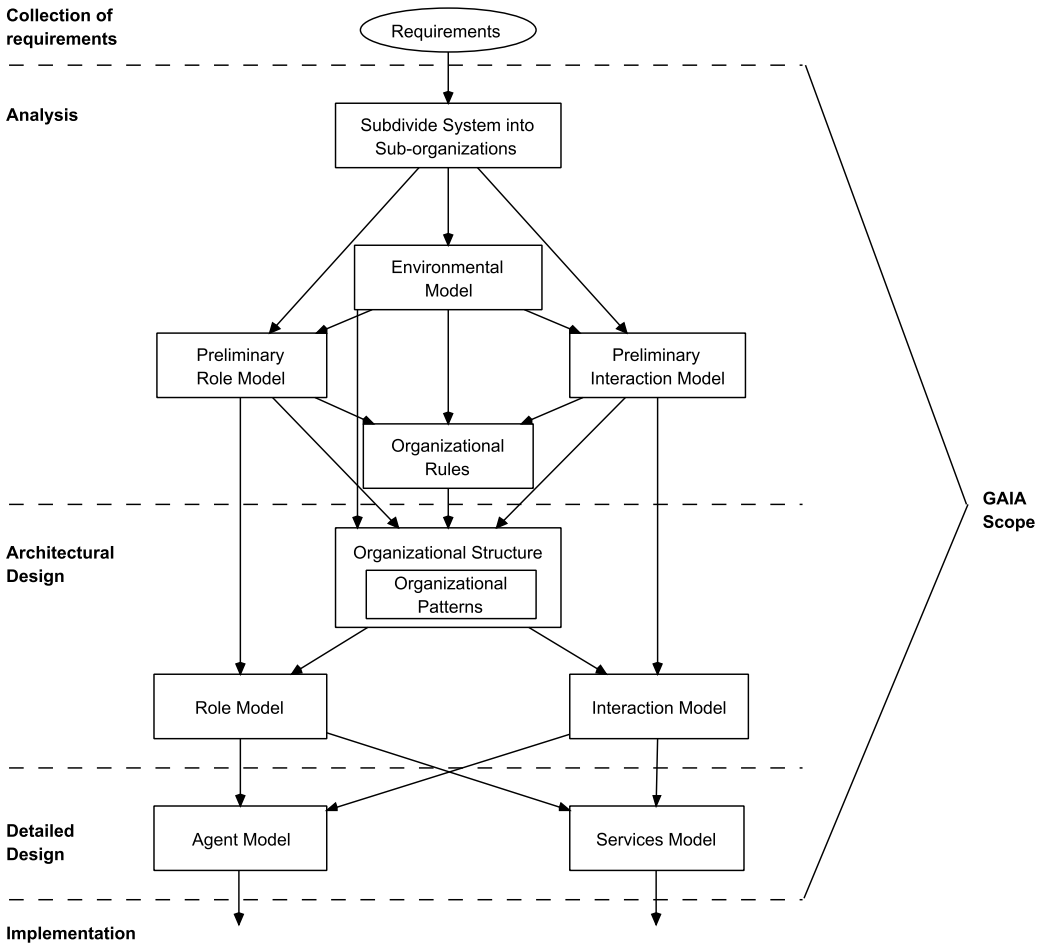

Abbildung 3.6: Verwendete Modelle und ihre Abhängigkeiten im Gaia Entwicklungsprozess (nach [Zambonelli et al., 2003, S. 20])

die für einzelne Rollen gelten.

Architekturentwurf Aus der Beschreibung der Umgebung und der funktionalen Anforderungen an das Multiagentensystem wird im Architekturentwurf versucht, eine vollständige Beschreibung der am System beteiligten Rollen und Interaktionen, sowie der resultierenden Organisationsstruktur zu erstellen.

Zunächst muss die Organisations- und Kontrollstruktur des System festgelegt werden. Diese können sich an der Struktur des unterstützten Realweltsystems orientieren, oder an einer Struktur, die für dieses gewünscht wird, können aber auch unabhängig davon sein. Insbesondere muss die Einhaltung der organisationalen Regeln durch die Kontrollstruktur gewährleistet werden.

An dieser Struktur werden dann die Rollen und Interaktionen ausgerichtet. In dieser Phase können weitere Rollen und Interaktionen identifiziert werden, die sich aus der Struktur ergeben. Für alle Rollen wird definiert, welche Ak-

tivitäten und Dienste sie ausführen, welche Nebenbedingungen für sie gelten und an welchen Interaktionen sie beteiligt sind.

Feinentwurf Die im Architekturentwurf beschriebenen Rollen- und Interaktionsmodelle werden beim Feinentwurf in konkrete Beschreibungen der zu implementierenden Agenten und Dienste überführt. Es wird festgelegt, welche Arten von Agenten im System vorkommen, welche Rollen diese jeweils übernehmen und welche Instanzen davon erzeugt werden müssen.

Die verschiedenen Agenten verfügen nun über eine Menge von Diensten oder Aktivitäten, die sie ausführen können. Aktivitäten lassen sich charakterisieren über ihre Vor- und Nachbedingungen, sowie über Eingabe- und Ausgabeparameter, über die sie mit der Umwelt des Agenten verbunden sind. Über Protokolle können verschiedene Dienste in einen Zusammenhang gebracht werden.

Implementierung Die Implementierung eines konkreten Agentensystems wird nicht durch Gaia beschrieben. Die im Laufe des Entwurfs erzeugten Beschreibungen sollen von Agentenentwicklern verwendet werden, um mit dem jeweils verwendeten Framework eine Anwendung zu entwickeln. Im Fall von offenen Agentensystemen können auch verschiedene Entwickler jeweils ihre eigenen Agenten anhand der Spezifikation entwickeln und in das offene System einbringen.

3.2.4.4 Der Paose-Ansatz

Für die Entwicklung des Potato-Systems und der darauf aufbauenden Entwicklungsumgebung wird der Paose-Ansatz zur verteilten Softwareentwicklung verwendet, der im folgenden Abschnitt vorgestellt wird. Anschließend wird in Abschnitt 3.3.4 ein Vergleich zwischen den verschiedenen Ansätzen gezogen.

3.3 Der Paose-Ansatz der agentenorientierten Softwareentwicklung

Bei der Softwareentwicklung im Rahmen dieser Arbeit wird nach dem Paose-Ansatz (Petrinetz-, Agenten- und Objektorientierte Software-Entwicklung[6] vorgegangen [Moldt, 2005, Moldt, 2006, Cabac, 2010, PAOSE, 2011]. In der Arbeit wird sowohl der Ansatz selbst dargestellt, als auch die Konzepte für eine Software entwickelt, die seine Anwendung unterstützt.

[6] In alternativen Interpretationen steht das P auch für Prozesse oder Personen und das O für Organisationen.

3.3.1 Überblick

Erste Ansätze der agentenorientierten Softwareentwicklung mit Petrinetzen führten zur Entwicklung der Agentenplattform Mulan [Rölke, 1999]. Aufbauend darauf, sowie auf der Erweiterung Capa [Duvigneau, 2002], wurden verschiedene Lehreprojekte zur agentenorientierten Softwareentwicklung durchgeführt (z.B. [Bosch et al., 2002, Willmott et al., 2005]). Aus den Erfahrungen, die im Verlauf mehrerer Lehreprojekte gemacht wurden, wurde der Paose-Entwicklungsansatz formuliert. Dieser Ansatz setzt sich zusammen aus einem Framework (Renew/Mulan/Capa) und dazugehörigen, über die Zeit stetig weiterentwickelten Werkzeugen, einer agentenorientierten Organisationsstruktur des Entwicklungsteams, sowie einem einheitlichen Vorgehen der Entwicklung.

Bei der Entwicklung von Multiagentensystemen mit Capa müssen insbesondere drei Arten von softwaretechnischen Artefakten erzeugt werden:

- Agenten, charakterisiert durch ihre Wissensbasis und ihre Protokolle;
- Interaktionen zwischen den Agenten, spezifiziert in Form von Agentenprotokollen und AUML-Agenteninteraktionsprotokollen;
- Ontologieklassen, die die Konzepte beschreiben, die in dem Multiagentensystem vorkommen und über die sich die Agenten austauschen.

An diesen Artefakten ist die Entwicklungstätigkeit ausgerichtet, und zur Erzeugung dieser Artefakte existieren spezialisierte Werkzeuge [Cabac, 2010]. Die Herausforderung besteht insbesondere darin, die miteinander verzahnten Aspekte des Systems miteinander in Einklang zu bringen.

3.3.2 Teamorganisation

Ein zentrales Konzept im Paose-Ansatz ist die Organisation des Entwicklerteams nach den Konzepten, die im entwickelten Multiagentensystem identifiziert wurden. Dementsprechend erfolgt eine Aufteilung der Aufgaben nach den verschiedenen Agenten/Rollen sowie nach den identifizierten Interaktionen.

Dies führt zu einer Matrixorganisation, bei der in einer Dimension die verschiedenen Agentenrollen auftauchen, in der anderen die Interaktionen. Abb. 3.7 zeigt ein Beispiel für eine solche Matrix. Ein Kreis in der Darstellung bedeutet, dass die entsprechende Rolle an der Interaktion beteiligt ist. Die Entwickler der Rollen müssen sich mit den Entwicklern der Interaktionen abstimmen, an denen ihre Agenten beteiligt sind.

Die Entwicklung und Pflege der Ontologie stellt eine weitere Dimension dar, die ebenfalls orthogonal zu den beiden genannten Dimensionen ist. Bei der Entwicklung verschiedener Module oder bei größeren Projekten muss hier ebenfalls eine sinnvolle Aufteilung in Teilontologien gefunden werden, die nach Möglichkeit die Abhängigkeiten untereinander minimiert.

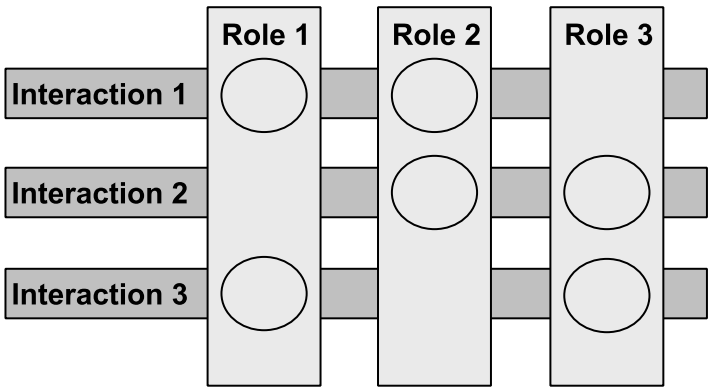

Abbildung 3.7: Matrixorganisation von Rollen und Interaktionen

Bei der Koordination an den Schnittpunkten müssen die jeweiligen Verantwortlichkeiten genau geklärt sein. Im Rahmen der Entwicklung einer Interaktion hat der Ersteller der Interaktion die Verantwortung für die Agentenprotokolle und die Nachrichten, die ausgetauscht werden. Die Verarbeitung dieser Nachrichten und die Entscheidungsfindung innerhalb des Agenten liegt dann im Verantwortungsbereich des Agentenentwicklers, der sich um Wissensbasis und Entscheidungskomponenten kümmert. Auch das Registrieren der Protokolle in der Wissensbasis fällt in seinen Zuständigkeitsbereich.

Es ist jedoch schwierig, die Abgrenzung exakt festzulegen. Oft lassen sich bestimmte Details sowohl auf Agentenseite als auch auf Interaktionsseite behandeln. Hier muss dann im Dialog die passende Lösung gefunden werden. Wobei potentiell eine Lösung gefunden werden muss, ist durch die Matrix jedoch eindeutig beschrieben.

Ein Beispiel, dass in den Lehreprojekten wiederholt behandelt wurde, ist das sogenannte Siedler-Spiel, eine agentenorientierte Adaption des Brettspiels „Die Siedler von Catan“ ([Teuber, 1995, Bosch et al., 2002, Cabac et al., 2005, Cabac, 2010]). Es wurden unter anderem Agenten entwickelt für den Spieler, für die Insel, die Bank, die die Ressourcen verwaltet und die Spielleitung. Eine Interaktion in diesem Spiel ist beispielsweise der Bau einer Siedlung. Hierfür muss die Insel befragt werden, ob der Bau einer Siedlung an dieser Stelle durch diesen Spieler erlaubt ist (Verbindung zu vorhandenen Siedlungen, Abstand zu anderen Siedlungen); die Bank muss sicherstellen, dass der Spieler über die nötigen Ressourcen verfügt und die Spielleitung, dass er überhaupt an der Reihe ist zu bauen.

Bei der Entwicklung dieser Interaktion war daher die Gruppe federführend, die die Verantwortung für die Siedlungsbau-Interaktion übernommen hatte. Sie musste sich aber laufend abstimmen mit den Gruppen, die für die verschiedenen Agenten verantwortlich waren und dafür sorgen, dass die richtigen Schnittstellen definiert und eingehalten wurden. Die Agentengruppen mussten

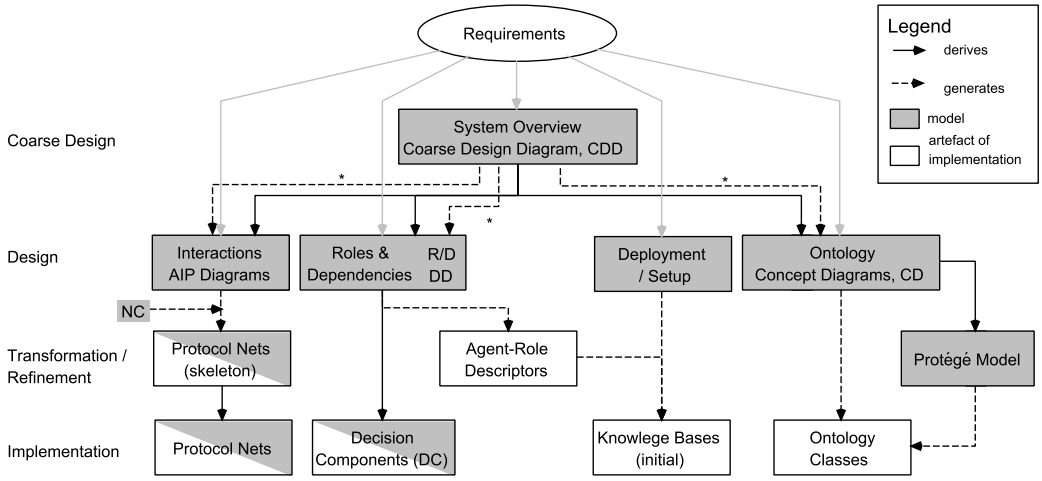

Abbildung 3.8: Verwendete Modelle im PAOSE-Entwicklungsprozess (aus [CABAC, 2010, S. 134])

dann dafür sorgen, dass die Agenten ihre jeweiligen Teilprotokolle ausführen und die richtigen Entscheidungen treffen konnten. Zusätzlich war eine Gruppe für die Pflege der gemeinsamen Ontologie zuständig und musste gegebenenfalls ebenfalls einbezogen werden.

3.3.3 Modelle, Diagramme und Werkzeuge

Bei der Softwareentwicklung nach dem PAOSE-Ansatz werden eine Reihe verschiedener Modelle und Diagramme erstellt, die miteinander verzahnt sind und in RENEW oder anderen Werkzeugen bearbeitet werden können. Abb. 3.8 zeigt die verschiedenen Modelle und ihre Abhängigkeiten untereinander. Hier werden nur einige dieser Modelle beschrieben, mehr dazu findet sich in [CABAC, 2010].

3.3.3.1 Grobentwurfs-Diagramm

Die generelle Struktur einer Multiagentenanwendung wird durch ein Grobentwurfs-Diagramm beschrieben [CABAC und MARKWARDT, 2009]. Dieses Diagramm stellt in einer ähnlichen Form wie ein UML Use-Case-Diagramm die im entworfenen System vorkommenden Rollen dar und an welchen Interaktionen sie teilhaben. Auf diese Weise definiert es die Matrix der Rollen und Interaktionen, die die Basis der Teamorganisation darstellt.

Das RENEW-Werkzeug kann verwendet werden, um dieses Diagramm zu erstellen. Zusätzlich können für die hier definierten Rollen und Interaktionen die nötigen Dateien und Verzeichnisse in einer initialen Form erzeugt werden.

3.3.3.2 Rollen- und Abhängigkeitsdiagramm

Innerhalb des Multiagentensystems existieren eine Reihe von Agenten, die verschiedene Rollen einnehmen. Das Rollendiagramm entspricht gewissermaßen einem Klassendiagramm für Agentenrollen, in dem Rollenhierarchien definiert werden können. Gleichzeitig werden hier die Wissensbasiseinträge und Entscheidungskomponenten festgehalten, die für die Rolle benötigt werden.

Mit Hilfe des Knowledge Base Editors (KBE) [Klenski und Willner, 2007] werden die in einem Multiagentensystem vorkommenden Rollen bearbeitet. Rollenhierarchien und Wissensbasen können in einer graphischen Oberfläche editiert werden. Zusätzlich kann der KBE auch zum Aufsetzen eines Systems zu Testzwecken verwendet werden.

3.3.3.3 Agenteninteraktionsdiagramme und Agentenprotokolle

Eine Interaktion in Mulan besteht aus mehreren Agentenprotokollen, die miteinander interagieren. Jeder Agent kennt seinen Teil der Interaktion, welche Nachrichten er sendet, und welche er von anderen Agenten erwartet. Um Inkompatibilitäten zu vermeiden, wird zunächst die gesamte Interaktion mit allen beteiligten Rollen in einem Agenteninteraktionsprotokoll (AIP) entworfen. Dieser Diagrammtyp ist Teil der AUML-Diagramme, einer agentenorientierten Erweiterung von UML.

Ähnlich einem normalen Interaktionsdiagramm werden hier die verschiedenen beteiligten Partner einer Interaktion festgehalten. Für jede Rolle werden dann die Aktionen im Verlauf der Interaktion und die Arten von Nachrichten festgehalten, die an andere Rollen versandt werden. Zusätzlich existieren Konstrukte für Verzweigungen und Schleifen innerhalb der Interaktion.

Aus dem AIP-Diagramm werden dann Agentenprotokolle automatisch generiert, indem jeweils der Anteil einer Rolle extrahiert und in ein Agentenprotokoll konvertiert wird. Dabei werden die Elemente des AIP in Netzkomponenten im Agentenprotokoll übertragen und die Nachrichtentypen, die im AIP angegeben worden sind, an die Ein- und Ausgangskanäle der Protokolle geschrieben. Abbildung 3.9 zeigt im oberen Bereich ein einfaches Agenteninteraktionsprotokoll und unten die daraus generierten Protokollnetze[7]

Ein Renew-Plugin setzt das Interaktionsdiagramm in Agenteninteraktionsprotokolle um, je ein Protokollnetz je beteiligter Rolle. Für die verschiedenen Aktionen der Interaktion (z.B. Nachricht senden/empfangen, Entscheidung, etc) werden Netzkomponenten [Cabac et al., 2003] eingesetzt. Diese können dann in der späteren Bearbeitung angepasst werden, indem beispielsweise die versendeten Nachrichten oder die getroffenen Entscheidungen genauer spezifiziert werden.

[7]Die Anschriften in den Protokollnetzen sind hier nicht wichtig, die Grafik zeigt die Struktur der Protokollnetze im groben Überblick

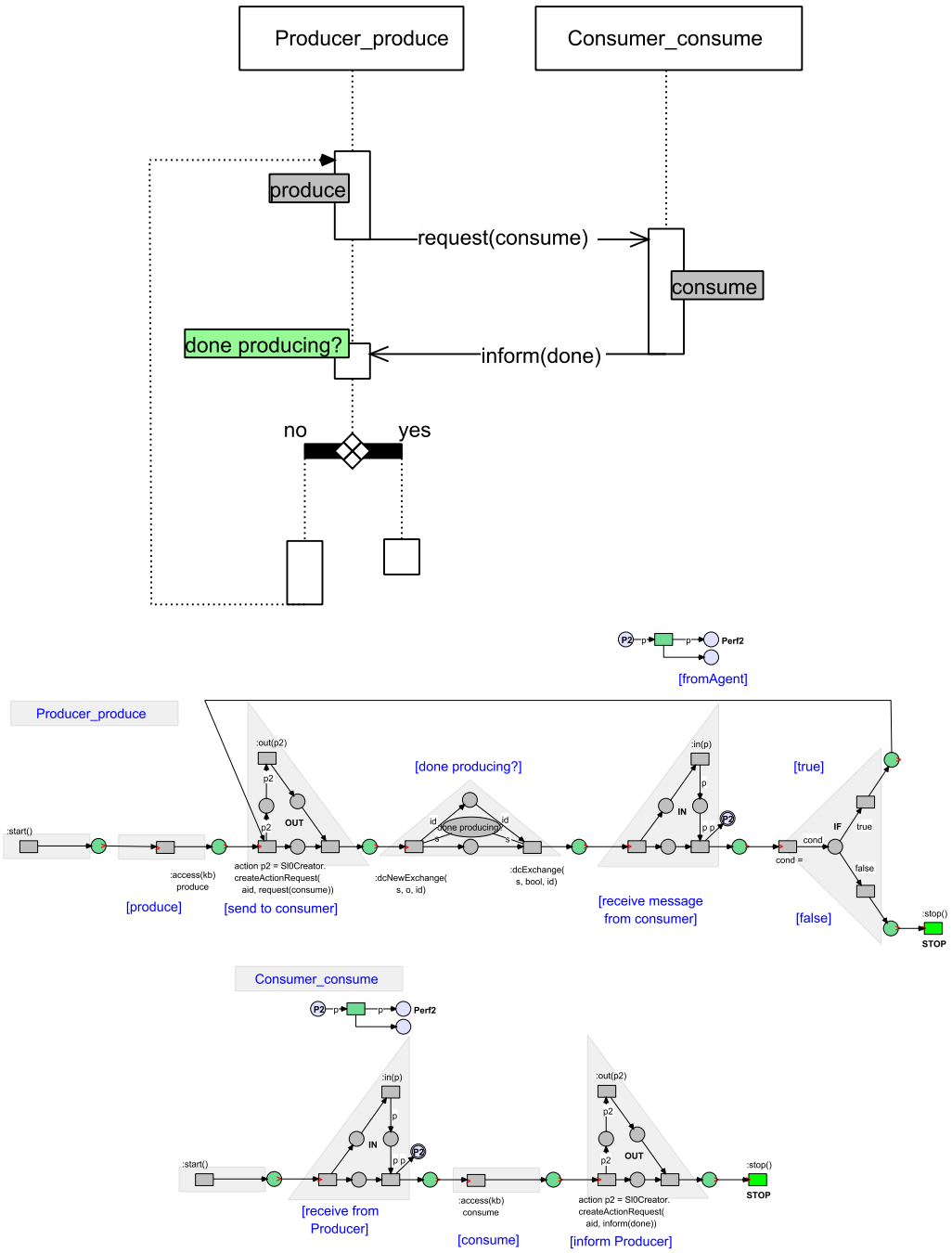

Abbildung 3.9: Agenteninteraktionsdiagramm und generierte Protokollnetze für das Producer/Consumer-Beispiel

Die Agentenprotokolle sind der „Programmcode“, der letztlich von den Agenten ausgeführt wird. In der Regel sind nach der Generierung noch weitere Bearbeitungen nötig. In [Dirkner, 2006] werden Ansätze entwickelt, diese Bearbeitungen mit dem Ausgangsmodell im Sinne eines Roundtrip-Engineering konsistent zu halten.

3.3.3.4 Ontologiediagramm

In der Ontologie werden die Konzepte festgehalten, die für die jeweilige Multiagentenanwendung von Bedeutung sind und die die Agenten miteinander kommunizieren. Ähnlich einem Klassendiagramm werden hier Hierarchien und Abhängigkeiten zwischen Konzepten, sowie ihre Attribute eingetragen.

Zur Modellierung dieser Konzepte in Paose wird das Werkzeug Protégé [Protégé, 2011] verwendet. Aus den Konzepten werden dann Java-Klassen erzeugt, die bei der weiteren Entwicklung in Referenznetzen verwendet werden können.

Alternativ können zur Modellierung der Ontologiekonzepte Feature-Structure-Netze [Duvigneau et al., 2006] verwendet werden. Mit diesen Netzen können Konzeptstrukturen gut dargestellt werden. Die Bearbeitung dieser Netze kann mit einem entsprechenden Plugin in Renew vorgenommen werden.

3.3.3.5 Monitoring und Debugging

Ein weiterer Aspekt der Entwicklung mit Capa sind die verschiedenen Werkzeuge zur Überwachung und zum Debugging des entwickelten Systems. Hierzu dient vor allem der MulanViewer (Abb. 3.10), in dem sich das Multiagentensystem betrachten lässt. Er zeigt die laufenden Agenten mit ihren Protokollen und Wissensbasen an und erlaubt auch die Manipulation zur Laufzeit [Cabac et al., 2008].

Der linke Bereich zeigt das Multiagentensystem mit den darin befindlichen Agenten. Zu den Agenten können die Entscheidungskomponenten und derzeit laufenden Protokolle angezeigt werden. Im rechten Bereich findet sich eine Detailansicht der auf der linken Seite ausgewählten Elemente. Im Beispiel wird der Inhalt der Wissensbasis eines Agenten angezeigt. Von hier aus können auch die Referenznetze für die Agenten und Protokolle untersucht werden, um beispielsweise Fehler im Schaltverhalten aufzuspüren.

Im MulanViewer lassen sich die verschiedenen Protokoll- und Agentennetze auswählen, die gerade aktiv sind. Diese können dann in Renew geöffnet werden. Renew erlaubt es, Netze zur Laufzeit zu betrachten, die Marken zu untersuchen und bei Bedarf auch im Einzelschrittmodus Transitionen zu schalten, um Fehlern nachzuforschen. Mit Hilfe eines Sniffers können laufende Interaktionen im Multiagentensystem aufgezeichnet und untersucht werden. Damit kann beispielsweise ein Process Mining betrieben werden, um die im

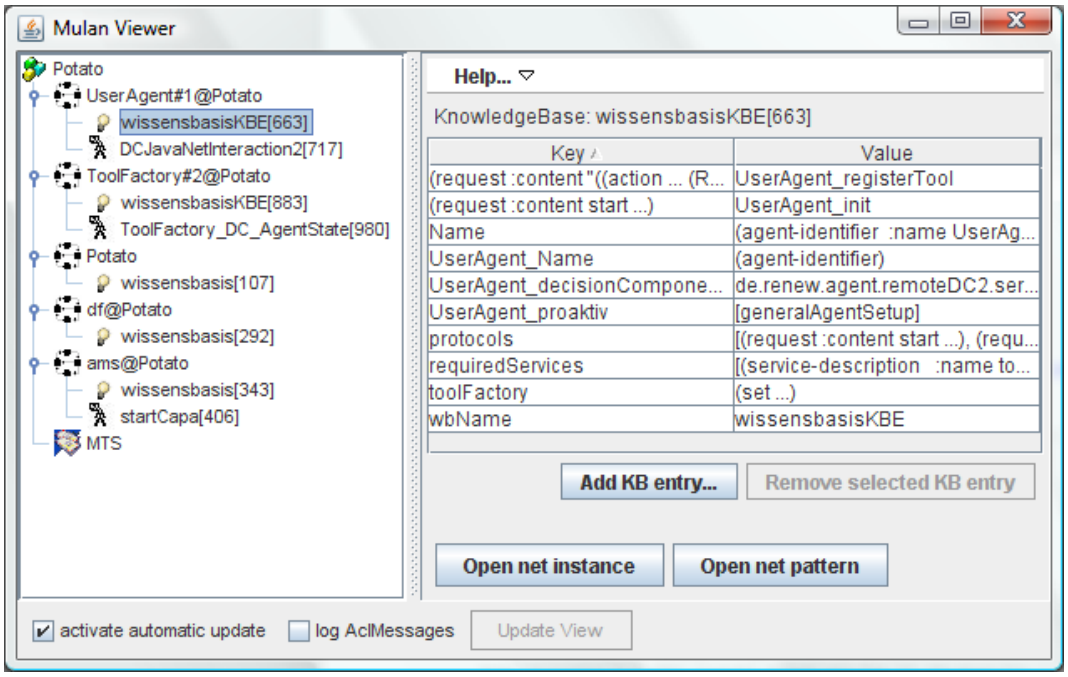

Abbildung 3.10: Untersuchung eines laufenden Multiagentensystems mit dem MulanViewer

System ausgeführten Prozesse zu analysieren. MulanViewer und Sniffer werden beschrieben in [Cabac und Dörges, 2007].

Oft ist es erforderlich, Unterschiede zwischen verschiedenen Modellen oder Modellversionen zu untersuchen, etwa um herauszufinden, was sich zwischen einer funktionierenden und einer nicht mehr funktionierenden Version eines Agentenprotokolls geändert hat. Hierfür kann ein grafischer Vergleich der Diagrammversionen hilfreich sein. Hierfür hilft das Werkzeug ImageNetDiff [Cabac et al., 2009a], das es ermöglicht, Unterschiede in grafischen Modellen sichtbar zu machen.

3.3.4 Vergleich mit anderen AOSE-Methodiken

Wie in Abschnitt 3.2.4 ausgeführt, gibt es eine Reihe verschiedener agentenorientierter Softwareentwicklungsansätze. Hier soll nun ein Vergleich des Paose-Ansatzes mit diesen anderen Ansätzen vorgenommen werden, um aufzuzeigen, warum gerade dieser Ansatz im Rahmen der vorliegenden Arbeit verwendet wird.

3.3.4.1 Grundsätzliches

Tropos und Prometheus sind Entwicklungsmethodiken, die insbesondere für die Entwicklung von BDI-Agenten geeignet sind. Petrinetzagenten können

ebenfalls deliberative Komponenten enthalten, der Fokus der Entwicklung liegt aber mehr in der Verwendung von Agenten als softwaretechnischem Strukturierungsmerkmal.

Der Ansatz ähnelt dem Gaia-Ansatz in einigen Punkten. Auch bei PAOSE wird das Agentensystem durch die Agentenrollen und ihre Interaktionen beschrieben.

3.3.4.2 Entwicklungsprozess

Im PAOSE-Ansatz wird der komplette Softwareentwicklungsprozess abgebildet, von der Anforderungsermittlung bis zur Implementierung. Im Vergleich dazu ist bei Gaia die Anforderungsermittlung und Implementierung nicht Teil der Methodologie, bei Tropos ist die Beschreibung des Implementationsprozesses knapper gehalten, da nicht auf ein spezifisches Werkzeug Bezug genommen wird. Prometheus stellt ebenfalls eine Methodologie für den kompletten Entwicklungsprozess dar, der ebenfalls durch entsprechende Werkzeuge und Modelle unterstützt wird.

Mit Petrinetzen wird ein Formalismus verwendet, der üblicherweise hauptsächlich für die Systemmodellierung und nicht für die Implementierung verwendet wird. Die durchgehende Verwendung dieses Formalismus auch für die Implementierung selbst erlaubt einen fließenden Übergang zwischen Entwurf und Implementierung. Die verschiedenen verwendeten Modelle, z.B. Agenteninteraktionsdiagramme und Protokollnetze, lassen sich teilweise ineinander überführen (vgl. [DIRKNER, 2006]), was ein iteratives Vorgehen unterstützt. Leider ist die Softwareunterstützung für dieses Roundtrip-Engineering noch unvollständig, so dass es meist immer noch erforderlich ist, Änderungen an den verschiedenen Modellen separat vorzunehmen.

3.3.4.3 Strukturierungsschwerpunkte

Bei der Entwicklung mit PAOSE liegt der Hauptschwerpunkt auf den drei Entwurfsdimensionen Agentenrollen, Interaktionen und Ontologie. Diese orthogonalen Komponenten definieren das Agentensystem und auch die Strukturierung der Entwicklungsaktivitäten und -teams.

3.3.4.4 Agenten

Agenten in einem CAPA-Multiagentensystem sind gekennzeichnet durch drei Aspekte: Die Wissensbasis, in der Fakten- und Methodenwissen gespeichert ist, Protokolle für die verschiedenen Interaktionen, an denen der Agent beteiligt sein kann, sowie Entscheidungskomponenten für Planungs- und Entscheidungsprozesse. Ein Agent kann dabei Wissen, Protokolle und Entscheidungskomponenten für verschiedene Rollen enthalten, die er ausfüllt.

3.3.4.5 Prozesse

Ein wichtiger Schwerpunkt bei der PAOSE-Entwicklung liegt auf dem Entwurf und der Implementation der Interaktionsprozesse zwischen den verschiedenen Agenten. Dies ähnelt dem Gaia-Ansatz, bei dem ebenfalls die Interaktionen einen hohen Stellenwert einnehmen. Tropos und Prometheus legen den Fokus eher auf die Pläne und Aktionen einzelner Agenten, aus denen dann indirekt Interaktionen entstehen.

3.3.4.6 Werkzeugunterstützung

Der PAOSE-Entwicklungsprozess wird durch eine Reihe von Werkzeugen unterstützt, in der Hauptsache RENEW-Plugins für die Modellierung, Implementierung und Untersuchung von Multiagentensystemen in MULAN. Tropos ist ausgelegt auf die Verwendung des JACK-Systems, das die hierbei verwendeten Modelle und Prozesse unterstützt, während Prometheus und Gaia weitgehend werkzeugneutral sind und für die jeweils verwendete Multiagentenplattform angepasst werden müssen.

3.4 Computer Supported Cooperative Work

In diesem Abschnitt werden verschiedene Ansätze zur Rechnergestützten Gruppenarbeit (Computer Supported Collaborative Work - CSCW) aufgezeigt. Während Programmierung häufig eine Aktivität ist, die effektiv von einzelnen Personen durchgeführt werden kann, ist Softwaretechnik, also die ingenieursmäßige *Konstruktion* größerer Softwaresysteme eine gruppenorientierte Aktivität [GHEZZI et al., 2002, S. 1], die eine sorgfältige Abstimmung der verschiedenen Parteien erfordert. Dies gilt umso mehr, je größer und komplexer die entwickelten Systeme werden.

Aus diesem Grund bietet es sich an, Softwareentwicklungsumgebungen unter dem Aspekt der rechnergestützten Zusammenarbeit (CSCW) zu betrachten. Die in dieser Arbeit betrachtete verteilte Entwicklungsumgebung kann damit letztlich als ein Spezialfall einer CSCW-Anwendung für einen spezialisierten Zweck betrachtet werden, nämlich die Koordination und Unterstützung von Softwareentwicklungsteams.

3.4.1 Begriffsklärung

Rechnergestützte Gruppenarbeit befasst sich mit der Kooperation zwischen Menschen sowie mit Theorien, Methoden und praktischen Werkzeugen zur Unterstützung derselben durch Computer. Sie ist ein interdisziplinäres Forschungsgebiet insbesondere aus den Gebieten Informatik, Soziologie und Psychologie. Konkrete Werkzeuge zur Umsetzung von CSCW werden als Groupware bezeichnet [ELLIS et al., 1991, KLUSSMANN, 2000].

Dabei wird für eine Gruppe von Personen mit einem gemeinsamen Ziel bzw. einem gemeinsamen Kontext (z.B. innerhalb einer Firma), die geographisch verteilt sein kann, eine verteilte Arbeitsumgebung geschaffen. Ziele sind eine verbesserte Kommunikation, Kooperation, Koordination und Informationsaustausch der beteiligten Personen, um die soziale Interaktion effektiver, effizienter und angenehmer zu gestalten [Gross und Koch, 2007, S. 4ff].

Hierbei sind sowohl Interaktionsprozesse zwischen den beteiligten Personen, als auch technische Rahmenbedingungen zu beachten. Organisationsstrukturen, soziale Gefüge und kulturelle Besonderheiten sind zu berücksichtigen [Ellis et al., 2005, S. 38]. Eine Möglichkeit der Modellierung von Benutzerinteraktionen mit Netzen bieten Rollen-, Funktions- und Aktionsnetze (RFA-Netze)[Oberquelle, 1987].

3.4.2 Teilbereiche

Die wesentlichen Aspekte, in denen Groupware-Systeme Benutzer in ihrer Arbeit unterstützen, sind Kommunikation, Koordination und Kooperation [Gross und Koch, 2007, S. 8]. Im Folgenden werden diese einzelnen Aspekte zunächst unter einem generellen Groupware-Blickwinkel betrachtet und dann speziell auf die Relevanz für den Anwendungsbereich der verteilten Softwareentwicklung eingegangen.

3.4.2.1 Kommunikation

Die Basis für jede Art von Zusammenarbeit ist Kommunikation und natürlich basiert auch die weitergehende Zusammenarbeit auf der Grundlage der Kommunikation. [Gross und Koch, 2007, S. 79ff] unterteilen bei Groupware zur Kommunikationsuntertützung in direkte und indirekte, sowie in synchrone und asynchrone Kommunikation.

Bei direkter Kommunikation steht der Sender in direktem Kontakt mit dem Empfänger, weiß also genau, an wen er eine Nachricht übermittelt. Bei indirekter Kommunikation werden Informationen kategorisiert abgespeichert und können von den „Empfängern“ über entsprechende Zugriffsverfahren ausgelesen oder an diese übermittelt werden. Synchrone Kommunikation bezeichnet ein Kommunikationsverhalten, bei dem die Nachrichten in Echtzeit zwischen Sender und Empfänger ausgetauscht werden (z.B. Chatprogramme oder Videokonferenzen), während dies bei asynchroner Kommunikation zeitversetzt passiert (z.B. Email).

Abb. 3.11 veranschaulicht den Unterschied zwischen synchroner und asynchroner Kommunikation. Im asynchronen Fall (oben) erfolgt die Kommunikation über einen Nachrichtenpuffer, auf den jeder Beteiligte zu einem beliebigen Zeitpunkt zugreifen kann, unabhängig vom Kommunikationspartner. Im synchronen Fall werden die Kommunikationstransitionen miteinander synchronisiert (hier über ein Kommunikationsmedium, das die Interaktion ermöglicht)

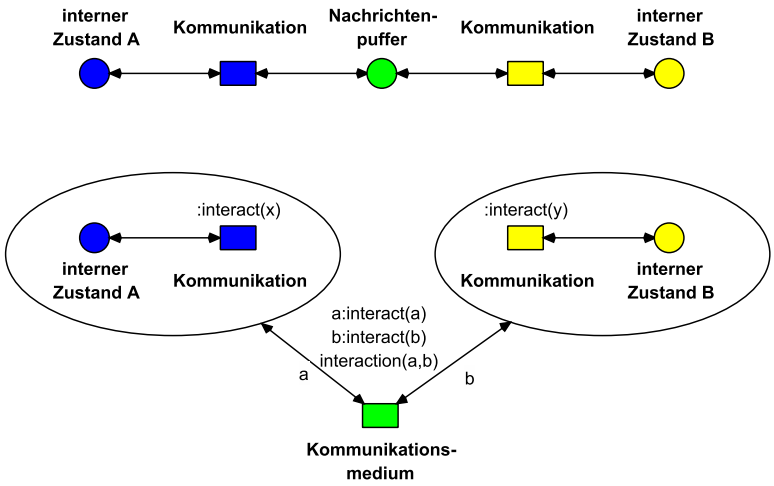

Abbildung 3.11: Asynchrone (oben) vs. synchrone Kommunikation

und schalten gemeinsam. Hierfür müssen die Partner zur selben Zeit aktiv werden. Der Vorteil ist meist eine direktere Interaktion.

Im Bereich der Softwareentwicklung, insbesondere wenn sich die beteiligten Parteien an verschiedenen Orten befinden, ist die Kommunikation einer der, wenn nicht der wichtigste Faktor für den Erfolg eines Projektes. Angemessene Kommunikationswerkzeuge müssen daher Teil einer verteilten Entwicklungsumgebung sein. Dies kann von einfachen Chat- [Willmott et al., 2005] oder Mailboxsystemen bis hin zu Videokonferenzen und virtuellen Meetings [Boulila et al., 2003] reichen.

3.4.2.2 Koordination

Unter Koordination versteht man das harmonische Zusammenarbeiten verschiedener Akteure. Akteure führen bestimmte Aktivitäten aus, die auf bestimmte Ziele ausgerichtet sind und zwischen denen Abhängigkeiten bestehen können. Ziel der Koordination ist es, wünschenswerte Ergebnisse zu erreichen und unerwünschte zu vermeiden. Dazu werden die Abhängigkeiten zwischen den Aktivitäten untersucht. Nach [Malone und Crowston, 1993] lassen sich die Abhängigkeiten zwischen Aktivitäten in drei Typen einteilen:

- Erzeuger/Verbraucher-Abhängigkeit, bei der die Ergebnisse einer Aufgabe Voraussetzung für die nächste sind;
- Gemeinsame Ressourcen, die von verschiedenen Aktivitäten benötigt werden und nur begrenzt zur Verfügung stehen, so dass die Aktivitäten sich im wechselseitigen Ausschluss befinden;
- Gleichzeitige Aktivitäten, die nur gemeinsam stattfinden können, beispielsweise ein Treffen mit mehreren Teilnehmern;

Bei der Softwareentwicklung lassen sich eine Reihe von Abhängigkeiten zwischen verschiedenen Aktivitäten identifizieren, die eine Koordination erforderlich machen. Aufgaben sind auf vorher zu erledigende Aufgaben angewiesen, z.B. muss eine Software erst implementiert werden, bevor sie getestet werden kann. Gemeinsam genutzte Ressourcen betreffen hauptsächlich Personen, die nur eine Aufgabe zur Zeit bearbeiten können, aber auch beispielsweise Softwarelizenzen oder bearbeitete Dokumente (sofern nicht Mechanismen existieren, die eine nebenläufige Bearbeitung ermöglichen). Synchronisation von Aktivitäten ist erforderlich, wenn mehrere Beteiligte miteinander direkt interagieren müssen, etwa in einem Abstimmungsmeeting oder beim kollaborativen Bearbeiten eines Dokumentes.

3.4.2.3 Kooperation

Wie erwähnt, ist es das Hauptziel eines CSCW-Systems üblicherweise, die Zusammenarbeit verschiedener Beteiligter zu einem gemeinsamen Ziel zu ermöglichen. Dafür können eine ganze Reihe verschiedenster Werkzeuge verwendet werden. Man kann unterscheiden zwischen synchroner und asynchroner Gruppenarbeit, je nachdem ob die beteiligten Personen in Echtzeit kooperieren oder zu verschiedenen Zeiten [Gross und Koch, 2007, S. 103ff].

Asynchrone Kooperationsunterstützung umfasst Werkzeuge wie Entscheidungsunterstützungssysteme, gemeinsame Arbeitsbereiche und Teamräume, Knowledge Management-Systeme oder asynchrone Editoren. Synchrone Werkzeuge sind unter anderem synchrone Editoren zur gleichzeitigen Bearbeitung von Dokumenten und verschiedene Kommunikationswerkzeuge (Chat, virtuelle Meetings).

Viele dieser Werkzeuge können die Softwareentwicklung unterstützen. Bei gemeinsamen Editoren liegt der Nutzen auf der Hand, Meetingsysteme können die erforderlichen Kommunikationsprozesse auf allen Stufen des Entwicklungsprozesses unterstützen, von der Konzeption bis zum Test. Aber auch Knowledge Management-Systeme und andere Plattformen zum Austausch von Informationen können die Effektivität und Effizienz der Softwareentwicklung deutlich steigern.

Häufig wird jedoch das Hauptaugenmerk bei der Entwicklung von Groupware Systemen nicht auf applikationsspezifische Lösungen gelegt, sondern eher auf generelle Kommunikationswerkzeuge. Gründe dafür sind insbesondere der damit verbundene Aufwand für spezialisierte Anwendungsfelder sowie oft fehlender Einblick in die spezifischen Probleme [Pankoke-Babatz, 2003, S. 15].

3.4.3 Technologien im CSCW

Oben wurde Groupware definiert als Software zur Unterstützung rechnergestützter Zusammenarbeit. Im weitesten Sinne umfasst dies also alle Arten von Software, die eine Zusammenarbeit unterstützen kann. Das reicht von gene-

rischen Anwendungen wie E-Mail oder Instant-Messaging bis zu Multimedia-Anwendungen wie Video-Conferencing. Dazu kommen spezielle, oft anwendungsspezifische Anwendungen, etwa zur gemeinschaftlichen Bearbeitung von Dokumenten oder zum Austausch strukturierter Informationen.

Workflow Management Systeme (WfMS) werden meist nicht direkt zur Groupware gezählt, weil man unter Groupware meist Systeme versteht, die weniger stark strukturiert sind und eine informalere Interaktion unterstützen sollen [Klussmann, 2000]. WfMS dienen aber natürlich auch der rechnergestützten Zusammenarbeit.

Häufig findet man in Groupware-Systemen gemeinsam genutzte Arbeitsbereiche oder Gruppenräume, in denen Dokumente abgelegt und gemeinsam genutzt werden können und Informationen ausgetauscht werden. Weitere Features sind z.B. Wikis zum gemeinschaftlichen Bearbeiten und Sammeln von Informationen, Terminkalender, Mailinglisten und Diskussionsforen, häufig in Form von Intranet- und Internet-Anwendungen [Gross und Koch, 2007, S. 10].

Aufgrund der verteilten Natur von Groupware und dem Fokus auf die Unterstüzung einzelner Personen ist aber auch die Agententechnologie ein Ansatz bei der Entwicklung von CSCW-Anwendungen. Die Benutzer werden im System durch einen User-Agent vertreten und können mit den Agenten anderer Benutzer interagieren. Das erlaubt eine intuitive und ausdrucksstarke Modellierung der Personen und ihrer Interaktionen [Gross und Koch, 2007, S. 144ff]. Mehr zum Thema Usability-Engineering, auch für Groupware-Systeme findet sich in [Finck, 2007].

3.5 Der Werkzeug und Material Ansatz (WAM)

Der Werkzeug und Material Ansatz (WAM) [Züllighoven, 2004] beschreibt einen am Arbeitsbereich Softwaretechnik der Universität Hamburg entwickelten, anwendungsorientierten Ansatz zur Softwareentwicklung, bei dem besonderes Augenmerk auf die Identifizierung und softwaretechnische Abbildung von Materialien und Werkzeugen des Anwendungskontextes gelegt wird. Das „A“ lässt sich als Automaten oder Aspekte interpretieren [Gryczan, 1996, S. 111,119,221]. Eine Formalisierung dieses Ansatzes findet sich in [Becker-Pechau et al., 2006]. Für die softwaretechnische Unterstützung dieses Ansatzes wurde das JWAM-Framework entwickelt [Breitling et al., 2000].

Das Hauptaugenmerk liegt bei diesem Ansatz meist auf der Unterstützung von Expertenarbeitsplätzen zur selbstständigen Bearbeitung komplexer Aufgaben. Ein großer Teil des Ansatzes orientiert sich an dem Leitbild eines Experten, der ähnlich einem Handwerksmeister sehr genau weiß, mit was für Aufgaben er es zu tun hat und wie diese gelöst werden können. Er benötigt lediglich den Zugang zu den optimalen Werkzeugen, um seine Aufgaben zu erledigen, aber keine Vorschriften, wie er das zu tun hat.

Im Folgenden werden zunächst die Konzepte und das Vorgehen des WAM-

Ansatzes erläutert. Anschließend wird der Bereich der Prozess- und Kooperationsunterstützung in WAM genauer beleuchtet, da dieser für den hier betrachteten Einsatzkontext von besonderer Bedeutung ist. Schließlich wird erörtert, wie dies mit dem Vorgehen nach dem PAOSE Ansatz vereinbar ist.

3.5.1 Konzepte

Um zu erörtern, auf welche Weise die Ideen des Werkzeug und Material Ansatzes bei der Entwicklung von POTATO verwendet werden können, werden zunächst die Konzepte dieses Ansatzes vorgestellt. Nicht alle diese Konzepte werden auf die gleiche Weise verwendet, wie dies im WAM-Ansatz vorgesehen ist, andere werden komplett ausgelassen.

3.5.1.1 Leitbild

Ein Leitbild oder Leitmotiv im Softwareentwicklungsprozess ist ein Thema oder eine andere durchgängige Idee, die klar definiert und benannt wird. Es gibt dem gesamten Entwicklungsprozess und allen daran beteiligten Parteien eine gemeinsame Orientierung. Ein Leitbild besitzt sowohl eine analytische als auch eine konstruktive Funktion [ZÜLLIGHOVEN, 2004, S. 59ff].

3.5.1.2 Entwurfsmetapher

Im Entwurfsprozess ergibt sich an vielen Stellen die Fragestellung, auf welche Art und Weise eine Tätigkeit, ein Gegenstand oder ähnliches in der Software repräsentiert werden soll. Entwurfsmetaphern nehmen Gegenstände aus dem Anwendungskontext und setzen sie in den Kontext der Software, um bekannte Konzepte auf die neuen Softwareartefakte zu übertragen.

In einem Büro beispielsweise wird ein Aktenordner verwendet, um Dokumente zu verwalten. Die Metapher des Aktenordners kann in die Software übertragen werden als ein Objekt, dass bestimmte andere Objekte enthalten kann und dass sich analog zu einem Aktenordner verwenden lässt (Dokumente können darin abgelegt oder entnommen werden etc.). Werkzeuge und Materialien sind spezielle Entwurfsmetaphern, die zentral für den WAM-Ansatz sind. Sie werden im Folgenden genauer beschrieben.

Die Arbeitsumgebung besteht aus verschiedenen Arbeitsplätzen, die jeweils einem Benutzer zugeordnet sind. Auf diesen Arbeitsplätzen befinden sich verschiedene Arten von Gegenständen, nämlich Werkzeuge, Materialien und Automaten. Werkzeuge und Automaten können andere Gegenstände bearbeiten.

3.5.1.3 Materialien

Materialien sind die Objekte, auf denen eine Arbeit durchgeführt wird. Sie werden durch Werkzeuge bearbeitet und stellen bestimmte Konzepte des Anwendungskontextes dar.

Als Softwarematerialien stellen Materialien Dinge dar, die im Arbeitsverlauf hergestellt, bearbeitet, umgeformt und letztlich in ein Endprodukt überführt werden. Je nach aktuellem Kontext kann ein Gegenstand Material oder Werkzeug sein (Mit einem Bleistift schreiben vs. einen Bleistift anspitzen) [Züllighoven, 2004, S. 68f].

Verschiedene Materialien unterscheiden sich durch die Daten und Ressourcen, auf die sie Zugriff erlauben, sowie durch die Operationen, die auf ihnen ausgeführt werden können.

3.5.1.4 Werkzeuge

In der normalen Anschauung ist ein Werkzeug ein Gegenstand, mit dem Materialien untersucht und verändert werden können. Ein Werkzeug kann normalerweise für verschiedene Tätigkeiten und Materialien verwendet werden, und für eine Aufgabe können zum Teil verschiedene Werkzeuge verwendet werden. Zur sinnvollen Verwendung eines Werkzeuges ist in der Regel eine gewisse Vertrautheit mit den entsprechenden Tätigkeiten erforderlich. Werkzeuge sind im WAM-Ansatz das vornehmliche Mittel zur Benutzerinteraktion. Diese Interaktion besteht in der Regel in der Bearbeitung von Materialien, daher besitzen Werkzeuge Kenntnis über die verwendeten Materialien, umgekehrt kennen aber Materialien die Werkzeuge nicht, von denen sie verwendet werden [Becker-Pechau, 2009a].

Ein Softwarewerkzeug wird entsprechend dazu verwendet, Softwarematerialien zu untersuchen und zu bearbeiten. In der Regel unterscheiden sich Softwarewerkzeuge in ihrer Gestaltung und Handhabung von physikalischen Werkzeugen (zum Beispiel ist es in der Regel nicht sinnvoll, einen Stift 1:1 als Softwarewerkzeug umzusetzen, da die Funktion einer Texteingabe über eine Tastatur meist effizienter zu bewerkstelligen ist). Auch kann es erforderlich sein, verschiedene (physikalische) Werkzeuge zu einem Softwarewerkzeug zusammenzufassen, verschiedene Softwarewerkzeuge für verschiedene Anwendungsmöglichkeiten eines physikalischen Werkzeuges zu entwerfen, oder Werkzeuge zu implementieren, für die es kein physikalisches Gegenstück gibt [Züllighoven, 2004, S. 67f].

Abb. 3.12 verdeutlicht das Zusammenspiel von Werkzeugen und Materialien anhand eines einfachen Netzmodells: Ein Material besitzt einen internen Zustand (dargestellt durch die Stellen im Modell), der über verschiedene Operationen betrachtet und verändert werden kann (Transitionen mit Kanalanschriften). Ein Werkzeug kann ein Material, auf das es Zugriff hat, bearbeiten, wenn es die entsprechenden Operationen beherrscht.

3.5.1.5 Behälter

Behälter dienen als Sammlung gleichartiger Materialien und ermöglichen den Zugriff auf diese Materialien. Beispiele wären etwa eine Ablage oder ein Ordner,

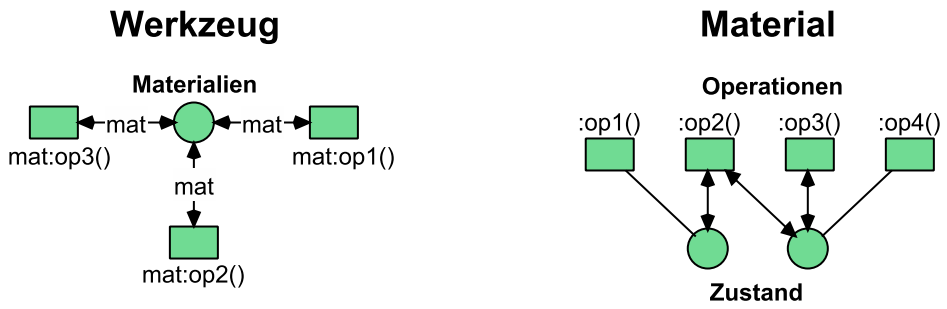

Abbildung 3.12: Einfaches Netzmodell für Werkzeuge (links) und Materialien (rechts)

in dem Dokumente abgelegt werden.

3.5.1.6 Services

Ein Service stellt eine fachliche Dienstleistung dar, die innerhalb eines Systems angeboten und verwendet werden kann. Services können dabei auf Materialien operieren und ihrerseits von Werkzeugen verwendet werden [Becker-Pechau, 2009b].

3.5.1.7 Aspekt

Werkzeuge können verschiedene Materialien bearbeiten und Materialien können durch verschiedene Werkzeuge bearbeitet werden. Die Übereinstimmung der Benutzungsmuster wird unter dem Begriff des Aspektes gefasst. Ein Aspekt beschreibt eine bestimmte Charakteristik eines Materials, z.B. dass es sortiert werden kann oder dass es auf eine bestimmte Art manipuliert werden kann, ein Aspekt ist also eine Beschreibung für ein Interface, dass ein Material implementiert. Entsprechend kann ein Werkzeug bestimmte Aspekte bearbeiten. Wenn es möglich ist, einen Aspekt zu finden, der zwischen Werkzeug und Material übereinstimmt, kann das Werkzeug mit dem Material verwendet werden.

In Abb. 3.12 wären die Aspekte die übereinstimmenden Kanalanschriften. Über diese Schnittstelle wird das Zusammenspiel zwischen kompatiblen Werkzeugen und Materialien ermöglicht.

3.5.1.8 Automaten

Der WAM-Ansatz wird teilweise auch durch die drei Bestandteile Werkzeug, Automat und Material gekennzeichnet [Gryczan, 1996, S. 119]. Automaten sind Programme, die selbstständig einen vordefinierten Prozess nach bestimmten Eingangsparametern durchführen, ohne dass weitere Benutzeraktivität erforderlich wäre.

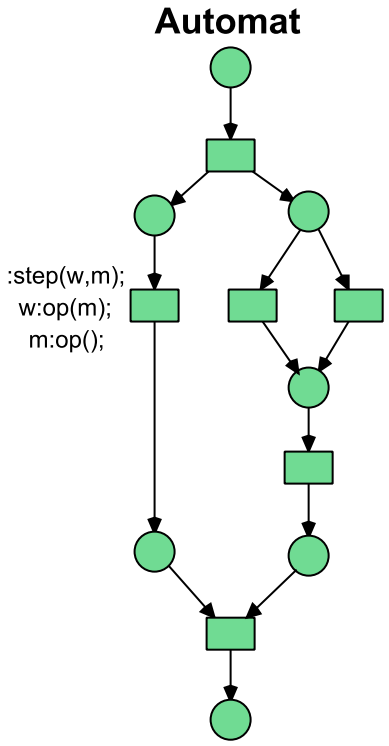

Abbildung 3.13: Automat als Workflow zur Koordinierung von Werkzeugen und Materialien

Dabei kann unterschieden werden in kleine Automaten, die Routineaufgaben erledigen, wie etwa die Berechnung eines Zinssatzes, ein Compiler, die automatische Generierung von Javadoc-Dokumentation etc. und große Automaten, die analog zu einer Produktionsstraße in einer Fabrik die Arbeitsabläufe auf einer höheren Ebene reguliert.

Im Netzmodell für einen Automaten in Abb. 3.13 ist ein Prozessschritt dargestellt, der sich mit den im System vorhandenen Werkzeugen und Materialien synchronisiert, um die passenden Gegenstände für die Ausführung dieses Schrittes auszuwählen. Diese werden dann miteinander geschaltet, indem das Material an das Werkzeug zur Bearbeitung übergeben wird.[8] Zusätzlich kann auch noch eine Synchronisierung mit einem geeigneten Benutzer im System vorgenommen werden.

[8]In diesem und dem folgenden Modell wird davon ausgegangen, dass das Werkzeug die zu bearbeitenden Materialien nicht direkt kennt, sondern diese von außen für die Bearbeitung hinzugegeben werden.

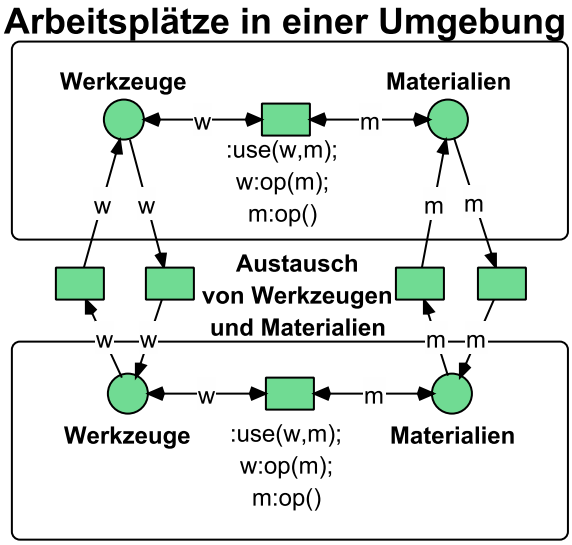

Abbildung 3.14: Netzmodell für Arbeitsplätze in einer Arbeitsumgebung

3.5.1.9 Arbeitsplatz und Arbeitsumgebung

Der Benutzer eines Systems verwendet eine Reihe von Werkzeugen, Automaten und Materialien, um seine Aufgaben zu erledigen. Der Arbeitsplatz beschreibt das spezifische Arbeitsumfeld eines Benutzers, also welche Gegenstände (Werkzeuge und Materialien) er in unmittelbarem Zugriff besitzt. Verbunden damit ist eine räumliche Komponente, die den Arbeitsplatz als eine Lokalität auffasst, an der sich die entsprechenden Gegenstände befinden.

Die Arbeitsumgebung ist weiter gefasst und beschreibt das weitere Umfeld, also Orte, die für den Benutzer erreichbar sind, Ressourcen, die er sich verschaffen kann, Mitarbeiter etc. Während der Arbeitsplatz für den Benutzer einen privaten Raum darstellt, ist die Arbeitsumgebung ein öffentlicher Raum. Gegenstände können zwischen verschiedenen Orten innerhalb der Arbeitsumgebung ausgetauscht werden.

Abb. 3.14 zeigt ein einfaches Modell für den Arbeitsplatz. Die dort vorhandenen Werkzeuge und Materialien können miteinander über die Auswahl im Arbeitsplatz in Verbindung gebracht werden. In einer verteilten Arbeitsumgebung können zudem Gegenstände zwischen den Arbeitsplätzen ausgestauscht werden.

3.5.1.10 Weitere Konzepte

Der WAM-Ansatz wird seit Jahren beständig weiterentwickelt und besteht aus einem kompletten Vorgehensmodell mit einer Reihe zusammenhängender Konzepte, Leitbilder und Metaphern. An dieser Stelle können nicht alle diese Konzepte ausführlich beschrieben werden. Vielmehr werden einzelne Teile

des Ansatzes als Inspiration verwendet, um Strukturen in agentenorientierten Systemen anzulegen.

3.5.2 Prozess- und Kooperationsunterstützung in WAM

Viele Anwendungen erfordern die Kooperation verschiedener Benutzer, um ein gemeinsames Ziel zu erreichen. Daher gibt es auch im WAM-Ansatz eine Reihe von Konzepten zur Unterstützung von Kooperationsprozessen. Analog zu anderen Aspekten der Softwareentwicklung mit WAM werden Gegenstände auch verwendet, um die Kooperationsmechanismen umzusetzen [Züllighoven, 2004, S. 341ff].

3.5.2.1 Implizite Kooperation

Im einfachsten Fall erfolgt die Kooperation implizit dadurch, dass mehrere Benutzer innerhalb des Systems auf das selbe Material zugreifen müssen. Voraussetzung dafür ist die Existenz verschiedener Arbeitsplätze innerhalb der Umgebung, die auf gemeinsame Materialien zugreifen können, z.B. über ein gemeinsam genutztes Speichermedium, das die Materialien verwaltet und dem Benutzer mitteilen kann, dass ein Material gerade zur Bearbeitung durch einen anderen Benutzer verwendet wird.

Die Abstimmung über die Art der Kooperation und die gemeinsame Verwendung der Materialien des Systems müssen außerhalb des Systems selbst erfolgen, also beispielsweise durch Konventionen zwischen den Benutzern oder durch individuelle Absprachen im Einzelfall.

3.5.2.2 Explizite Kooperation durch Austausch von Materialien

Kooperation kann expliziert werden, indem spezielle Kooperationsmedien verwendet werden, um gemeinsame Aufgaben zu unterstützen. Ein Mailsystem oder eine gemeinsame Ablage, mit denen Materialien zwischen Arbeitsplätzen ausgetauscht werden können, sind Beispiele für Kooperationsmedien, die eine explizite Kooperation unterstützen.

3.5.2.3 Kooperation durch Prozessmuster

Die nächste Stufe der Formalisierung besteht in der Identifizierung und Implementation von Prozessmustern [Wulf, 1995, Gryczan, 1996]. Ein Prozessmuster ist ein Material, das den normalen Ablauf einer Routineaufgabe abbildet. Es unterstützt die Koordination der notwendigen Schritte und der Verantwortlichkeiten für die Ausführung dieser Schritte. Dies muss aber keine feste Vorgabe darstellen, sondern kann zur Laufzeit angepasst werden, wenn es in der jeweiligen Situation erforderlich ist.

Prozessmuster können zum Beispiel durch Kooperationsmedien wie Laufzettel implementiert werden. Ein Laufzettel enthält die Aufgaben, Abhängigkeiten und Verantwortlichkeiten für einen bestimmten Fall und begleitet das Material, das den Fall darstellt (z.B. eine Vorgangsmappe) durch das System. Benutzer können aus dem Laufzettel ersehen, was sie bei einem bestimmten Fall zu tun haben, welche Dokumente benötigt werden, an wen der Fall nach der Bearbeitung weitergegeben werden soll etc.

3.5.2.4 Prozesssteuerung durch Automaten

Eine weitere Möglichkeit bietet die oben erwähnte Verwendung von (großen) Automaten. Die Entwurfsmetapher ist dann die einer Fertigungsstraße, die den Nutzern als Arbeitsstationen die einzelnen Arbeitsschritte vorgibt. In [Müller, 2001] beispielsweise werden Zustandsautomaten verwendet, um eine Vorgangssteuerung in Web-Anwendungen zu implementieren.

In der Regel wird bei WAM jedoch versucht, Prozesse eher durch Interaktionsmaterialien zu modellieren, anstatt sie in der Art eines Steuerungsautomaten fest vorzugeben. Nach Möglichkeit wird vermieden, zu stark in die Eigenverantwortlichkeit der Benutzer einzugreifen [Roock und Wolf, 1998].

3.5.3 Berührungspunkte und Abgrenzung zu Paose

Die Entwicklung von Software ist eine Tätigkeit, die sicherlich unter den Begriff der komplexen Expertentätigkeit fällt und somit für den WAM Ansatz geeignet ist. Die ersten Ideen zu diesem Ansatz stammen auch aus der Entwicklung einer Programmierumgebung für Prolog aus den Achtziger Jahren.

Im Paose-Ansatz gibt es eine ganze Reihe von Werkzeugen, die die Entwicklungsprozesse unterstützen. Renew ist nicht nur ein Werkzeug zur Modellierung und Ausführung von (Referenz-)Petrinetzen, durch verschiedene Plugins kann damit inzwischen der gesamte Softwareentwicklungsprozess nach Paose abgebildet werden.

Die Entwicklung nach Paose selbst orientiert sich aber in der Regel nicht an Werkzeug- oder Materialkonzepten. Die Erstellung von Werkzeugen und Unterstützungssystemen für einzelne Benutzer ist im Kontext der petrinetzbasierten agentenorientierten Softwareentwicklung bislang nicht betrachtet worden. Vielmehr sind für verschiedene Anwendungen immer neue Strukturierungskonzepte für die Agenten innerhalb der Anwendung entworfen worden.

In dieser Arbeit wird daher ein Vorgehen zur Strukturierung von Multiagentensystemen unter Verwendung von Entwurfsmetaphern des WAM-Ansatzes entwickelt. Dies soll helfen, die Entwicklung von Multiagentensoftware besser an Anwendungserfordernissen auszurichten und dem Entwicklungsprozess eine Struktur zu geben.

Der WAM Ansatz auf der anderen Seite ist hauptsächlich für Einzelplatzanwendungen entwickelt worden, nicht so sehr für verteilt zusammen arbeitende

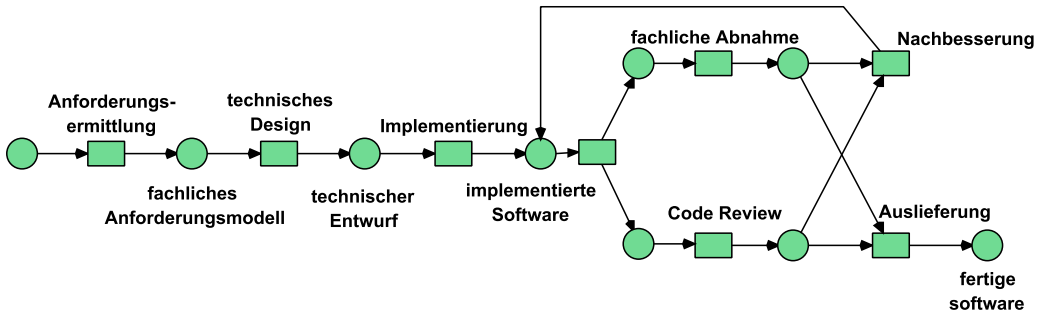

Abbildung 3.15: Beispiel für einen Softwareentwicklungsprozess

Anwender. Die Konzepte zur Unterstützung verteilter Arbeit sind daher noch nicht besonders stark ausgeprägt. Für die Modellierung der Prozesse innerhalb einer Anwendung werden daher Petrinetze und Workflow-Methodiken verwendet. In Abschnitt 6.1 wird näher ausgeführt, wie die verschiedenen Bestandteile des WAM-Ansatzes in PAOSE integriert werden.

3.6 Workflow Management Systeme

Informationssysteme zur Unterstützung der Geschäftsprozesse in einem Unternehmen hatten häufig das Problem, dass Annahmen über die Reihenfolge, in der die Aktivitäten ausgeführt werden, fest im Design der Anwendung verankert waren. Dies führte dazu, dass Änderungen des Vorgehens schwierig und oft fehleranfällig waren. Die explizite Modellierung und Flexibilisierung der Prozesse mit dem Ziel der einfachen Restrukturierung führte zu Workflow Management Systemen [OBERWEIS, 2005, S. 21].

3.6.1 Definition

Ein Geschäftsprozess oder Workflow umfasst alle Aktivitäten, die zur Bearbeitung eines bestimmten Falles erforderlich sind, sowie die Abhängigkeiten zwischen diesen Aktivitäten. In einem Softwareentwicklungsprozess muss beispielsweise zunächst ein fachliches Anforderungsmodell und ein technischer Entwurf erstellt werden, bevor mit der Implementation begonnen werden kann. Fachliche Abnahme und code Review können dann nebenläufig erfolgen. Abb. 3.15 zeigt einen solchen Entwicklungsprozess.[9] Zusätzlich zur Beschreibung der kausalen Abhängigkeiten kann die Workflow-Definition Informationen über die am Prozess beteiligten Rollen, benötigte Ressourcen und andere Parameter enthalten.

[9]Der hier dargestellte Workflow zeigt einen sehr klassischen Prozess nach dem Wasserfallmodell. Iterative Vorgehensweisen sind natürlich ebenfalls denkbar.

3.6.2 Modelle

Verschiedene Modelle und Formalismen können verwendet werden, um Prozesse zu definieren, von UML Aktivitätsdiagrammen [UML, 2011] über Ereignisgesteuerte Prozessketten (EPK) [Keller et al., 1992] bis hin zu Petrinetzen [Oberweis, 1996, Aalst, 1998, Moldt und Rölke, 2003, Russell et al., 2009]. In dieser Arbeit werden Petrinetze verwendet, da sie nicht nur eine intuitive graphische Repräsentation der Prozesse ermöglichen, sondern auch auf einem soliden mathematischen Fundament basieren und über entsprechende Werkzeuge direkt ausgeführt werden können [Desel, 2005]. Abb. 3.15 zeigt ein Beispiel für ein Workflow-Petrinetz (siehe auch Abschnitt 3.1.2).

3.6.3 Workflow Referenzmodell

Die weite Verbreitung von WfMS in Unternehmen hat dazu geführt, dass eine Reihe von Standards für die Beschreibung von Workflows und WfMS etabliert wurden. Diese Arbeit orientiert sich bei Entwurf und Implementation des WfMS am Referenzmodell der Workflow Management Coalition (WfMC) [WfMC, 2011a]. Die WfMC ist ein Zusammenschluss verschiedener Entwickler, Anwender, Berater und anderer Parteien, die ein Interesse an der Standardisierung von Workflow und Business Process Management Systemen haben. Sie hat ein Referenzmodell für Workflow Management Systeme (WfMS) herausgegeben [WfMC, 1995], an dem sich die Implementierung der agentenorientierten Prozessinfrastruktur Pia anlehnt, die eine Grundlage für das Potato-System darstellt.

Das Workflow Referenz Modell [WfMC, 1995, Hollingsworth et al., 2004] beschreibt die Bestandteile eines WfMS. Der Schwerpunkt liegt dabei auf der Interoperabilität von Komponenten verschiedener Hersteller. Neben einem einheitlichen Vokabular werden daher eine Reihe von Schnittstellen zwischen verschiedenen Bestandteilen eines WfMS beschrieben. Das Referenzmodell gibt dabei zunächst keine konkrete Ausgestaltung dieser Schnittstellen an, spätere Publikationen beschreiben dann konkrete Schnittstellen in C, XML etc.

3.6.3.1 Workflow-Begriffe

Dieser Abschnitt erläutert die Begrifflichkeiten, die im Referenzmodell für die verschiedenen Aspekte eines WfMS verwendet werden. Diese Begriffe werden auch im weiteren Verlauf der Arbeit verwendet, um Missverständnisse zu vermeiden [WfMC, 1999, Kreplin, 1998]. Auch wenn zu allen Begriffen deutsche Übersetzungen bei der WfMC angegeben werden, werden hier zum Teil auch die englischen Begriffe verwendet, da diese häufig gebräuchlicher sind. Abb. 3.16 zeigt den Zusammenhang zwischen den Begriffen, wie im Folgenden beschrieben.

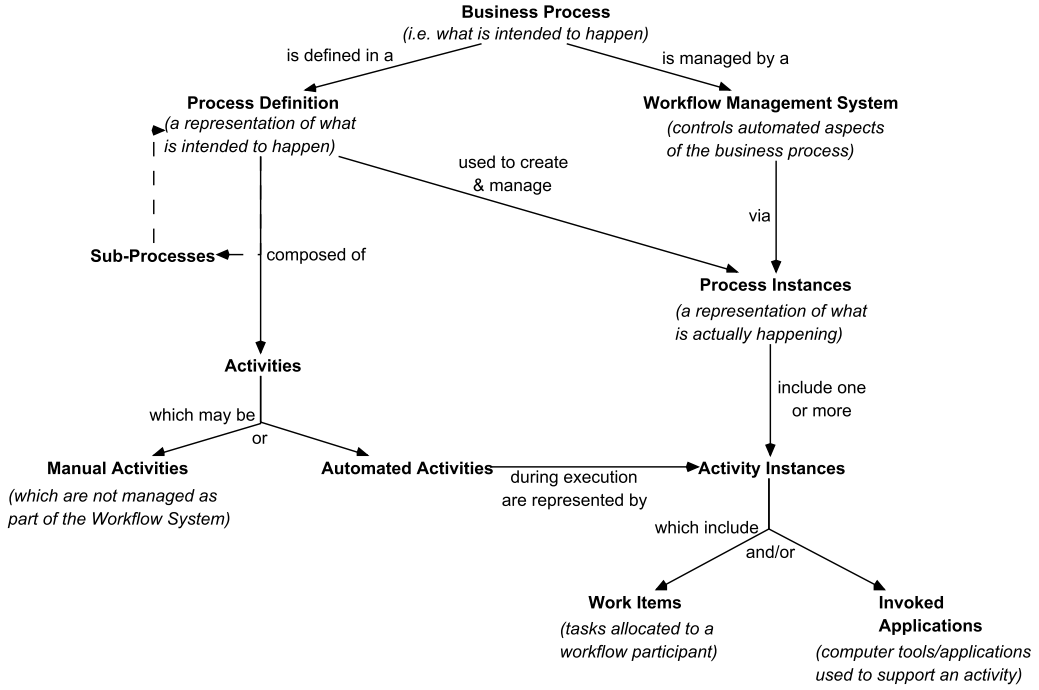

Abbildung 3.16: Zusammenhang der Konzepte im WfMC Referenzmodell (nach [WfMC, 1999, S. 7])

Business Process - Geschäftsprozess Ein Geschäftsprozess ist eine Reihe zusammengehöriger Aktivitäten, die gemeinsam ein Geschäftsziel innerhalb eines Unternehmens erreichen sollen. Üblicherweise ist ein Geschäftsprozess in eine Organisationsstruktur eingebettet, die funktionale Rollen und Beziehungen definiert.

Workflow - Vorgangssteuerung Ein Workflow ist eine Automatisierung für einen Geschäftsprozess oder Teile davon. Dabei werden Dokumente und Informationen nach festgelegten Regeln zwischen verschiedenen Beteiligten zur Bearbeitung ausgetauscht. Workflows können unterteilt werden in Produktionsworkflows, die fest definiert sind und in unveränderter Form häufig wiederholt ausgeführt werden und Ad-hoc-Workflows, die für eine Einzelsituation definiert und ausgeführt werden.

Workflow Management System - Vorgangssteuerungssystem Ein Workflow Management System (WfMS) ist ein Softwaresystem, das Workflowdefinitionen erzeugen, verwalten und ausführen kann. Ein WfMS besitzt eine oder mehrere Workflow Engines, die Prozessdefinitionen interpretieren können, steht in Interaktion mit den Workflowteilnehmern und kann Applikationen aufrufen, die im Workflow verwendet werden. Zusätzlich bietet ein

WfMS in der Regel Möglichkeiten, um die laufenden Prozesse zu überwachen und zu administrieren.

Prozess und Prozessinstanz Um einen Geschäftsprozess innerhalb eines WfMS ausführen zu können, muss er in Form einer Prozessdefinition formalisiert werden. Diese besteht aus einer Reihe von Aktivitäten, die miteinander in Beziehung stehen, Bedingungen für Beginn und Ende des Prozesses, und Informationen über die einzelnen Aktivitäten, wie Bearbeiter, zugeordnete Programme und Daten, Bedingungen etc. Prozessdefinitionen können in Unterprozesse strukturiert sein.

Die konkrete Ausprägung eines Prozesses für einen bestimmten Fall wird als Prozessinstanz oder Geschäftsvorfall bezeichnet. Die Instanz besitzt lokale Fall- und Steuerdaten und ist ein individuelles Objekt innerhalb des WfMS, mit einem eindeutigen Bezeichner, über den sie referenziert werden kann. Das WfMS ist zuständig für ihre Erzeugung, Verwaltung und Beendigung.

Ein Prozess (oder Prozessmuster) beschreibt allgemein, wie eine bestimmte Aufgabe ausgeführt wird, eine Prozessinstanz bezeichnet einen konkreten Fall, also beispielsweise einen Versicherungsfall, die Implementation eines bestimmten Softwaremoduls, einen Change Request etc. Der Zusammenhang entspricht in etwa dem zwischen einer Klasse und einem Objekt in der objektorientierten Programmierung.

Aktivität und Aktivitätsinstanz Eine Aktivität ist eine Arbeitseinheit, die einen logischen Schritt innerhalb eines Prozesses darstellt. Aktivitäten werden unterschieden in manuelle Aktivitäten, die von einem menschlichen Bearbeiter ausgeführt werden müssen, und automatisierte Workflow-Aktivitäten, die durch das WfMS verwaltet werden. Manuelle Aktivitäten können zu Modellierungszwecken in einer Prozessdefinition verwendet werden, sind aber nicht Teil des Workflows, da ihre Ausführung nicht vom WfMS überwacht wird.

Workflow-Aktivitäten werden durch das WfMS verwaltet und können verschiedene Aktionen auslösen. Eine Aktivität kann eine Anwendung aufrufen, die eine Aufgabe ohne Benutzereingriff ausführt. Es können ein oder mehrere Workitems erzeugt und Benutzern des WfMS zugewiesen werden. Für die Bearbeitung dieser Aufgabe stellt das WfMS dem Benutzer dann Werkzeuge und Anwendungen zur Verfügung. Schließlich kann eine Aufgabe auch komplett ohne Unterstützung des Systems ausgeführt werden. Der Benutzer meldet dann nur die Erledigung der Aufgabe an das System zurück, so dass der Prozess weitergeführt werden kann.

Die konkrete Ausprägung einer Aktivität in einer Prozessinstanz wird als Aktivitätsinstanz bezeichnet. Das WfMS erzeugt Aktivitätsinstanzen während der Ausführung einer Prozessinstanz nach den Vorgaben der Prozessdefinition. Eine Aktivitätsinstanz ist immer genau einer Prozessinstanz zugeordnet. In Pia wird der Ausdruck Aktivität für eine Aufgabe verwendet, die von ei-

nem konkreten Teilnehmer zur Bearbeitung angenommen und ihm zugewiesen worden ist.

Participant - Teilnehmer Aktivitätsinstanzen werden durch Workflow Teilnehmer bearbeitet. Die Aufgaben, die ein Teilnehmer bearbeiten kann, werden in Form von Workitems in einer Worklist verwaltet. In der Regel ist ein Workflow Teilnehmer ein menschlicher Benutzer des Systems und hier werden Workflow Teilnehmer und Benutzer synonym verwendet, ein Teilnehmer kann aber auch ein automatisierter Prozess, wie z.B. ein Agent sein.

Innerhalb einer Workflowdefinition kann der Teilnehmer für die Bearbeitung einer Aktivität auf verschiedene Weise spezifiziert werden. Es kann eine konkrete Person oder eine Computerressource genannt werden, meist aber wird eine organisationale Rolle angegeben, die im konkreten Fall über ein Organisationsmodell auf konkrete Benutzer abgebildet wird.

Workitem - Elementaraufgabe Ein Workitem bezeichnet eine Arbeitseinheit, die ein Workflow Teilnehmer im Rahmen einer Aktivitätsinstanz ausführt, ein Workitem ist damit einem konkreten Teilnehmer zugeordnet. Eine Aktivitätsinstanz kann ein oder mehrere Workitems erzeugen. Die Workitems, die einem Benutzer zugewiesen sind, bilden seine Worklist.

In Pia kann ein Workitem gleichzeitig mehreren Benutzern zur Bearbeitung angeboten werden. Die konkrete Zuordnung zu einem Benutzer erfolgt erst, wenn dieser das Workitem zur Bearbeitung angefordert hat. Dann wird daraus eine dem Benutzer zugeordnete Aktivität.

Worklist - Aufgabenliste Die Aufgabenliste oder Worklist stellt eine Schnittstelle zwischen Workflow Teilnehmern und Workflow Engine dar. Sie verwaltet die Workitems eines Teilnehmers und meldet zu bearbeitende Aufgaben an den Benutzer, fertig bearbeitete Aufgaben zurück an die Workflow Engine. Über einen Worklist Handler erhalten die Benutzer Zugriff auf ihre jeweilige Worklist. Die konkrete Implementierung des Worklist Handlers kann über Push- oder Pull-Mechanismen funktionieren und ist abhängig von der konkreten Umgebung.

3.6.3.2 WfMS Bestandteile und Schnittstellen

Die WfMC Spezifikation definiert eine Reihe von Bestandteilen und Schnittstellen, die von Workflowprodukten implementiert werden können. Es müssen nicht zwangsläufig immer alle Teile vorhanden sein, um der Spezifikation zu genügen. Ein WfMS verfügt über eine Reihe von Schnittstellen zu benachbarten Systemen, die im Referenzmodell beschrieben werden. Anhand dieser Schnittstellen lassen sich auch die Funktionalitäten des Systems erläutern. Der Grund für die Spezifikation dieser Schnittstellen ist die große Heterogenität im

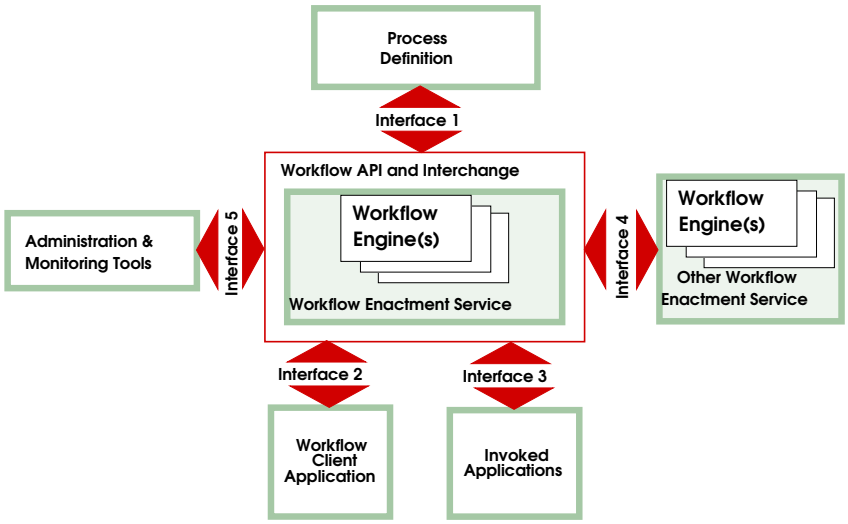

Abbildung 3.17: WfMC Referenzarchitektur: Bestandteile und Schnittstellen (nach [WfMC, 1995, S. 20])

Bereich der Workflowmanagementsysteme und benachbarter Systeme; durch klare Schnittstellen soll eine bessere Interoperabilität von Produkten verschiedener Hersteller erreicht werden.

Abb. 3.17 zeigt die Bestandteile und Schnittstellen eines WfMS. Im Kern hat das WfMS einen Workflow Enactment Service, der verschiedene Workflow Engines verwaltet, in denen Workflowinstanzen ausgeführt werden. Externe Applikation können über die verschiedenen Schnittstellen angebunden werden.

Workflow Engine Eine Workflow Engine ist die Ausführungsumgebung für Workflowinstanzen. Sie ist verantwortlich für die Interpretation der Prozessdefinition und die Verwaltung der Prozess- und Aktivitätsinstanzen, der Steuerungsdaten etc. Ein WfMS kann mehrere Workflow Engines enthalten, die verschiedene Prozessinstanzen bearbeiten, oder eine einzelne Engine für alle Prozesse.

Workflow Enactment Service Der Workflow Enactment Service besteht aus mehreren Workflow Engines und ermöglicht die Erzeugung, Verwaltung und Ausführung von Workflow Instanzen. Sie stellt diese Dienste externen Anwendungen zur Verfügung und dient damit quasi als Ansprechpartner für das WfMS. Der Zugriff erfolgt im Allgemeinen über das Workflow Application Programming Interface (WAPI). Dieses Interface besteht aus den folgenden fünf Schnittstellen:

Prozessdefinition Die erste Schnittstelle beschreibt die Definition von Prozessen. Hierfür gibt es mittlerweile eine Reihe verschiedener Formate (z.B.

XPDL [WfMC, 2011b], BPEL [OASIS, 2011], BPMN [OMG, 2011]) und eine Vielzahl an Programmen, die diese Formate verarbeiten können. Eine Prozessdefinition kann unter anderem folgende Aspekte umfassen:

- Beschreibung des Workflows (Name, Start- und Endbedingungen, Workflowsteuerdaten)
- Aktivitäten (Aktivitätstypen, Prä- und Postkonditionen, Terminierungsbedingungen)
- Transitionsbedingungen (Bedingungen für die Ausführung verschiedener Prozesszweige)
- Prozessrelevante Daten (Falldaten, die im Prozess mitgeführt werden)
- Rollen
- Verwendete Applikationen (Typ, Parameter, Pfade)

Workflow Client Application Benutzer des WfMS müssen die Möglichkeit erhalten, mit dem System zu kommunizieren. Hierzu dient die Workflow Client Application Schnittstelle. Sie bietet unter anderem folgende Funktionen an:

- Abfragen der Worklist eines Benutzers
- Annehmen und Erledigen von Aufgaben
- Abfrage von Prozessinformationen
- Starten neuer Prozessinstanzen
- Statusänderungen von Prozess- und Aktivitätsinstanzen
- Abfrage und Änderung der Attribute von Prozess- oder Aktivitätsinstanzen

Invoked Applications Im Laufe eines Prozesses werden verschiedene Anwendungen aufgerufen, um unterschiedliche Aufgaben im Prozess zu bearbeiten. Hierzu können verschiedene Schnittstellen verwendet werden und nicht jedes WfMS kann mit jedem Programm kommunizieren. Im einfachsten Fall beinhaltet dies einfach den Namen eines Programms, dass für die Aktivität aufgerufen werden soll. Bei stärkerer Integration zwischen WfMS und aufgerufenem Programm können Parameter aus dem Prozess an die Applikation gegeben werden und nach Abschluss aus der Applikation in den Prozess übernommen werden.

Remote Workflow Engine Diese Schnittstelle ermöglicht die Kommunikation zwischen verschiedenen Workflow Management Systemen bzw. verschiedenen Workflow Engines. Es sind verschiedene Arten von Interaktion möglich, die alle durch eine gemeinsame Schnittstelle unterstützt werden sollen:

- Sequentielle Ausführung (Workflowende auf WfMS A löst neuen Prozess auf WfMS B aus)
- Subprozess (Aktivität auf WfMS A wird durch Prozess auf WfMS B ausgeführt)
- Synchronisation (Aktivitäten aus verschiedenen Prozessen werden gemeinsam ausgeführt)
- Engine-übergreifende Ausführung (Aktivitäten eines Prozesses werden auf mehrere Workflow Engines verteilt)

Administration und Monitoring Diese Schnittstelle dient dem Zweck, Informationen über die verschiedenen Prozesse zu erhalten, die auf einem WfMS ausgeführt werden, sowie bei Bedarf in diese Prozesse einzugreifen. Durch eine standardisierte Schnittstelle soll es ermöglicht werden, dass verschiedene WfMS, die unabhängig voneinander laufen, in einem gemeinsamen Werkzeug administriert werden.

Bei der Implementierung der agentenorientierten Prozessinfrastruktur Pia (Kapitel 7) wird eine agentenorientierte Zerlegung der WfMS-Funktionalität vorgenommen. Die verschiedenen Komponenten werden als einzelne Agenten implementiert, die in Kooperation die Gesamtfunktionalität des WfMS erbringen und nach außen als Dienst anbieten.

3.6.4 Verifikation von Workflows

Nachdem ein Workflowprozess beschrieben worden ist, ist es erforderlich zu prüfen, ob der Prozess zum einen das tut, wofür er entwickelt worden ist, zum anderen, dass er fehlerfrei ist. In [Aalst, 1997] wird ein formales Petrinetzmodell für Workflows aufgestellt und das Kriterium der Soundness definiert, mit dem nachgewiesen werden kann, ob ein Workflow jederzeit korrekt terminiert. Dies wurde bereits in Abschnitt 3.1.2.1 behandelt.

In [Aalst et al., 1994] werden höhere Petrinetze, in [Aalst et al., 1999] Referenznetze verwendet, um ein WfMS zu modellieren. Im Gegensatz zu den Workflownetzen werden hier auch Ressourcen explizit modelliert und das Gesamtsystem dargestellt. An einem solchen Modell können mit netzbasierten Methoden weitergehende Untersuchungen vorgenommen werden.

Allerdings kann mit formalen Methoden nur überprüft werden, ob ein Workflow grundsätzlich korrekt ist, also zum Beispiel nicht verklemmen kann. Schwieriger ist die Frage, ob er fachlich das tut, wofür er entworfen worden ist. Diese Überprüfung von Workflows ist nicht Thema in dieser Arbeit.

3.6.5 Workflows für die Softwareentwicklung

Im Bereich der Softwareentwicklung spielen Workflows und WfMS eine große Rolle. Workflow Management Systeme dienen als Bausteine für komplexere Applikationen, die eine Prozesssteuerung erfordern. Bei der Entwicklung müssen dann die relevanten Workflows definiert werden und die Steuerung mit dem Rest der Applikation integriert werden.

Zum anderen werden WfMS oder Case Handling Tools auch für den Prozess der Softwareentwicklung verwendet. Software Engineering erfordert die Zusammenarbeit verschiedener Personen und oft ist eine Koordinationsunterstützung hilfreich, etwa im Bereich des Change Managements und der Fehlerbehebung. Grundsätzlich kann aber für den gesamten Softwareentwicklungsprozess eine Workflowunterstützung verwendet werden.

In Jazz [Jazz, 2011a] beispielsweise ist es möglich, Workflows für die verschiedenen Projektphasen zu definieren. Dadurch soll sowohl die individuelle wie auch die Teamarbeit besser koordiniert und die Projekttransparenz erhöht werden.

3.7 Terminologie

In dieser Arbeit werden eine ganze Reihe verschiedener Bereiche berührt. Daher werden an dieser Stelle die Begriffe aufgeführt, die verwendet werden und klargestellt, in welcher genauen Bedeutung sie in dieser Arbeit verwendet werden. Teilweise können Begriffe, wie etwa Protokoll oder Prozess auf verschiedene Weise verwendet werden, dieser Abschnitt soll daher für Klarheit sorgen.

3.7.1 Protokoll

Unter einem Protokoll versteht man eine Vereinbarung über die Art und Weise, wie eine Kommunikation oder Interaktion zwischen verschiedenen Beteiligten abläuft. Wenn der Begriff des Protokolls in diesem Zusammenhang gemeint ist, wird hier der Ausdruck Interaktionsprotokoll verwendet.

Im Mulan-Multiagentensystem bezeichnet ein Protokoll (oder Agentenprotokoll) die internen und externen Aktionen eines Agenten während einer solchen Interaktion. Die Spezifikation einer Interaktion mittels eines Interaktionsprotokolls resultiert also in mehreren (Agenten-)Protokollen in den ausführenden Agenten.

3.7.2 Workflow-Begriffe

Im Zusammenhang mit Workflow Management Systemen werden eine Reihe von Begriffen verwendet. Diese werden im Abschnitt 3.6.3.1 ausführlich diskutiert. Teilweise werden diese Begriffe jedoch in Pia und daher auch in dieser Arbeit abweichend verwendet:

- Der Zusatz Instanz wird häufig weggelassen, wenn es aus dem Zusammenhang ersichtlich ist, dass eine konkrete Instanz und nicht die abstrakte Beschreibung gemeint ist (etwa Aktivitätsinstanz)

- Der Begriff Workflow wird im Rahmen des WfMS synonym mit Prozess bzw. Prozessbeschreibung verwendet

- Workflow-Teilnehmer (Participants) werden auch als Executor bezeichnet, da sie Aufgaben innerhalb des WfMS ausführen

- Aktivitäten innerhalb eines Workflows werden in Pia als Tasks oder Aufgaben bezeichnet

- Instanzen einer Aufgabe werden immer als elementar angesehen und als Workitems bezeichnet. Sie sind einer Workflowinstanz zugeordnet, können aber mehreren Workflow-Teilnehmern angeboten werden

- Eine Aktivität in Pia ist ein Workitem, das einem konkreten Teilnehmer zur Bearbeitung zugeordnet ist und sich in Bearbeitung befindet

3.7.3 Agent

Der Begriff des Agenten und des Multiagentensystems wird im Rahmen dieser Arbeit im Sinne von autonom agierenden Einheiten innerhalb eines größeren Systems verstanden, die miteinander interagieren können. Agenten befinden sich auf Agentenplattformen und interagieren miteinander durch den Austausch von Nachrichten.

Agenten sind nicht im Sinne von Expertensystemen zu verstehen, sondern als interagierende Entitäten. Auch sind keine physikalischen Agenten, wie etwa autonome Roboter damit gemeint. Ausführlicher wird dies in Abschnitt 3.2 diskutiert.

3.7.4 Werkzeug und Material

Ein wichtiger Aspekt des Potato-Systems ist die Bereitstellung einer Werkzeugumgebung für kollaborative Anwendungen. In dieser Umgebung können Werkzeuge und Materialien erstellt und verwendet werden. Diese Metaphern sind inspiriert vom Werkzeug und Material Ansatz [Züllighoven, 2004], ohne dass der komplette Ansatz hier verwendet wird. Werkzeuge wie Materialien sind als Agenten repräsentiert.

Der Hauptaspekt, der hier verwendet wird, ist die Unterteilung in Arbeitsobjekte (Materialien), die erstellt und bearbeitet werden können, und Arbeitsmittel (Werkzeuge), die diese Bearbeitung erlauben und unterstützen. Als Arbeitsplatz wird der persönliche Einflussbereich eines Benutzers verstanden, der

die ihm zugeordneten Werkzeuge und Materialien umfasst. Die Arbeitsumgebung umfasst auch andere Arbeitsplätze und Orte in der Umgebung des Benutzers, die seine Arbeit beeinflussen können. Dies wird erörtert in Abschnitt 3.5.

3.8 Zusammenfassung

Ein komplexes System wie POTATO kann nicht ohne Rückgriff auf verschiedene Vorarbeiten und Basistechnologien realisiert werden. Dieses Kapitel diente dazu, diese Hintergründe zu beleuchten, um das Fundament für POTATO zu legen.

Die Arbeit basiert zum einen auf verschiedenen technischen und formalen Grundlagen, namentlich Petrinetzen, Workflow Management Systemen, Agenten und Multiagentensystemen. Als agentenorientierter Entwicklungsansatz dient der PAOSE-Ansatz. Zum anderen sind aber auch Anwendungsaspekte für die Konzeption und den Einsatz der verteilten Softwareentwicklungsumgebung von Interesse, nämlich der Werkzeug und Material-Ansatz (WAM) der Softwareentwicklung und das Feld der Computer Supported Cooperative Work.

Die formale Grundlage und zugleich die Modellierungs- und Implementationssprache für POTATO bilden Petrinetze. Daher stellt die Arbeit zunächst den Basisformalismus für Petrinetze sowie zwei spezielle, in der Arbeit verwendete Netzklassen vor. Zur Modellierung von Geschäftsprozessen werden Workflownetze verwendet, die eine formale Überprüfung von Prozessen auf Korrektheit erlauben und später für die agentenorientierte Prozessinfrastruktur verwendet werden. Referenznetze stellen einen sehr ausdrucksstarken Formalismus dar, mit dem komplexe Architekturen anschaulich in Form von Netzen modelliert werden können. Daher werden sie für die Modellierung der Architektur von POTATO, sowie für die Implementierung der Multiagentenanwendung verwendet.

Agentenorientierung ist ein Paradigma der Softwareentwicklung, das sich insbesondere für verteilte Systeme eignet, die sich durch ein hohes Maß an Flexibilität, Heterogenität und Autonomie der beteiligten Parteien auszeichnen. In dieser Arbeit wird das petrinetzbasierte Multiagentensystem MULAN verwendet. Für die agentenorientierte Softwareentwicklung sind eine Reihe von Methodiken entwickelt worden, in dieser Arbeit wird der, speziell auf Petrinetze und Prozesse zugeschnittene PAOSE-Ansatz verwendet.

Die Unterstützung verteilter Zusammenarbeit in der Softwareentwicklung lässt sich dem Bereich der computergestützten Zusammenarbeit (Computer Supported Cooperative Work - CSCW) zuordnen. In diesem Forschungsgebiet finden sich die Hauptaspekte Kommunikation, Koordination und Kooperation als Bereiche für die Entwicklung von Unterstützungssystemen, auf die daher besonderes Augenmerk gelenkt wird.

Der Werkzeug und Material Ansatz (WAM) stellt ein anwendungsnahes

Vorgehen zur Softwareentwicklung dar, das Anwendungen anhand der darin verwendeten Werkzeuge und Materialien modelliert. In POTATO werden Ideen aus diesem Ansatz aufgegriffen und auf die Entwicklung von Multiagentensystemen übertragen und erweitert, so dass Agenten als intelligente Helfer die Benutzer in ihren Aufgaben unterstützen können. Diese Integration von petrinetzbasierter Multiagentenentwicklung mit den Konzepten aus dem WAM Ansatz resultiert im HERA-System, das eine der beiden Säulen von POTATO bildet.

Die zweite Säule des POTATO-Systems stellt die Prozessunterstützung dar. Die Grundlage hierfür stellen die Konzepte, Modelle und Bestandteile von Workflow Management Systemen dar. Die agentenorientierte Prozessinfrastruktur PIA, die im Kontext dieser Arbeit entwickelt wurde, setzt diese Konzepte für Multiagentenanwendungen um.

Da eine Vielzahl verschiedener Konzepte im Rahmen der Arbeit angesprochen werden, wurde anschließend die Terminologie definiert, die im weiteren Verlauf verwendet worden ist. Auf diese Weise ist klar definiert, was jeweils mit einem Begriff gemeint ist, auch wenn er in verschiedenen Kontexten unterschiedlich verwendet werden kann.

Im nächsten Kapitel wird auf Verteilte Softwareentwicklungsprozesse eingegangen, die das Anwendungsbeispiel der Arbeit bilden. Anforderungen aus dem Software Engineering werden beleuchtet ebenso wie die Besonderheiten, die sich aus der Verteilung der Prozesse ergeben.

Kapitel 4

Verteilte Softwareentwicklungsprozesse

In diesem Kapitel wird zunächst der Begriff der Softwaretechnik und die darin vorgefundenen Prozesse diskutiert. Zusätzlich wird auf die Implikationen eingegangen, die sich durch verteilte Systeme und verteilte Entwicklungsprozesse ergeben. Anschließend werden Entwicklungsumgebungen und -werkzeuge, speziell für verteilte Entwicklung diskutiert. Schließlich werden Kriterien aufgestellt, anhand derer eine verteilte Entwicklungsumgebung evaluiert werden kann. Hieraus ergeben sich Anforderungen und Entwurfsvoraussetzungen, die in den späteren Kapiteln aufgegriffen werden.

4.1 Software Engineering

Das Gebiet der Softwaretechnik (engl. Software engineering) beschäftigt sich mit der Erstellung und dem Betrieb von Softwaresystemen, die aufgrund ihrer Größe und Komplexität nicht mehr von einzelnen Entwicklern, sondern nur durch größere Entwicklerteams geleistet werden können. Im Laufe des Lebenszyklus dieser Systeme sind in der Regel verschiedene Erweiterungen und neue Versionen erforderlich [Ghezzi et al., 2002, S. 6]. Daher ist es Ziel der Softwaretechnik, „Prinzipien, Methoden und Werkzeuge für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen“ [Balzert, 2001, S. 36] zu erstellen und anzuwenden.

Da es sich bei der verteilten Softwareentwicklung um einen Teilbereich der Softwaretechnik handelt, gelten die dargestellten Prinzipien hierfür entsprechend, wenngleich zusätzlich eigene Methoden und Werkzeuge erforderlich sind, um den speziellen Herausforderungen zu begegnen, die sich aus der Verteilung ergeben.

4.1.1 Der Softwareentwicklungsprozess

Für die Erstellung und den Betrieb einer Software sind eine Reihe von Aktivitäten erforderlich, die zusammen den Softwareentwicklungsprozess bilden. Zur Modellierung und Durchführung dieses Prozesses gibt es eine Reihe von Vorgehensmodellen. Im einfachen Wasserfallmodell wird davon ausgegangen, dass diese Aktivitäten eine nach der anderen durchgeführt werden, ohne Rückgriff auf vorhergehende Prozessschritte. Andere Modelle, wie beispielsweise evolutionäre Entwicklungsprozesse, verwenden einen iterativen Ansatz, bei dem auf die Ergebnisse vorhergehender Entwicklungszyklen Rückgriff genommen wird. Meist findet man aber ähnliche Grundaktivitäten [Sommerville, 2007, S. 94f].

Im Verlauf dieser Arbeit werden diese Aktivitäten sequentiell behandelt und die Ergebnisse im Sinne eines Wasserfallmodells vorgestellt, bei der tatsächlichen Entwicklung des Potato-Systems wurden aber an vielen Stellen wiederholte Iterationen vorgenommen, Zwischenergebnisse korrigiert und Anforderungen an neue Erkenntnisse angepasst, wie dies auch im Paose-Entwicklungsprozess vorgesehen ist.

4.1.1.1 Anforderungsermittlung

In dieser Phase wird ermittelt, welche Funktionen die Software ausführen soll, welche Nebenbedingungen einzuhalten sind und in welchem Kontext sich die Software befindet. Hieraus wird dann eine Systemspezifikation abgeleitet, die als Basis für die folgenden Entwurfs- und Implementationsaktivitäten dient.

Die Anforderungen an das System werden zunächst hauptsächlich aus einer fachlichen Sicht aufgeführt. Es wird festgehalten, welchen Zweck es aus Anwendersicht erfüllen soll. Die konkrete technische Umsetzung spielt zu diesem Zeitpunkt noch keine oder nur eine untergeordnete Rolle.

4.1.1.2 Entwurf

Beim Systementwurf werden die Anforderungen an das System aufgeteilt in Funktionseinheiten, die die jeweiligen Funktionalitäten erbringen sollen. Eine grundlegende Architektur wird festgelegt, die die Struktur und die Beziehungen der einzelnen Softwarebestandteile definiert.

Im Gegensatz zur fachlichen Anforderungsanalyse ist dieser Entwurf technischer Natur und berücksichtigt bereits technische Gegebenheiten und Rahmenbedingungen. Aus diesem Grund kann es vorkommen, dass ursprüngliche Anforderungen in Absprache mit dem Kunden der Software abgeändert werden, wenn dies aus technischer Sicht vorteilhaft erscheint.

4.1.1.3 Implementation

Die im Entwurfsdokument festgehaltene Spezifikation wird nun in konkrete Software umgesetzt. Übereinstimmung mit den Spezifikationen lässt sich über Testsuiten prüfen, oft treten aber auch während der Implementationsphase Unklarheiten oder Inkonsistenzen in der Spezifikation zu Tage, die eine Überprüfung des bisherigen Entwurfs erforderlich machen.

4.1.1.4 Integrations- und Abnahmetest

Wenn alle Teile fertig implementiert sind, muss das Zusammenspiel der verschiedenen Komponenten getestet werden. Im Abnahmetest wird überprüft, ob das implementierte System mit der Spezifikation übereinstimmt, bevor die Software an den Kunden ausgeliefert wird.

4.1.1.5 Auslieferung und Wartung

Nominell ist die Software zu diesem Zeitpunkt fertig und wird beim Kunden in Betrieb genommen. Gerade bei komplexer, geschäftskritischer Software macht die Wartung aber einen erheblichen Teil der Aufwände aus, die für eine Software anfallen. Nicht nur müssen Fehler behoben werden, die erst im produktiven Betrieb bemerkt werden, sondern die Software muss auch an die sich ständig ändernden Anforderungen des Einsatzkontextes angepasst werden.

4.1.1.6 Vorgehensmodelle

Diese Basisaktivitäten werden in verschiedenen Vorgehensmodellen auf unterschiedliche Art und Weise zu einem Entwicklungsprozess kombiniert, der Teil eines Vorgehensmodells für die Softwareentwicklung ist. Beispiele für Vorgehensmodelle zur Softwareentwicklung sind etwa das Wasserfallmodell [Royce, 1970], Rational Unified Proces (RUP) [Kruchten, 2004, IBM, 2011], das V-Modell [BmI, 2011] oder auch Scrum [Scrum Alliance, 2011] und Extreme Programming [Beck, 1999] als agile Vorgehensweisen. Abb. 4.1 zeigt als vereinfachendes Beispiel links einen klassischen Wasserfallprozess, bei dem alle Aktivitäten streng in Folge abgewickelt werden, sowie ein Beispiel für einen agilen Prozess, bei dem die Entwicklung in einer Reihe kleinerer Iterationen stattfindet, in denen jeweils ein Subset der Gesamtfunktionalität entwickelt wird. Ein Vergleich verschiedener Vorgehensmodelle zur Softwareentwicklung kann hier nicht vorgenommen werden. Mehr zum Thema agile Softwareentwicklung findet sich beispielsweise in [Bleek und Wolf, 2010], im Kontext verteilter Softwareentwicklung auch in [Sauer, 2010].

Beide Prozesse sind natürlich idealisierte Darstellungen. Auch in einem klassischen Wasserfallprojekt ist ein Rückgriff auf frühere Projektphasen oft erforderlich, dies gestaltet sich nur in der Regel als schwieriger, weil es nicht explizit vorgesehen ist. Iterative Vorgehensmodelle, insbesondere agile Vorgehen

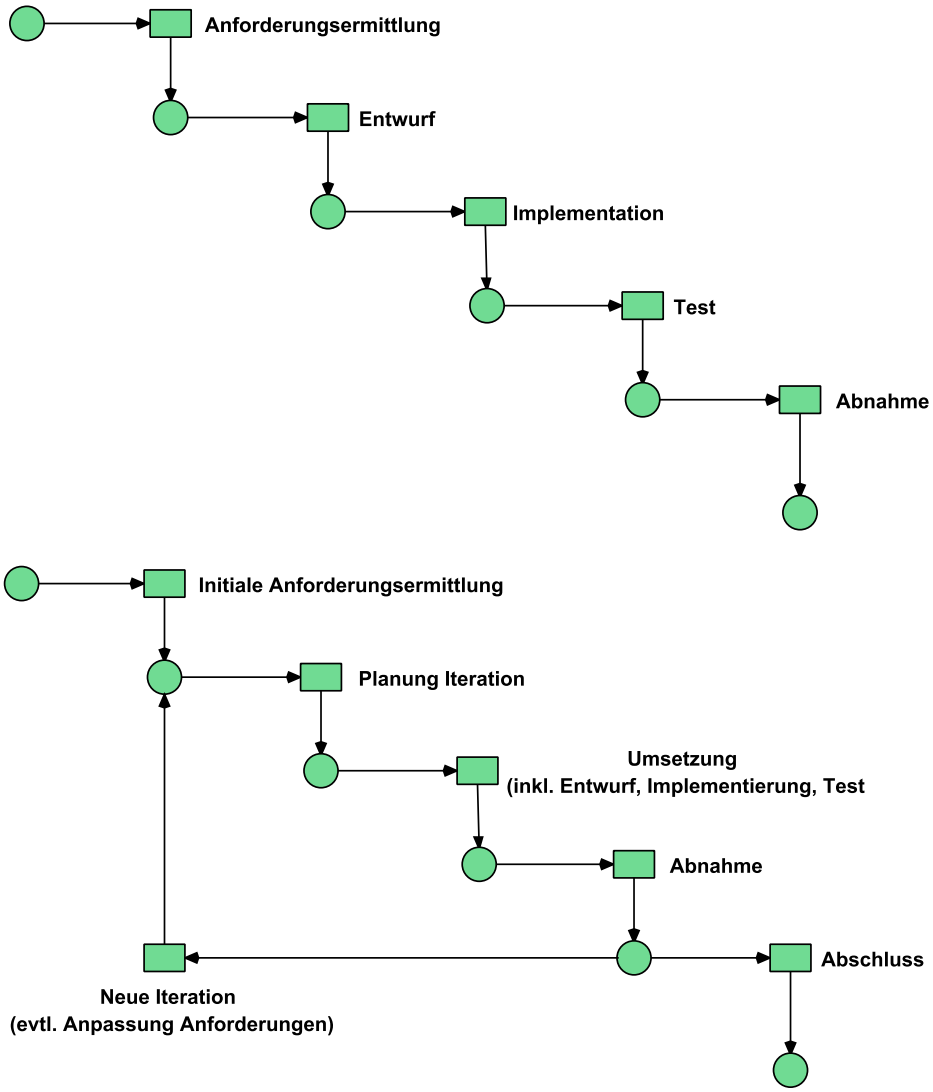

Abbildung 4.1: Wasserfallprozess vs. Iteratives Vorgehen

der Softwareentwicklung erlauben meist eine weitergehende Verzahnung zwischen den einzelnen Aktivitäten, daher ist die Umsetzung hier als eine atomare Operation dargestellt. Bei dieser Art von Vorgehensmodell ist es einfach möglich und oft sogar erwünscht, dass sich von Iteration zu Iteration der geplante Gesamtumfang verändert und auch der Entwicklungsprozess selbst durch das Entwicklungsteam angepasst wird.

Die dargestellten Prozesse beschreiben bisher nur die Abfolge der Aktivitäten, nicht aber die hierbei beteiligten Personen, Rollen, Werkzeuge und Artefakte. Diese Aspekte werden in späteren Modellen wieder aufgegriffen.

4.1.2 Softwarequalität

Das gewählte Vorgehen zur Softwareentwicklung muss sich am Ergebnis messen lassen. Daher ist es wichtig, Qualitätsmerkmale für Software zu definieren, um Vorgehensmodelle und Werkzeuge daran zu messen, wie sie dabei helfen können, qualitativ hochwertige Software zu erzeugen [Ghezzi et al., 2002, S. 17ff]. Dafür sind eine Reihe von Qualitätsmerkmalen relevant, von denen im Folgenden einige ausgeführt werden.

4.1.2.1 Korrektheit

Unter (funktionaler) Korrektheit versteht man die Anforderung an eine Software, alle Funktionsbestandteile so umzusetzen, wie sie in der Spezifikation vorgesehen sind. Die Spezifikation muss nicht vollständig sein, oder sie kann Anforderungen enthalten, die so nicht den tatsächlichen Anforderungen der Anwender entsprechen. Korrektheit bedeutet eine Äquivalenz zwischen Spezifikation und Implementation, die sich verhältnismäßig gut durch Tests verifizieren lässt.

Zudem lässt sich die Korrektheit einer Software auch durch formale Verfahren verifizieren. Für größere Softwaresysteme ist dies aber in der Regel zu aufwändig, daher wird dies meist nur für spezielle, besonders kritische Bereiche durchgeführt.

4.1.2.2 Verlässlichkeit

Verlässlichkeit ist weniger exakt definiert als Korrektheit. Diese Eigenschaft beschreibt, dass eine Software sich so verhält, wie der Anwender dies erwartet. Eine Software kann aus verschiedenen Gründen unzuverlässig sein. Es kann sein, dass das spezifizierte (und implementierte) Verhalten nicht mit den Erwartungen der Benutzer übereinstimmt, dann liegt ein Fehler bei der Spezifikation vor. Fehler bei der Implementierung können zu unerwünschtem Verhalten führen, oder bestimmte Bereiche der Software waren schlicht unvollständig spezifiziert.

Verlässlichkeit ist weit schwieriger durch Tests festzustellen, da nicht einfach die Punkte eines Pflichtenheftes abgetestet werden können. Zudem ist nicht immer eindeutig zu entscheiden, ob ein bestimmtes Verhalten als Fehlverhalten zu werten ist, wenn für diesen Fall keine klare Spezifikation vorliegt.

4.1.2.3 Robustheit

Robustheit beschreibt das Verhalten einer Software unter Bedingungen, die nicht in den Anforderungsdokumenten vorgesehen waren. Bei fehlerhaften Eingaben, Hardwaredefekten, Netzwerkfehlern etc. sollte sich das System angemessen verhalten und nicht beispielsweise Daten verlieren oder abstürzen. Dies betrifft nicht Eigenschaften, die bereits im Rahmen der Korrektheit berücksichtigt sind, die also explizit in den Anforderungen erwähnt wurden, sondern Ausnahmefälle, die nicht im Voraus bedacht wurden. Dementsprechend ist diese Eigenschaft schwer zu überprüfen.

4.1.2.4 Performanz

Performanz bezeichnet die Ressourceneffektivität eines Programms. Insbesondere die Verarbeitungsgeschwindigkeit und der Speicherverbrauch sind hier wichtige quantitative Kriterien. Welche Werte hier angemessen für eine Software sind, hängt natürlich stark vom Einsatzkontext ab; interaktive Programme sollen in der Regel innerhalb einer Sekunde oder weniger eine Rückmeldung liefern (je nach Aufgabe kann mehr oder weniger akzeptabel sein), Webanwendungen dürfen in der Regel etwas länger brauchen, bei Stapelverarbeitungsprozessen müssen eine bestimmte Anzahl von Datensätzen pro Minute verarbeitet werden etc.

Die Performanz ist meist erst dann ein Thema, wenn die Grundfunktionalität sichergestellt ist, wenngleich auch bei der Entwicklung immer die Komplexität der verwendeten Algorithmen beachtet werden sollte, um nicht unnötige Ressourcenverschwendung zu betreiben. Bereiche, die sich beim Test als Engpässe herausstellen, können erneut überarbeitet und optimiert werden.

4.1.2.5 Usability

Usability oder Benutzerfreundlichkeit bezeichnet, wie einfach das System durch einen Benutzer zu bedienen ist. Dieser Aspekt ist subjektiv und stark abhängig vom jeweiligen Benutzer und dem Einsatzkontext. Ein Maß für die Usability ist die Konsistenz und Erwartungskonformität der Benutzungsoberfläche. Eine Software zum Beispiel, die den gleichen Benutzungsmustern folgt wie andere, dem Benutzer bekannte Software, ist leichter zu erlernen und damit benutzerfreundlicher als eine, die ein völlig eigenes Bedienkonzept verfolgt.

4.1.2.6 Wartbarkeit

Unter dem Warten einer Software ist naturgemäß nicht das Gleiche zu verstehen, wie bei einer Maschine, da Software keinen Verschleiß oder Materialermüdung aufweist. Dennoch ist es in der Regel erforderlich, eine Software regelmäßig zu erweitern, an neue Anforderungen anzupassen, oder Lücken und Fehler der ursprünglichen Spezifikation zu korrigieren, daher spricht man auch von Software Evolution.

Wartbarkeit lässt sich unterteilen in Reparierbarkeit und Erweiterbarkeit. Ein System ist gut reparierbar, wenn Fehler leicht zu finden und zu beheben sind. Erweiterbarkeit bezeichnet, wie leicht es ist, die Software um neue Funktionalitäten zu erweitern. Voraussetzungen für beides sind unter anderem ein sinnvoller Grad der Modularisierung, Werkzeuge für die Fehlersuche und Behebung (Debugger) und Dokumentation der getroffenen Implementationsentscheidungen und der verwendeten Schnittstellen.

Mangelnde Wartbarkeit kann dazu führen, dass es ökonomischer ist, eine prinzipiell funktionsfähige Software durch eine neue zu ersetzen, weil zukünftige Änderungen nicht oder nur zu unverhältnismäßig hohen Kosten umgesetzt werden können.

4.1.2.7 Wiederverwendbarkeit

Verwandt mit dem Begriff der Wartbarkeit ist der der Wiederverwendbarkeit. Anstatt die Software weiterzuentwickeln, kann sie auch als Bestandteil einer anderen Software eingesetzt werden. Dies kann auch einzelne Bestandteile der Software betreffen, die in neuen Anwendungen wiederverwendet werden.

Auf diese Weise können die Kosten zur Erstellung einer Software deutlich reduziert werden, indem auf bestehende Komponenten zurückgegriffen wird, die sich bereits in anderem Kontext bewährt haben. Voraussetzung hierfür ist, die Schnittstellen eines Moduls zu anderen Modulen innerhalb einer Software möglichst klar und generisch zu gestalten.

4.1.2.8 Portabilität

Gerade in verteilten Umgebungen kommt es häufig vor, dass eine Software in unterschiedlichen Umgebungen ausgeführt werden muss. Eine portable Software lässt sich in verschiedenen Umgebungen ausführen. Dies kann erreicht werden durch unterschiedliche Compiler für verschiedene Zielumgebungen, durch systemunabhängige Sprachen (z.B. Skriptsprachen oder durch die Verwendung eines Bytecode, wie etwa bei Java) oder durch eine spezielle systemabhängige Abstraktionsschicht, auf der dann der systemunabhängige Code aufsetzen kann.

4.1.2.9 Interoperabilität

Interoperabilität bedeutet, dass ein Softwaresystem mit anderen Systemen zusammenarbeiten kann. Zum Beispiel kann ein Textverarbeitungsprogramm eine Grafik einbinden, die mit einem externen Grafikprogramm erzeugt wurde etc. Interoperabilität wird am Besten durch klar definierte, standardisierte Schnittstellen erreicht. Offene Systeme bauen auf dem Gedanken der Interoperabilität auf. Diese Systeme sind so entworfen, dass neue Komponenten über offene Schnittstellen in das System eingebracht werden können, um so aus Komponenten verschiedener Hersteller ein Gesamtsystem zu erstellen, dass die gewünschte Funktionalität erbringt.

4.1.2.10 Produktivität

Unter der Produktivität des Softwareentwicklungsprozesses versteht man die Geschwindigkeit, mit der die Entwicklung voranschreitet. Je effizienter der Prozess ist, desto schneller kann die Software fertig gestellt werden.

Die Produktivität eines einzelnen Entwicklers kann sehr stark schwanken und ist abhängig unter anderem von seiner Ausbildung und Erfahrung mit den angewendeten Technologien, den verfügbaren Werkzeugen, Erfahrungen in ähnlichen Projekten etc. Ein effizienter Softwareentwickler kann leicht die mehrfache Produktivität eines weniger effizienten Kollegen erreichen[Brooks, 1995, S. 30].

Bei der Planung eines Projektes ist es wichtig zu bedenken, dass sich in einem Team nicht einfach die Produktivität der einzelnen Teammitglieder mit deren Anzahl multiplizieren lässt. Vielmehr sinkt die Produktivität stark mit der Teamgröße. Dies ist umso stärker spürbar, je größer die Abhängigkeiten zwischen den einzelnen Entwicklern ist. Eine sinnvolle Modularisierung und die Verwendung wiederverwendbarer Komponenten kann die Produktivität deutlich erhöhen.

4.1.2.11 Pünktlichkeit

Auch Pünktlichkeit bezieht sich mehr auf den Softwareentwicklungsprozess als auf die Software selbst. Die beste Software nützt nichts, wenn sie nicht rechtzeitig ausgeliefert werden kann. Dies kann so weit führen, dass ein Unternehmen nicht sinnvoll arbeiten kann, weil die Software hierfür zwingend notwendig ist. Auch können sich im Laufe einer zu langen Entwicklungszeit die Anforderungen bereits geändert haben.

Um die Pünktlichkeit der Softwareentwicklung zu gewährleisten, ist eine gute Planung des gesamten Prozesses erforderlich, aufbauend auf verlässlichen Maßen für die Produktivität des Entwicklungsteams. Oft ist es besser, eine in der Funktionalität reduzierte Version der Software rechtzeitig auszuliefern und Erweiterungen auf spätere Releases zu verschieben, als zu versuchen, alles auf einmal zu schaffen und das gesamte Projekt zu verzögern.

4.1.2.12 Qualitätskennzahlen

Da Softwarequalität einerseits ein wichtiges Ziel, andererseits aber schwer messbar ist, wurden eine Reihe von Versuchen unternommen, Standards hierfür zu entwickeln. Ein allgemeiner Standard für Qualitätssicherung in industriellen Prozessen ist die Norm ISO 9001. Eine Zertifizierung unter dieser Norm bescheinigt einer Organisation, dass sie entsprechende Prozesse etabliert hat, um reproduzierbar Produkte einer gewissen Qualität zu erstellen. Der Standard umfasst organisatorische Aspekte wie ein etabliertes Qualitätsmanagement mit entsprechender Dokumentation, Anforderungen an die Ressourcen, die im Prozess verwendet werden, sowie an den Softwareerstellungsprozess selbst und Mechanismen zur laufenden Verbesserung dieses Prozesses.

Allerdings ist eine ISO 9001 Zertifizierung ein langwieriger und kostspieliger Prozess, den viele Softwarefirmen scheuen, zudem ist der Standard hauptsächlich auf industrielle Produktionsprozesse ausgerichtet, so dass die Anwendung auf Softwareerstellung mit gewissen Schwierigkeiten verbunden ist. Daher wurden andere Modelle entwickelt, die sich direkter auf das Gebiet der Softwaretechnik beziehen.

Eines der bekanntesten Systeme zur Beurteilung der Prozessqualität in der Softwaretechnik ist das Capability Maturity Model Integration (CMMI), der Nachfolger des ursprünglichen Capability Maturity Model (CMM). Dieses Modell stuft Organisationen in fünf „Reife“-Stufen ein. Unternehmen können entscheiden, für welche Stufe sie eine Zertifizierung anstreben.

Stufe eins (Initial) bedeutet, dass keine Maßnahmen zur Qualitätssicherung vorgenommen worden sind. Stufe zwei (Managed) bescheinigt eine Wiederholbarkeit des Prozesses; Projektplanung und -management sind implementiert, es wird ein Anforderungs- und Qualitätsmanagement vorgenommen, und Konfigurationsmanagement-Werkzeuge werden verwendet. Auf diese Weise ist die Organisation in der Lage, ihren bisherigen Prozess hinsichtlich Kosten und Ressourcen zu überblicken und ähnliche Projekte verlässlich durchzuführen.

Für Stufe drei (Defined) muss der Prozess über die gesamte Projektpalette der Organisation geregelt werden. Wenn klare Prozesse für Entwicklung und Wartung über die gesamte Organisation implementiert und dokumentiert sind, kann diese Ebene erreicht werden.

Stufe vier (Quantitatively managed) bedeutet, dass Prozess- und Produktqualität quantitativ gemessen und überwacht werden. Verschiedene Kennzahlen werden routinemäßig erfasst und ausgewertet. Dadurch sind Kosten, Zeiten und Qualität kontrollierbar.

Stufe fünf (Optimizing) schließlich bedeutet, dass die gesamte Organisation auf die beständige Optimierung des Prozesses ausgerichtet ist [Vliet, 2008, S. 137ff].

4.1.3 Prinzipien des Software Engineering

Bei der Entwicklung von Software haben sich eine Reihe von Prinzipien herausgestellt, deren Einhaltung hilft, die vorgenannten Qualitätsmerkmale zu erzielen. Auf der einen Seite werden diese Prinzipien bei der Entwicklung mit dem PAOSE Ansatz berücksichtigt, auf der anderen Seite wird darauf eingegangen, wie POTATO bei der Einhaltung dieser Prinzipien helfen kann. Diese Darstellung folgt [GHEZZI et al., 2002, S. 56ff].

4.1.3.1 Gründlichkeit und Formalität

Die Erstellung von Software ist ein kreativer Prozess, bei dem die Intuition des Entwicklers oft eine wichtige Rolle spielt. An vielen Stellen ist es aber erforderlich, erprobte, klar definierte Prozesse zu verwenden, um zuverlässige und effiziente Ergebnisse zu erzielen. Der höchste Grad dieser Gründlichkeit ist die Verwendung von formalen Methoden zur Spezifikation und Implementation.

Je kritischer eine Komponente innerhalb einer Software ist, umso gründlicher muss bei ihrer Erstellung vorgegangen werden. Während sich ein Entwickler bei der Gestaltung einer Benutzungsoberfläche in weiten Teilen auf seine Intuition verlassen kann, ist bei der Implementation komplexer Geschäftslogik zumindest eine exakte textuelle Beschreibung der Funktionalität erforderlich. Kritische Teile, wie etwa eine Sicherheitskomponente können sogar eine formale Überprüfung erfordern.

Der PAOSE-Ansatz unterstützt dieses Prinzip sehr gut. Auf der einen Seite erlaubt die grafische Implementation eine intuitive Herangehensweise. Auf der anderen Seite werden verschiedene formal basierte Werkzeuge verwendet, wie etwa Netzkomponenten und die automatische Generierung von Protokollen und Ontologieklassen. An kritischen Stellen können überdies bewährte Verifikationsmethoden aus dem Bereich der Petrinetze verwendet werden.

4.1.3.2 Separation of Concerns

Unter Trennung der Zuständigkeiten versteht man das Entwurfsprinzip, dass jeder Teil eines Programms genau eine Aufgabe hat und nicht mehr. Das erlaubt, sich bei der Implementation auf genau diesen Aspekt zu konzentrieren und erleichtert später das Verständnis des Codes. Nachträgliche Änderungen müssen nur an den (wenigen) entsprechenden zuständigen Stellen durchgeführt werden und haben nach Möglichkeit keine Nebenwirkungen auf andere Teile der Software.

Diese Trennung kann auf verschiedenen Ebenen stattfinden. Verschiedene Qualitätskriterien (etwa Korrektheit und Performanz), verschiedene Sichten auf die Software (Ablauf, Struktur, Daten) oder verschiedene funktionale Bestandteile werden separat betrachtet. Dabei ist es wichtig, die Aufteilung nicht zu fein und nicht zu grob vorzunehmen. Eine zu grobe Trennung führt dazu, dass zu viele Aspekte in einer Einheit betrachtet werden müssen, eine zu feine

Aufteilung führt zum Verlust von Synergien, die bei gemeinsamer Behandlung erzielt werden könnten.

Bei PAOSE erfolgt die Trennung der Zuständigkeiten hauptsächlich anhand der Agentenrollen, Interaktionen und Ontologien. Jede Agentenrolle steht für sich, ebenso wird jede Interaktion für sich betrachtet, ebenso die Ontologie. Die notwendigen Berührungspunkte zwischen diesen drei Aspekten und die daraus resultierenden Kommunikationsbedarfe sind essentiell im Ablauf eines PAOSE-Projektes.

4.1.3.3 Modularisierung

Bei der Erstellung komplexer Systeme ist es von Vorteil, diese in kleinere Teile oder Module aufzuteilen. Im Kontext des Software Engineering erlaubt Modularisierung, ein komplexes System in einfachere Teile zu zerlegen, aus einfachen Teilen ein komplexes System aufzubauen, das Verständnis eines Systems zu erleichtern, indem seine Einzelteile betrachtet werden und die Modifikation eines Systems zu vereinfachen, da immer nur bestimmte Module angepasst werden müssen.

Bei der Zerlegung eines Systems in Module sind hoher Zusammenhalt und geringe Kopplung die zwei zentralen Kriterien für die Aufteilung in Module. Die Elemente innerhalb eines Moduls sollen stark miteinander zusammenhängen, während zwischen verschiedenen Modulen möglichst geringe Abhängigkeiten bestehen sollen.

Bei PAOSE sind die Module wie erwähnt die Agentenrollen und Interaktionen, die jeweils eine zusammengehörige Einheit bilden. Die Kopplung zwischen Agenten und Interaktionen wird geregelt durch die Vorgaben des Interaktionsdesigns, wo festgelegt wird, welche Entscheidungskomponenten und Wissensbasiseinträge ein Agent innerhalb einer Interaktion benötigt. Deren Ausgestaltung ist dann weitgehend unabhängig von der Interaktionsentwicklung selbst.

4.1.3.4 Abstraktion

Im Verlauf des Entwicklungsprozesses wird eine Reihe verschiedener Modelle erstellt, die jeweils eine Abstraktion des Anwendungskontextes und der zu erstellenden Software darstellen. Auf oberster Ebene wird nur spezifiziert, welche Funktionsbestandteile das System im Groben haben soll, während von der Implementation komplett abstrahiert wird.

Im Agenten- und Interaktionsmodell werden sowohl die verschiedenen Rollen spezifiziert als auch welche Interaktionen vorkommen, bzw. was diese tun. Vom inneren Aufbau wird jeweils abstrahiert. Der modellbasierte Ansatz von PAOSE mit immer weiter verfeinerten Modellen hilft hier, auf jeder Ebene den richtigen Abstraktionsgrad zu finden. Verschiedene Werkzeuge helfen dabei, von einem Modell zum nächsten, genaueren zu kommen, indem beispielsweise

aus einem Interaktionsdiagramm die (detaillierteren) Agentenprotokolle automatisch generiert werden.

4.1.3.5 Anticipation of Change

Bei der Erstellung von großen Softwaresystemen muss berücksichtigt werden, dass sich im Verlauf des Lebenszyklus der Software meist eine Reihe von Änderungen ergeben, die an der Software vorgenommen werden müssen. Anticipation of Change bedeutet, dass im Vorfeld bereits überlegt wird, welche Aspekte mit der größten Wahrscheinlichkeit Änderungen unterworfen sein werden. Diese Aspekte sollten so implementiert werden, dass Änderungen mit möglichst geringem Aufwand möglich sind, zum Beispiel durch eine sinnvolle Modularisierungsstrategie und die Verwendung von Entwurfsmustern [Gamma et al., 1995]. Dies hilft bei der Verfolgung der Qualitätsziele Wartbarkeit und Wiederverwendbarkeit.

Ein Beispiel für Änderungen während des Lebenszyklus einer Software ist der Prozess, der für einen bestimmten Geschäftsvorfall abläuft. Wenn davon ausgegangen wird, dass dieser Prozess sich nie ändern wird, kann er durch die getroffenen Annahmen bei der Implementation fest verankert werden. Auf der anderen Seite bieten Workflowmanagementsysteme die Möglichkeit, den Prozess als eigene Entität zu isolieren und leicht änderbar zu machen.

Der Paose Ansatz legt seinen Fokus auf die Aspekte, bei denen von häufigen Änderungen ausgegangen wird. Agenten und Agentenrollen abstrahieren von der Struktur des Systems, von Benutzern und Ressourcen, die im System vorkommen. Neue Agenten können leicht erzeugt und ins System eingebracht werden, die Struktur kann jederzeit durch das Hinzufügen neuer Plattformen erweitert werden. Auch die Abläufe sind flexibel gestaltet, da sie durch Interaktionen explizit gemacht sind und jederzeit an einer Stelle geändert werden können. Die Prozessinfrastruktur erlaubt zudem die Anpassung von Geschäftsprozessen auf der Anwendungsebene, während die Werkzeugumgebung das Ziel hat, die Ausstattung der Benutzer mit Arbeitsmitteln und -materialien zu flexibilisieren.

4.2 Verteilte Softwareeentwicklung

Unter verteilter (kollaborativer) Softwareentwicklung versteht man Softwareprojekte, die durch verteilte Teams durchgeführt werden. Die Mitglieder dieser Teams können für verschiedene Organisationen und an verschiedenen Orten arbeiten. Solche Projekte können auf vielfältige Weise organisiert sein. Organisationale und technische Rahmenbedingungen bezüglich der Kommunikations- und Kooperationswerkzeuge, können einen erheblichen Einfluss auf den Verlauf solcher Projekte haben [Loftus et al., 1996, S. 5ff].

Die Gründe, warum Software verteilt entwickelt wird, sind vielfältig. Beispielsweise können sich ökonomische Vorteile erzielen lassen, indem Teile der Arbeit in anderen Ländern durchgeführt werden (Outsourcing). Es kann aber auch eine notwendige Folge aus der entwickelten Software sein, etwa wenn die Verteilung des entwickelten Produktes es nahelegt, auch die einzelnen Komponenten jeweils dort zu entwickeln, wo sie eingesetzt werden sollen.

Verteilte Softwareentwicklung ist häufig in Opensource-Projekten zu finden, bei denen sich Freiwillige in virtuellen Arbeitsnetzen zusammenfinden, um gemeinsam an einem Projekt zu arbeiten [Brand, 2009, S. 9]. Aber auch in kommerziellen Softwareprojekten sind verteilte Teams keine Seltenheit, wenn verschiedene Teilaufgaben eines Projektes von unterschiedlichen Organisationen bearbeitet werden oder für bestimmte Projekte organisationsübergreifende Teams neu gebildet werden.

Aus der Verteilung ergeben sich eine Reihe von Herausforderungen für die Organisation und Durchführung eines Softwareprojektes. Die durchzuführenden Tätigkeiten müssen sinnvoll zwischen den Beteiligten aufgeteilt und Abhängigkeiten berücksichtigt werden, die geschaffenen Artefakte müssen verfügbar gemacht und verteilt bearbeitet werden.

Vor allem aber werden Kommunikationsaufgaben, die einen erheblichen Teil der Softwareentwicklung ausmachen, erheblich erschwert, wenn sich die Beteiligten an unterschiedlichen Orten befinden. Das betrifft zum Beispiel die Abstimmung von Anforderungen und Entwurfsentscheidungen, aber auch tagtägliche Fragen etwa zur Benutzung einer bestimmten Bibliothek oder Klarifikation bezüglich unklarer Anforderungen. In diesen Fällen können rechnergestützte Kollaborationswerkzeuge helfen, die Arbeit zu erleichtern.

Im Folgenden wird kurz auf verteilte Systeme eingegangen, bevor Serviceorientierte Architekturen als wichtiges Anwendungsgebiet verteilter Software näher beleuchtet und gegenüber Agentensystemen abgegrenzt werden. Anschließend werden Aspekte der Organisation verteilter Softwareprojekte betrachtet und diskutiert, welche Werkzeuge generell für die Unterstützung dieser Art von Projekten erforderlich sind.

4.2.1 Verteilte Systeme

In vielen Fällen ist es notwendig, dass eine Software nicht nur auf einem Rechner ausgeführt wird, sondern erst durch die Interaktion verschiedener Rechner sinnvoll wird, die sich an unterschiedlichen Orten befinden. Durch die zunehmende Verbreitung schneller Netzwerkverbindungen nimmt der Anteil verteilter Software immer mehr zu.

Verteilte Software stellt die Entwickler vor eine Reihe von Herausforderungen, die in der Art bei nicht-verteilter Software nicht vorkommen können. Beispielsweise kann es vorkommen, dass bei einer verteilten Anwendung einer oder mehrere der Knotenrechner ausfallen, die Ausführung auf den anderen Rechnern aber fortgesetzt werden kann. Auch die Netzwerkverbindung kann

ganz oder teilweise ausfallen, unverhersehbare Nachrichtenlaufzeiten können dazu führen, dass Nachrichten nicht in derselben Reihenfolge empfangen werden, in der sie abgesendet wurden, oder dass einzelne Nachrichten komplett verloren gehen.

Bei der Betrachtung der Komplexität eines verteilten Algorithmus muss zusätzlich besonderes Augenmerk auf die Nachrichtenkomplexität, also die Anzahl der benötigten Nachrichten in Abhängigkeit von der Problemgröße, gelegt werden, weil Nachrichtenlaufzeiten oft die Zeit für Berechnungen innerhalb eines Knotenrechners um ein Vielfaches übersteigen.

Multiagentensysteme sind ein Beispiel für verteilte Systeme, da zum einen die Agenten innerhalb einer Agentenplattform sich als verteiltes System verhalten, zum anderen aber auch durch verteilte Agentenplattformen innerhalb eines Systems mehrere Knotenrechner im Multiagentensystem verbunden sind. Gemeinsam implementiert das Multiagentensystem eine Multiagentenanwendung, eine verteilte Anwendung.

4.2.2 Serviceorientierte Architekturen und Web Services

Das Internet ermöglicht den einfachen Zugriff auf Ressourcen außerhalb einer Organisation über Internet Protokolle. Viele Informationen werden im WWW zur Verfügung gestellt, zumeist aber in Form von HTML-Seiten, die zwar für menschliche Benutzer mit einem Browser gut lesbar sind, für eine automatisierte Verarbeitung aber weniger geeignet sind.

Um Informationen auch für eine automatisierte Verarbeitung besser zugänglich zu machen, wurde die Webservice Schnittstelle definiert, eine standardisierte Dienstschnittstelle, die Internetprotokolle (HTTP, TCP/IP) verwendet, um Daten zwischen Organisationen auszutauschen und Dienste und Informationen anzubieten. Diese Dienste lassen sich dynamisch komponieren und zu neuen Diensten und Anwendungen zusammenstellen.

4.2.2.1 Dienste

Der erste Grundbegriff, der hier definiert werden muss, ist der Begriff des Dienstes oder Service. Ein Dienst ist eine Softwareeinheit, die Klienten eine klar definierte Dienstleistung anbietet. Hierfür registrieren sie sich mit einer genauen Beschreibung der angebotenen Dienstleistung bei einer Service Registry, die das Matching zwischen Dienstanbietern und -nachfragern ermöglicht.

Dienste sind eigenständige Einheiten, deren Schnittstelle nach außen hin durch klar definierte Dienstverträge geregelt sind, so dass eine sehr lose Kopplung zwischen Diensten besteht. Die Kopplung ist so lose, dass prinzipiell ein Dienst jederzeit durch einen anderen ausgetauscht werden kann. Einfache Dienste sind allgemein und wiederverwendbar gestaltet und können zu komplexeren Diensten komponiert werden, die wiederum anderen Klienten angeboten werden können, ähnlich wie Komponenten.

Um die Interoperabilität von Diensten zwischen verschiedenen Anbietern zu gewährleisten, müssen sich Dienste an offene Standards halten, wie dies z.B. bei Web Services gegeben ist [Vliet, 2008, S. 641].

Web Services sind eine spezielle Implementation von Diensten, die auf Internetprotokollen basiert und von der W3C standardisiert ist [Booth et al., 2004]. Die hierbei verwendeten Standardformate werden in Abschnitt 4.2.2.3 ausgeführt.

4.2.2.2 Serviceorientierte Architektur

Eine Serviceorientierte Architektur (SOA) ist eine Softwarearchitektur, die auf der Komposition von Diensten basiert. Statt eine Anwendung von Grund auf zu implementieren, werden existierende Dienste, die weltweit verteilt angeboten werden können, neu zusammengestellt, unter Umständen sogar zur Laufzeit.

Dies ermöglicht auch völlig neue Geschäftsmodelle, etwa kurzfristige Partnerschaften, bei denen jeder Partner einige Dienste bereitstellt, um gemeinsam einen neuen Dienst anzubieten. Aber auch klassischere IT-Infrastrukturen können von einer Serviceorientierten Architektur profitieren. Selbst wenn alle verwendeten Dienste innerhalb einer Organisation selbst erstellt und verwaltet werden, können durch standardisierte Web Service Schnittstellen und robuste Protokolle Integrationsaufwände reduziert werden [Werner und Fischer, 2007].

4.2.2.3 Standards

Da der Hauptzweck von Web Services darin besteht, das Zusammenspiel von Softwaresystemen über Organisationsgrenzen hinweg zu ermöglichen, ist es zwingend erforderlich, die erforderlichen Schnittstellenformate so genau wie möglich zu standardisieren. Eine Reihe von Standardformaten regelt die Definition und das Zusammenspiel von Web Services:

Extensible Markup Language (XML) ist das Basisformat für den Datenaustausch zwischen Web Services. In XML können über Dokumenttypdefinitionen (DTDs) spezielle Datenformate (wie z.B. SOAP, WSDL und UDDI) definiert werden.

Simple Object Access Protocol (SOAP) dient dem Austausch von Nachrichten zwischen Web Services [Gudgin et al., 2007]. Es ist ein Standard-Nachrichtenformat auf XML-Basis, das unabhängig von spezifischen Programmiersprachen und Betriebssystemen den Austausch strukturierter Informationen wie z.B. Objekten und Funktionsaufrufen innerhalb einer Anwendung ermöglicht [Werner und Fischer, 2007].

Web Services Description Language (WSDL) ist die Sprache, mit der Web Services beschrieben werden [Chinnici et al., 2007]. Der Nachfrager eines Dienstes kann die Beschreibung verwenden, um Anfragen an den Dienstanbieter zu formulieren, die mit dessen Dienstschnittstelle kompatibel sind.

Universal Description, Discovery and Integration (UDDI) wird verwendet, um vorhandene Dienste innerhalb eines Systems aufzulisten [OASIS, 2005]. UDDI ermöglicht es, Dienste zu veröffentlichen, also in einem Verzeichnisdienst bekannt zu machen, Dienste zu finden und sich mit diesen Diensten zu verbinden [Cardoso et al., 2007].

4.2.2.4 Agenten und Services

Agentenorientierte und Serviceorientierte Softwareentwicklung ähneln sich in vielen Belangen. Beides sind Vorgehensmodelle zur Entwicklung heterogener, verteilter Systeme, in beiden Fällen werden einzelne Funktionsbausteine konstruiert, die dynamisch komponiert und über Koordinationsmechanismen (Servicekomposition, Interaktionsprotokolle) miteinander gekoppelt werden können, um neue Funktionalitäten zu erzielen. Daher überrascht es nicht, dass eine Reihe von Arbeiten sich mit der Integration dieser beiden Ansätze beschäftigt.

Ein Beispiel hierfür ist [Willmott et al., 2005], dort wird ein Gateway-Agent vorgestellt, der es ermöglicht, einen Capa-Agenten oder ein Multiagentensystem als Dienst innerhalb einer SOA-Architektur anzusprechen. Umgekehrt können natürlich auch Agenten Web Services nutzen, um ihre Ziele zu verfolgen.

Viele neue Ideen werden im Bereich der Multiagentensysteme entwickelt und dann auf Web Services übertragen, da für diese stabilere Implementationsumgebungen und bessere Werkzeugunterstützung existieren (z.B. [Gibbins et al., 2004, Hendler, 2001]).

Der Hauptunterschied zwischen Agenten und Diensten ist die Betrachtungsweise. Bei der agentenorientierten Softwareentwicklung wird die Autonomie und Intelligenz der Agenten stärker in den Vordergrund gestellt. Serviceorientierte Architekturen beschäftigen sich mehr mit Diensten, die sich verhältnismäßig deterministisch entsprechend einer Spezifikation verhalten. Auf der anderen Seite kann sich natürlich auch ein Agent entsprechend einer vereinbarten Spezifikation verhalten und einen bestimmten Dienst erbringen, während ein Dienst die gleichen Möglichkeiten zu autonomem Verhalten besitzt wie ein Agent, wenn er entsprechend implementiert wird. Insofern sind die Ausführungen, die in dieser Arbeit bezüglich Multiagentensystemen gemacht werden, in weiten Teilen auch übertragbar auf Serviceorientierte Architekturen.

4.2.3 Organisation verteilter Softwareentwicklung

Bei einem verteilten Softwareprojekt wird meist ein größeres Projekt aufgeteilt in mehrere Teilprojekte, die dann von verschiedenen Teams bearbeitet werden. Diese Teilteams können für sich gesehen lokale Teams sein, dann ist die Verteilung nur für die Integration der Module von Interesse, oder sie können ebenfalls verteilt sein. Diese Aufteilung muss keine 1:1 Entsprechung sein; ein Team kann an mehreren Teilprojekten arbeiten, Teilprojekte können sich überlappen oder für ein Teilprojekt können mehrere Teams zuständig sein, die eine Aufgabe entweder kooperativ oder auch im Wettbewerb miteinander bearbeiten.

Die verschiedenen Parteien, die an einem verteilten Softwareprojekt beteiligt sind, stammen oft aus unterschiedlichen Organisationen oder Organisationsteilen. Daraus resultieren Unterschiede zwischen den Projektteilnehmern, etwa bezüglich verfügbarer Daten und Werkzeuge, aber auch organisationaler Standards und Protokolle.

Einerseits erlauben es diese Unterschiede, Ziele zu verwirklichen, die keinem Teilnehmer alleine möglich wären, da gemeinsam ein viel größeres Spektrum an Qualifikationen abgedeckt werden kann. Andererseits aber können daraus auch Probleme entstehen. Soll ein Team beispielsweise auf den Arbeitsergebnissen eines anderen Teams aufbauen, welches andere Werkzeuge benutzt, kann es zu Schwierigkeiten kommen, deren Daten zu interpretieren.

Ebenso können Unterschiede in der Organisations- und Kommunikationsstruktur zu Problemen führen. Innerhalb einer Organisation sind die Arbeitsprozesse in der Regel klar definiert und eingespielt. Bei einer interorganisationalen Zusammenarbeit müssen aber erst neue Regeln und Prozesse etabliert werden. Der Zugriff auf Daten ist komplizierter, da Außenstehende in der Regel nicht einfach auf interne Daten zugreifen dürfen. Wenn Organisationen unterschiedliche Kooperations- und Kommunikationswerkzeuge verwenden, muss zunächst ausgehandelt werden, ob man sich auf einen der verwendeten Standards einigt, Schnittstellen definiert oder versucht, jeweils sein eigenes System zu verwenden und die Koordination informell abzuhandeln.

Ein wichtiger Faktor ist hierbei das Verhältnis der Partner zueinander: Handelt es sich um einen unabhängigen Zusammenschluss gleichberechtigter Partner, um ein Auftraggeber-Auftragnehmer-Verhältnis oder besitzt eine Seite Weisungsbefugnis über die andere und kann dieser daher die verwendeten Werkzeuge und Prozesse vorschreiben [Loftus et al., 1996, S. 7f].

Durch die Verteilung des Entwicklungsprozesses entstehen eine ganze Reihe von Problemen, die organisatorisch gelöst werden müssen. Für den Teilbereich des verteilten Requirements Engineering identifiziert [Gumm, 2009] organisationale, räumliche und zeitliche Aspekte, die zu koordinieren sind. [Kiel und Eng, 2003] nennen in einer Fallstudie die Faktoren Zeit, Sprache, Kultur, Machtgefälle und Vertrauen als entscheidend für das Scheitern eines verteilten Projektes.

4.2.3.1 Zeit

Probleme bei der globalen Verteilung eines Teams entstehen durch unterschiedliche Zeitzonen, in denen sich die Beteiligten befinden. Hierdurch können synchronisierte Abstimmungsprozesse nur in Randbereichen der Standardarbeitszeiten oder sogar außerhalb dieser erfolgen. Im Extremfall gibt es keine Überlappung der Arbeitszeiten, so dass Synchronisation nur mit viel Zeitverzug oder unter größerem organisatorischem Aufwand erfolgen kann.

Unterschiedliche Zeitzonen können aber auch als Chance angesehen werden, Aufgaben zwischen Teams auszutauschen, um effektiv einen längeren Arbeitstag und damit kürzere Durchlaufzeiten zu erreichen. Dafür ist es aber entscheidend, die Übergabepunkte zwischen den Teams sehr genau und unmissverständlich zu definieren. Das führt in der Regel zu umfangreicherer und detallierterer Dokumentation, als es sonst erforderlich wäre [Treinen und Miller-Frost, 2006].

4.2.3.2 Sprache

Probleme können auftreten, wenn die Mitglieder des verteilten Temas unterschiedliche Muttersprachen sprechen und es dadurch zu einer Sprachbarriere innerhalb des Teams kommt. Auch wenn eine gemeinsame Sprache (meist englisch) als Projektsprache verwendet wird, werden hierdurch Nicht-Muttersprachler in Telefonkonferenzen und ähnlichen Situationen benachteiligt, da sie im Allgemeinen mehr Schwierigkeiten haben, aktiv an einer Konversation mitzuwirken. Das kann dazu führen, dass vermehrt der schriftliche Austausch gesucht wird, was wiederum negative Auswirkungen auf die Effizienz der Kommunikation haben kann. Zudem führt eine solche indirekte Kommunikation häufiger zu Missverständnissen [Kiel und Eng, 2003].

4.2.3.3 Kultur

Ein Faktor, der oft schwer einzuschätzen ist, sind unterschiedliche kulturelle Hintergründe. Das kann Dinge betreffen wie die Art der Kommunikation untereinander und zu Vorgesetzten, Feiertage, Humor etc. Mangelnde Sensibilität kann dann schnell zu Missverständnissen und Vertrauensverlust innerhalb des Teams führen. Um diesen Problemen entgegenzuwirken können Kulturtrainings in der Anfangsphase eines Projektes helfen [Kiel und Eng, 2003, Treinen und Miller-Frost, 2006].

4.2.3.4 Machtgefälle

Probleme können sich auch ergeben, wenn die Entscheidungskompetenzen zwischen den Teammitgliedern ungleich verteilt sind. Oft gibt es eine zentrale Stelle, an der alle wichtigen Entscheidungen gefällt werden, die die Außenstellen dann auszuführen haben.

Hierbei kann es helfen, bewusst Entscheidungskompetenzen zu verlagern, z.B. durch Modularisierung, oder zumindestens die Entscheidungsfindungsprozesse transparent zu machen und alle Betroffenen einzubeziehen [Kiel und Eng, 2003].

4.2.3.5 Vertrauen

Alle oben genannten Faktoren bestimmen, in welchem Maß sich ein Vertrauensverhältnis innerhalb des Teams einstellen kann. Vertrauen der Teammitglieder untereinander ist von entscheidender Bedeutung für den reibungslosen Ablauf eines Projektes. Durch die physikalische Trennung fällt es schwerer, einen menschlichen Kontakt mit den entfernten Teammitgliedern aufzubauen.

Als wirkungsvollstes Mittel zum Aufbau von Vertrauen ist der persönliche Kontakt anzusehen. Treffen in Person und Austausch von Personen zwischen den verschiedenen Standorten und Teilgruppen führt oft zu einem besseren Verständnis und in Folge zu einem verbesserten Vertrauensverhältnis [Kiel und Eng, 2003, Treinen und Miller-Frost, 2006].

4.2.3.6 Abgrenzung

Eine Möglichkeit, die auftretenden Reibungen zu reduzieren, besteht darin, die Software in mehrere kleinere Module aufzubrechen. Diese Module sollten möglichst wenig Berührungs- und Abstimmungspunkte haben. Dadurch wird die erforderliche Kommunikation reduziert [German, 2003].

Lösungen für diese Probleme sind jedoch nicht immer einfach zu finden. In der Regel sind diese Lösungen eher organisationaler Natur und nur schwer durch Werkzeuge zu beheben. Die organisationalen Aspekte verteilter Zusammenarbeit werden in dieser Arbeit nicht weiter behandelt.

4.2.4 Werkzeugunterstützung

Für die Zusammenarbeit verteilter Teams ist eine Kompatibilität der verwendeten Datenformate und Kommunikationsmittel erforderlich. Natürlich verwendet beispielsweise ein Systemanalyst andere Werkzeuge als ein Programmierer oder ein Softwaretester, dennoch müssen alle in der Lage sein, ihre jeweiligen Arbeitsergebnisse auszutauschen, etwa durch die Verwendung von Standardformaten, wo immer dies sinnvoll möglich ist.

Auf technischer Seite wird versucht, die Interaktionsprozesse zwischen den verschiedenen Beteiligten bei einem Softwareprojekt im verteilten Fall möglichst gut zu unterstützen. Dafür können verschiedene Groupware-Werkzeuge verwendet werden. Im Folgenden werden einige Beispiele genannt.

4.2.4.1 Abstimmungsprozesse

Abstimmungsprozesse bei der verteilten Softwareentwicklung lassen sich unterscheiden in synchrone und asynchrone Prozesse. In [Boulila et al., 2003] wird ein Framework vorgestellt zur Abwicklung synchroner Abstimmungsprozesse. Meetings zur Diskussion von Design- und Implementationsdetails können hiermit abgewickelt werden. An vielen Stellen im Entwicklungsprozess ist es wichtig, sich mit mehreren Personen offen auszutauschen und gemeinsam neue Zwischenergebnisse zu generieren.

4.2.4.2 Sourcecode Verwaltung

Bei der gemeinsamen und insbesondere bei der verteilten Bearbeitung von Programmcode ist es wichtig, Konflikte bei der Bearbeitung der verschiedenen Dateien zu vermeiden. Sourcecode Verwaltungssysteme helfen dabei, indem sie allen Beteiligten Zugriff auf den aktuellen Entwicklungsstand erlauben, Konflikte vermeiden oder auflösen helfen und Änderungen an den Dateien dokumentieren, um so Änderungen nachvollziehen zu können. Zudem erlauben sie es, separate Entwicklungsstränge (Branches) aus der Hauptentwicklung auszugliedern und bei Bedarf wieder zusammenzuführen.

Einige häufig verwendete Systeme sind z.B.

- CVS (Concurrent Versioning System) [CVS, 2011]
- SVN (Subversion) [Subversion, 2011]
- Git [Git, 2011]

4.2.4.3 Workflow Management Systeme

Um die verschiedenen Aktivitäten des Entwicklungsprozesses und ihre Abhängigkeiten untereinander zu verwalten, werden Workflow Management Systeme (WfMS) verwendet. Jazz [Jazz, 2011a] etwa erlaubt es, verschiedene Standardprozesse zu definieren, nach denen bestimmte Vorgänge während der verschiedenen Phasen der Entwicklung ausgeführt werden.

4.2.4.4 Collaborative Editing

Kollaborative Editoren erlauben es, von mehreren Arbeitsplätzen aus gleichzeitig ein Dokument zu bearbeiten. Dadurch können beispielsweise das Pair-Programming oder interaktive Code-Reviews, wie es im Extreme Programming und anderen agilen Ansätzen praktiziert wird, auch von verteilten Arbeitsplätzen aus praktiziert werden.

4.3 Softwareentwicklungsumgebungen

Ziel dieser Arbeit ist die Konzeption und die prototypische Implementierung einer Umgebung zur verteilten Softwareentwicklung. In diesem Abschnitt wird daher zunächst der Begriff der Softwareentwicklungsumgebung genauer herausgearbeitet. Anschließend werden Anforderungen beschrieben, die an Entwicklungsumgebungen gestellt werden, sowohl für die Einzelplatzentwicklung wie auch für verteilte Entwicklung. Anschließend werden verschiedene Konzepte aufgeführt, die sich heutzutage bei Entwicklungsumgebungen finden lassen.

4.3.1 Begriffsklärung

Eine integrierte Entwicklungsumgebung (engl. Integrated Development Environment (IDE)) ist eine Sammlung von Werkzeugen zur Unterstützung der Anwendungsentwicklung. Bei der Entwicklung von Software sind eine Reihe von Aufgaben zu erledigen (z.B. Editieren von Programmtexten, Compilieren, Linken, Debugging etc.), für die jeweils eigene Werkzeuge existieren. Eine IDE vereinigt diese unter einem Dach, so dass die Produktivität der Entwicklung erhöht wird. Zusätzlich können erweiterte Funktionen angeboten werden, wie etwa Syntax-Highlighting, Code-Vervollständigung, Integration von Sourcecode Verwaltungssystemen, Refactoring-Funktionen und anderes [Nourie, 2005].

Einige Beispiele von IDEs sind

- Visual Studio
- Eclipse
- IntelliJ IDEA
- Netbeans

Die wichtigste Anforderung, die an eine IDE gestellt wird, ist, dass sie dem Benutzer ermöglichen muss, seine Arbeit zu erledigen. Je nachdem, wie diese Arbeit aussieht, gestaltet sich das sehr unterschiedlich. In der Regel umfasst dies mindestens die Bearbeitung von Quelltexten und anderen Dokumenten, Übersetzung und Binden zu ausführbarem Code und Funktionen zur Fehlersuche.

Die weitaus meisten heute verbreiteten Entwicklungsumgebungen bieten dem Anwender eine grafische Benutzungsoberfläche, in der er Programmtexte bearbeiten kann, sowie einen Zugriff auf die verschiedenen Werkzeuge, die für die weitere Verarbeitung dieser Programme erforderlich sind.

4.3.2 Werkzeuge zur Verteilten Softwareentwicklung

Verschiedene Softwareentwicklungsumgebungen integrieren inzwischen Werkzeuge zur Unterstützung von Teamarbeit und verteilter Entwicklung. Beispiele dafür sind etwa das Jazz-Projekt von Eclipse oder Microsoft Visual Studio Team System. Eine alternative Herangehensweise ist die Verwendung spezieller Kollaborationswerkzeuge, die parallel zu den normalen Entwicklungswerkzeugen für verschiedene Aufgaben verwendet werden.

4.3.2.1 Jazz

Jazz[Jazz, 2011a, Jazz, 2011b] ist eine Kollaborationsplattform zur Unterstützung des Softwareerstellungsprozesses, die von Rational/IBM auf Basis der Eclipse-Plattform entwickelt wird. Jazz selbst ist kein fertiges Produkt, sondern eine Plattform, auf der andere Produkte, wie z.B. Rational Team Concert[Rational, 2011] aufgebaut werden können.

Konzepte Ziel der Jazz-Plattform ist die Unterstützung von Teams bei der Softwareentwicklung, um den Entwicklungsprozess kollaborativer, produktiver und transparenter zu gestalten. Dies soll durch drei Aspekte erreicht werden:

- **Eine Offene Architektur**, die es erlaubt, verschiedene Werkzeuge in einer Plattform zu integrieren, die im Laufe des Projektlebenszyklus verwendet werden. Dies soll erreicht werden durch einen Satz von Plattformdiensten, die nach definierten Regeln verwendet und erweitert werden können. Dies folgt den Spezifikationen des Open Services for Lifecycle Collaboration Projektes (OSLC)[IBM, 2011, OSLC, 2011], einer Initiative zur Standardisierung von Ressourcen und Schnittstellen für verschiedene Produkte innerhalb des Softwarelebenszyklus.

  Die Daten, auf die die verschiedenen Werkzeuge zugreifen, sind entkoppelt von der Implementation der Werkzeuge und können über Internetprotokolle verteilt gespeichert und abgerufen werden.

- **Werkzeuge für die Teamunterstützung**, die als zentralen Fokus die Zusammenarbeit von Gruppen haben und auf der Jazz-Plattform aufgebaut sind. Dies umfasst Werkzeuge für das Requirements Engineering, die Entwicklung, Qualitätsmanagement und Projektplanung.

- **Beteiligung der Community** an der Entwicklung durch Offenlegung der Projektfortschritte und Beteiligung der Kunden an der Ausgestaltung der weiteren Entwicklung.

Funktionsumfang Eine Reihe von Produkten zur Unterstützung des Softwareerstellungsprozesses wurden auf der Basis von Jazz entwickelt. Die wichtigsten sind:

- **Rational Team Concert** ist eine teamfähige Softwareentwicklungsumgebung, mit Features zur Verfolgung von Workitems, Build-Prozessen, Versionskontrolle und agiler Planung.
- **Rational Requirements Composer** erlaubt eine kollaborative Erarbeitung von Requirements über verschiedene Werkzeuge.
- **Rational Focal Point for Project Management** ist eine Projektplanungs- und -organisationslösung für Softwareprojekte, in der Ressourcen und Aufgaben verwaltet und der Arbeitsfortschritt überwacht werden können.
- **Rational Quality Manager** unterstützt die Vorgänge zur Software-Qualitätssicherung. Tests können geplant, konstruiert und ausgeführt und die Ergebnisse entsprechend aufbereitet werden. Zusätzlich können Workflows für Change Management Prozesse gestartet werden, um gefundene Probleme zu behandeln.

Alle diese Komponenten können über einen Jazz Team Server zusammenarbeiten und ihre Daten miteinander austauschen. Clients können durch die offene Interaktionsstruktur auf verschiedene Arten implementiert sein, z.B. als Weboberfläche, Rich Client oder als Integration in Visual Studio.

4.3.2.2 Microsoft Visual Studio Team System

Team System[Microsoft, 2011] ist eine Erweiterung der Entwicklungsumgebung Visual Studio von Microsoft zur Unterstützung von Entwicklungsteams. Über einen Team Foundation Server können Projekte geplant, erstellt und verwaltet werden. Im System können Prozesse instanziiert werden, die Aktivitäten werden dann von den Teammitgliedern in den verschiedenen angeschlossenen Werkzeugen (z.B. Visual Studio, MS Project oder MS Office) bearbeitet.

Konzepte Visual Studio ist eine Softwareentwicklungsumgebung, die seit Jahren von Microsoft entwickelt wird. Team Systems ist die Erweiterung dieser IDE um Kooperationswerkzeuge.

Funktionsumfang Team Systems umfasst Werkzeuge für den gesamten Softwareentwicklungsprozess, von Design, Architektur über Programmierung und Datenbankentwicklung bis zur Qualitätssicherung. Visual Studio Team Systems bietet zum einen die Einzelplatzfunktionalitäten einer normalen Entwicklungsumgebung, zum anderen Werkzeuge für die Unterstützung von Teamarbeit und verteilter Entwicklung. Eine Versionskontrolle erlaubt die Verwaltung von Quellcode und Projektdateien, Buildprozesse und Tests können auf Teamebene konfiguriert werden. Aufgaben für die verschiedenen Teammitglieder können im System erstellt und verfolgt werden und Berichte über den Projektfortschritt etc. aus dem System generiert werden.

4.3.2.3 Opensource-Tools zur Teamunterstützung

Eine Alternative zur Verwendung einer integrierten Entwicklungsumgebung mit eingebauter Teamunterstützung besteht darin, für die verschiedenen Kommunikations- und Koordinationsbedarfe jeweils eigenständige Werkzeuge zu verwenden. Viele dieser Werkzeuge stammen aus dem Opensource-Bereich und werden dort erfolgreich eingesetzt, um große, verteilte Projekte zu organisieren.

Beispiele für solche Programme zur Kooperationsunterstützung sind:

- **Editoren/Entwicklungsumgebungen** Zur Bearbeitung von Quellcode können verschiedenste Editoren verwendet werden, oft wird aber gleich eine Entwicklungsumgebung verwendet, wie etwa Eclipse, die auch das Kompilieren, Binden etc. übernimmt.

- **Sourcecode Verwaltung** Zur Verwaltung verschiedener Dateiversionen und zur Unterstützung der nebenläufigen Bearbeitung werden verschiedene Werkzeuge verwendet, wie etwa CVS[CVS, 2011], SVN[Subversion, 2011] oder Git[Git, 2011].

- **Forum** Foren bieten die Möglichkeit einer asynchronen Kommunikation und sind gut geeignet, um längerfristige Fragestellungen zu erörtern. Darüber hinaus bieten sie die Möglichkeit in früheren archivierten Diskussionen nach Informationen zu suchen. Ein bekanntes Beispiel hierfür ist phpBB [Jira, 2011b].

- **Wiki** Ein Wiki ist ein System, in dem sich Texte auf einfache Art speichern und editieren lassen. Dadurch kann projektbezogene Dokumentation leicht von allen Teammitgliedern gepflegt werden. Insbesondere bei häufig wechselnden Teammitgliedern bieten Wikis und Foren eine gute Möglichkeit, neue Teammitglieder in ein Projekt zu integrieren.

- **Bugtracker** Für die Verwaltung von Problemen und Anforderungen in einem Softwareprojekt ist es sinnvoll, spezialisierte Werkzeuge wie etwa Bugzilla [Bugzilla, 2011] oder Jira [Jira, 2011a] zu verwenden.

- **Sonstige Kommunikation** Ohne spezialisierte Kooperationswerkzeuge und feste Hierarchien, bzw. für alles, was von diesen Werkzeugen nicht adäquat erfasst werden kann, können allgemeine Tools wie Email und Chatprogramme verwendet werden. Je nach Projektgröße und -verteilung kommen auch Mailinglisten oder Telefon in Frage.

Der Vorteil der Verwendung verschiedener einzelner Programme ist die Flexibilität in der Auswahl der spezifischen Werkzeuge für jede Aufgabe, so dass man sich jeweils das am besten geeignete auswählen kann. Auf der anderen Seite kann es zu Problemen kommen, wenn die verschiedenen Aufgabenbereiche der Programme sich überschneiden und Inkonsistenzen auftreten. Auch muss ein

neues Werkzeug zunächst eingeführt und für das jeweilige Projekt eingerichtet werden. Ein neues Teammitglied muss unter Umständen die Bedienung einer ganzen Reihe von Werkzeugen zunächst erlernen.

Auf der anderen Seite ist es inbesondere bei Opensource-Projekten oft so, dass Mitgliederzahlen stark schwanken und der Grad der individuellen Beteiligung sehr unterschiedlich ist. Jeder Einzelne bringt dort das ins Projekt ein, was er bereit ist, beizusteuern. Daher müssen die verwendeten Werkzeuge frei verwendbar sein und dürfen nicht zu komplex zu installieren und zu bedienen sein. Webbasierte Werkzeuge sind daher für viele Aufgaben ideal.

4.3.3 Kriterien zur Beurteilung verteilter Softwareentwicklungsumgebungen

Dieser Abschnitt beschreibt Kriterien, anhand derer verteilte Softwareentwicklungsumgebungen beurteilt werden können. An diesen Kriterien wird die im Rahmen dieser Arbeit konzipierte Entwicklungsumgebung gemessen.

Aus Sicht eines Unternehmens, das Software entwickelt, sind unter anderem folgende Kriterien zur Entscheidung über das Vorgehen zur Entwicklung von Bedeutung:

1. **Anwendbarkeit** Die beste Umgebung nützt nichts, wenn die Entwickler nicht in der Lage sind, sie zu verwenden, oder wenn die Entwicklung nicht in der Sprache unterstützt wird, in der die vorhandene Codebasis vorliegt.

2. **Produktivität** Eine Softwareentwicklungsumgebung soll die Produktivität der Softwareentwicklung erhöhen und so die Kosten senken, die das Unternehmen aufbringen muss.

3. **Qualität der entwickelten Software** Wenn eine Softwareentwicklungsumgebung es erlaubt, beispielsweise durch die Verwendung spezieller Werkzeuge, höherwertige Software zu erstellen, als dies ohne diese möglich wäre, ergibt sich daraus unmittelbar ein Wettbewerbsvorteil für das Unternehmen.

4.3.3.1 Anwendbarkeit

In einer gegebenen Situation muss für eine Softwareentwicklungsumgebung zu allererst geprüft werden, ob sie grundsätzlich geeignet ist, die anstehenden Aufgaben zu bearbeiten. Damit ist gemeint, dass die verwendeten Technologien und Programmiersprachen durch die Umgebung unterstützt werden müssen, eine noch so gute Java-IDE nützt nichts, wenn die Entwicklung in C++ erfolgen soll.

Es gibt aber auch andere Rahmenbedingungen, die eine bestimmte Art von Entwicklungsumgebung erforderlich machen können, bzw. durch eine bestimmte Art von Entwicklungsumgebung erst ermöglicht werden. Projektgröße, -komplexität und geographische Verteilung sind beispielsweise Faktoren, die eine verteilte Entwicklungsumgebung zwingend erforderlich machen.

Bei einem großen, international verteilten Softwareprojekt wird es schlicht nicht mehr möglich sein, die Entwicklung ohne spezielle Werkzeuge erfolgreich zu planen, zu koordinieren und durchzuführen. Qualitätsmerkmale der Entwicklungsumgebung sind damit die Größe von Projekten, die damit erfolgreich durchgeführt werden können, ebenso wie die mögliche Heterogenität des Teams.

Ein nicht unerheblicher Anteil von Softwareprojekten scheitert, bzw. überschreitet Zeit und Kostenbudget so eklatant, dass man von einem Scheitern sprechen muss. Eine Softwareentwicklungsumgebung kann unter Umständen helfen, das Risiko eines solchen Scheiterns zu reduzieren.

4.3.3.2 Produktivität

Ein wesentlicher Grund für die Einführung neuer Entwicklungsumgebungen ist eine erhoffte Steigerung der Produktivität und damit einhergehend eine Senkung der Kosten der Entwicklung.

Diese Produktivitätssteigerung soll dadurch erzielt werden, dass Routineaufgaben abgekürzt werden können, das Debuggen erleichtert wird und, insbesondere bei verteilter Entwicklung, Koordinationsvorgänge effizienter gestaltet werden können.

In der Regel ist es so, dass die Anforderungen an die Prozesse innerhalb einer Organisation sich allmählich verändern, während die alten Vorgehensweisen und Werkzeuge beibehalten werden. Teilweise werden Änderungen vorgenommen, um auf erweiterte Anforderungen einzugehen, bis es sich nicht mehr vermeiden lässt, neue Systeme einzuführen. Nach einer Umgewöhnungsphase sollte dann eine erhöhte Produktivität oder Qualität festzustellen sein.

Die tatsächliche Produktivität von Softwareentwicklungsprozessen ist leider nur schwer messbar, da jedes Projekt seine eigenen Bedingungen hat und sich Kennzahlen schwer übertragen lassen.

4.3.3.3 Softwarequalität

Qualitätsmerkmale des Software Engineering wurden in Abschnitt 4.1.2 aufgeführt. Diese Kriterien lassen sich auf verschiedenste Weisen beeinflussen, die Wahl der geeigneten Werkzeuge ist nur ein Einflussfaktor. Einige Kriterien lassen sich aber stärker durch eine verteilte Entwicklungsumgebung verbessern als andere.

Korrektheit Eine Erhöhung der Korrektheit lässt sich gut durch eine Entwicklungsumgebung erzielen, indem entsprechende Module zur Erstellung und zum automatischen Ausführen von Tests angeboten werden. In einer verteilten IDE kann auch ein Prozess im Netzwerk regelmäßige, automatisierte Tests ausführen, um Fehlerquellen aufzuspüren, die durch unbeabsichtigte Seiteneffekte auftreten.

Performanz Das Tuning einer Anwendung auf Performanz ist oft ein aufwändiger Prozess, der von Werkzeugunterstützung stark profitieren kann. Probleme und Bottlenecks lassen sich durch Profiling-Werkzeuge aufspüren, Debugger erlauben das gezielte Nachverfolgen und Aufdecken der Ursachen.

In komplexeren, verteilten Systemen kann es auch hilfreich sein, wenn mehrere Personen beim Performance-Tuning zusammenarbeiten, um beispielsweise Probleme beim Zusammenspiel verschiedener Komponenten nachzuverfolgen. Hier können die Kooperationswerkzeuge einer verteilten Entwicklungsumgebung helfen.

Wartbarkeit Eine effektive Wartung setzt geeignete Werkzeuge und ein gutes Verständnis der zu wartenden Software voraus. Bei großen Projekten, die eine Lebensdauer von mehreren Jahren besitzen, kommt es häufig vor, dass sich die Verantwortlichkeiten innerhalb des Projektes wiederholt verschieben. Eine verteilte IDE kann hier helfen, indem durch Kooperationswerkzeuge der Wissenstransfer zu neuen Teammitgliedern unterstützt wird.

Pünktlichkeit Der pünktliche Abschluss eines Softwareprojektes ist in der Regel eher ein Problem des Projektmanagements. Eine integrierte Lösung, die alle Aspekte des Softwareentwicklungsprozesses umfasst, kann aber auch hierbei helfen, indem ein Überblick über die verschiedenen Teilaufgaben und deren Fortschritte ermöglicht wird. Verteilte Entwicklungsumgebungen umfassen zum Teil auch Funktionen zur Unterstützung des Projektmanagements oder Funktionen, die sich zumindest unterstützend verwenden lassen, wie etwa ein Bugtracking-System, dass es ermöglicht, sich einen Überblick über den aktuellen Stand der offenen Punkte zu verschaffen oder ein Workflow Management System zur Verwaltung der verschiedenen Aktivitäten im Projekt.

Qualitätskennzahlen Der erste Schritt zur Erhöhung der Qualität eines Softwareentwicklungsprozesses besteht oft darin, den Prozess transparenter und nachvollziehbarer zu gestalten. Für die CMM-Stufe zwei beispielsweise ist das Ziel, den Prozess wiederholbar zu gestalten. Hierfür ist eine Werkzeugunterstützung unabdingbar.

4.4 Agenten zur Prozessunterstützung

Der in dieser Arbeit gewählte Ansatz zur Unterstützung verteilter Softwareentwicklung besteht in der Erstellung eines spezialisierten Multiagentensystems. Die Benutzer des Systems sind Softwareentwickler und andere betroffene Parteien, die gemeinsam an einem verteilten Projekt arbeiten. Dies reicht von Kunden über Softwarearchitekten, Systemanalysten und Designer bis hin zu Testern.

4.4.1 Agenten

Die verteilte Softwareentwicklung ist ein Anwendungsfeld, das sich aufgrund seiner Eigenschaften besonders für eine agentenorientierte Unterstützung eignet. Die Anwender des Systems arbeiten weitgehend selbstständig an spezialisierten Aufgaben, interagieren aber miteinander, um ein gemeinsames Ziel zu verfolgen. Oft wechseln die Teilnehmer an einem Projekt mehrfach während der Lebensdauer eines Projektes, so dass flexible Strukturen bezüglich der Projektbeteiligten erforderlich sind.

Die Strukturierung eines solchen Systems als Multiagentensystem bietet die Möglichkeit, nach Bedarf neue Agenten und Agentenplattformen hinzuzufügen oder sie wieder aus dem System zu entfernen. Prozessbeteiligte werden innerhalb des Systems durch Software-Agenten repräsentiert, die miteinander interagieren können, um ihre Aufgaben zu bearbeiten. Zusätzliche Agenten im System unterstützen sie dabei, indem sie verschiedene Funktionen zur Verfügung stellen.

4.4.2 Werkzeugumgebung

Wie in Abschnitt 4.2 ausgeführt, ist die Verwendung der geeigneten Werkzeuge für die Softwareentwicklung und insbesondere für verteilte Softwareentwicklung ein wichtiger Erfolgsfaktor. Das bedeutet zum einen, die jeweils richtigen Werkzeuge für die Aufgabe zur Verfügung zu haben. Welche das sind, hängt von der Art der Aufgabe, der Qualifikation des Bearbeiters und dessen Präferenz ab. Zum anderen ist es wichtig, dass Arbeitsmaterialien problemlos zwischen den verschiedenen Benutzern ausgetauscht werden können, damit es nicht zu unnötigen Verzögerungen durch inkompatible Formate kommt. Schließlich ist eine weitere Gruppe von Werkzeugen zu betrachten, deren Zweck es ist, die Koordination der verschiedenen Akteure untereinander zu unterstützen, also die verschiedensten Arten von Kollaborationswerkzeugen.

4.4.3 Prozessunterstützung

Im Rahmen dieser Arbeit wird ein Ansatz verfolgt, bei dem stark auf die Verwendung von Prozessen und deren Unterstützung durch Workflows zurückge-

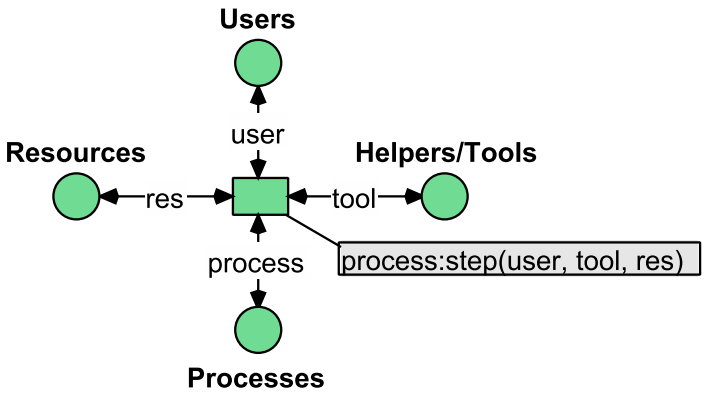

Abbildung 4.2: Abstraktes Modell der Unterstützungsumgebung

griffen wird. Grund hierfür ist die Notwendigkeit, sich insbesondere bei einem verteilten Projekt jederzeit einen Überblick über den aktuellen Arbeitsstand verschaffen zu können und die vorhandenen Ressourcen bestmöglich zu organisieren.

Es soll aber nicht versucht werden, die komplette Entwicklungsarbeit in einen rigiden Workflow zu sperren. Innerhalb der Vorgaben eines übergeordneten Prozesses muss es jedem Beteiligten möglich sein, seine Aufgaben flexibel auf die von ihm präferierte Weise auszuführen.

4.4.4 Modell

Abb. 4.2 stellt dieses Konzept auf einer sehr abstrakten Ebene dar. Im System befinden sich Benutzer, Hilfsmittel, Ressourcen und Prozesse, die miteinander interagieren, und zusammen eine Unterstützungsumgebung bilden. Ein Benutzer führt eine Aktion im Rahmen eines Prozesses aus. Dafür verwendet er geeignete dafür zur Verfügung stehende Ressourcen und Hilfsmittel. Der Prozess gibt die Reihenfolge der Aktivitäten vor, den oder die beteiligten Benutzer und die relevanten Helfer und Ressourcen.

Die Marken in den dargestellten Stellen sind wiederum als Netze zu verstehen, die über das Systemnetz synchronisiert werden. Ein Prozessschritt bedeutet dann jeweils einen Schaltvorgang in allen beteiligten Netzen.

Eine Aktion kann auch in Teilen von diesem Muster abweichen, etwa wenn ein Benutzer eine Aktion außerhalb eines Workflows ausführt oder ein Prozessschritt ohne Ressourcen oder auch ohne Benutzerinteraktion ausgeführt wird.

Ebenfall nicht dargestellt in der Abbildung sind die Modifikationsprozesse, durch die die Systemkonfiguration während der Laufzeit verändert werden kann. Neue Prozesse, Benutzer, Ressourcen und Helfer können jederzeit dem System hinzugefügt, verändert oder entfernt werden. Diese Dynamik wird

durch die Verwendung von Reservierungskanten angedeutet.

4.5 Zusammenfassung

In diesem Kapitel wurde das Feld der verteilten Softwareentwicklung als Anwendungsbeispiel für das POTATO-System vorgestellt. Ziel war es, den Anwendungsbereich zu definieren und festzustellen, welche Art von Unterstützung dafür erforderlich ist.

Software Engineering beschäftigt sich mit der Entwicklung großer Softwaresysteme, dafür wurden eine Reihe von Aktivitäten identifiziert, die verschiedenen Vorgehensmodellen gemein sind und generell in Entwicklungsprozessen in der ein oder anderen Form vorkommen. Diese Aktivitäten oder Teile davon muss eine Entwicklungsumgebung unterstützen.

Softwarequalität lässt sich anhand verschiedener Kriterien bestimmen, die sich teilweise durch eine geeignete Entwicklungsumgebung beeinflussen lassen. Ebenso lassen sich Prinzipien guten Software Engineerings durch eine Entwicklungsumgebung mehr oder weniger gut unterstützen.

Für verteilte Systeme und verteilte Entwicklungsprozesse ergeben sich weitere Herausforderungen und Vorgehensmodelle. Serviceorientierte Architekturen sind ein Ansatz, den Herausforderungen verteilter Systeme zu begegnen, der Gemeinsamkeiten mit dem agentenorientierten Ansatz aufweist.

Für verteilte Softwareprojekte wurden verschiedene organisationale Probleme benannt, sowie Ansätze, wie diesen durch geeignete Werkzeugunterstützung begegnet werden kann. Organisationale Lösungsansätze wurden dabei nicht betrachtet.

Softwareentwicklung wird durch Entwicklungsumgebungen unterstützt. Es wurden allgemeine Entwicklungsumgebungen, sowie existierende Ansätze der Unterstützung verteilter Entwicklung vorgestellt. Hierbei wurde festgestellt, dass die meisten IDEs eine Sammlung verschiedener Werkzeuge und Ressourcen darstellen, die bestimmte Aktivitäten der Softwareentwicklung ermöglichen, teilweise findet sich bei Umgebungen für die verteilte Entwicklung auch eine integrierte Prozessunterstützung. Die Aspekte der Werkzeug- und der Prozessunterstützung bilden die Grundbausteine des POTATO-Systems.

Zur Unterstützung verteilter Entwicklungsprozesse wurde hieraus ein erstes grobes Konzept für eine verteilte, agentenorientierte Entwicklungsumgebung aufgestellt. Im nächsten Kapitel wird diskutiert, auf welche Weise diese verteilte Entwicklungsgebung erstellt werden kann. Es werden entsprechende Anforderungen aufgestellt und das POTATO-System als ein Basisframework für eine solche Umgebung konzipiert.

Kapitel 5

Das POTATO-System

In diesem Kapitel werden zunächst die Anforderungen aufgenommen, die an eine verteilte Entwicklungsumgebung (engl. integrated development environment = IDE) gestellt werden. Die vollständige Entwicklung einer solchen Umgebung geht über den Rahmen dessen hinaus, was diese Arbeit leisten kann. Zunächst wird das POTATO Framework entworfen (Process-Oriented Tool Agents for Team Organisation), das zur Entwicklung einer verteilten IDE verwendet werden kann. Dabei wird zudem ein Schwerpunkt auf eine prozessorientierte Ausrichtung erfolgen. Es wird dann skizziert, wie aufbauend auf diesem Framework die weitere Entwicklung und Ausgestaltung erfolgen kann. Zugleich kann das Framework POTATO aber auch als Basis für andere Anwendungen dienen, die ähnlich gelagerte Anforderungen haben.

Im Folgenden werden zunächst die Anforderungen aufgestellt, die an das POTATO-System und eine darauf aufbauende verteilte IDE gestellt werden. Anschließend werden verschiedene Alternativen zur Architektur eines solchen Systems diskutiert. Die Architektur wird dann in Form eines Referenznetzsystems modelliert und die Zusammenhänge zwischen den verschiedenen Komponenten so deutlich gemacht. Schließlich wird ein Grobentwurf für das weitere Vorgehen aufgestellt, auf dem aufbauend dann in den folgenden Kapiteln das HERA- und PIA-System dargestellt werden.

5.1 Überblick

Die Werkzeugumgebung besteht aus einer Reihe von interagierenden Agenten, die teilweise durch menschliche Benutzer kontrolliert werden, teilweise autonom ihre Ziele verfolgen. Diese Agenten befinden sich in einem Multiagentensystem, das sich mittels eines Netzwerks über beliebig viele verteilte Rechner erstrecken kann. Das System kann sich jederzeit umstrukturieren, neue Agenten können hinzukommen oder das System wieder verlassen, wenn neue Benutzer oder neue Funktionalitäten hinzugefügt werden etc.

Entscheidend für die Anwendung, die durch das Multiagentensystem imple-

mentiert wird, sind die Geschäftsprozesse, die damit bearbeitet werden können. Ein wichtiger Aspekt des Systementwurfs ist daher die Strukturierung der Prozesse innerhalb des Systems. Im Abschnitt 5.2 wird auf das Zusammenspiel zwischen Struktur und Prozessen innerhalb des Multiagentensystems eingegangen.

Anschließend werden in Abschnitt 5.3 die Anforderungen an die verteilte Softwareentwicklungsumgebung formuliert. Zu diesem Zweck wird zunächst eine Systemabgrenzung vorgenommen. Auch wenn die Implementierung der Softwareentwicklungsumgebung nicht Teil dieser Arbeit ist, werden Anforderungen daran aufgestellt. Diese werden dann auf die Ebene eines Basisframeworks und der grundlegenden Anforderungen an das zugrundeliegende Multiagentensystem heruntergebrochen. Mit der Vision vor Augen ist es möglich zu entscheiden, was für das POTATO-Framework erforderlich ist.

In Abschnitt 5.4 werden zunächst die Anforderungen betrachtet und Überlegungen angestellt, welche Architektur auf einer Überblicksebene gewählt werden soll und welche Strukturierung der Entwicklung für das letztliche Ziel der verteilten Entwicklungsumgebung zu wählen ist. Dafür werden verschiedene aufeinander aufbauende Stufen angeführt, die jeweils eine weitergehende Abstraktion und technische Integration darstellen. Es wird diskutiert, welche Ebene für das POTATO-System gewählt wird.

Dieses Modell wird dann zunächst in Abschnitt 5.5 in Form eines Referenznetzsystems konkretisiert. Darin werden die verschiedenen Bestandteile des POTATO-Systems als miteinander verbundene Referenznetze dargestellt, mit denen die tatsächlichen Abläufe simuliert und die Zusammenhänge deutlich gemacht werden können.

Anschließend wird näher auf die interne Struktur und Hauptstrukturierungsaspekte in der Architektur eingegangen. Dabei sind zwei Aspekte besonders zu beachten, die in den folgenden Kapiteln dann ausführlicher beleuchtet werden, nämlich die Verwendung von Agenten als Werkzeuge (Kapitel 6), sowie die Strukturierung von Abläufen zwischen verschiedenen Beteiligten durch eine agentenorientierte Prozessinfrastruktur (Kapitel 7).

5.2 Struktur und Prozess

Ein wichtiges Entwurfskriterium bei der strukturierten Softwareentwicklung besteht darin, Änderungen an entwickelten Systemen möglichst einfach zu ermöglichen. Dafür ist es von zentraler Bedeutung, die Teile eines Systems zu identifizieren, bei denen Änderungen erwartet werden. Diese Teile werden dann zu Konzepten abstrahiert und in der Architektur isoliert.

Die Abstraktion und die Antizipation von Änderungen ist ein zentrales Anliegen verschiedener Design Patterns und ist auch zentral bei der Erstellung von Rahmenwerken zur Softwareentwicklung. Das Strategiepattern etwa kapselt ein bestimmtes Vorgehen, das Factorypattern die Erstellung von Objekten

[Gamma et al., 1995, S. 24f]. Zwei Aspekte, auf die speziell in dieser Arbeit eingegangen wird, sind die Struktur eines Systems, also die Einheiten, aus denen das System aufgebaut ist, sowie die Prozesse, die innerhalb des Systems ablaufen.

5.2.1 Struktur

Die Konzentration auf die am System beteiligten Einheiten lässt sich unter anderem in agentenorientierten Entwicklungsansätzen beobachten. Hier können die beteiligten Agenten sich zur Laufzeit jederzeit neu formieren, zwischen verschiedenen Orten migrieren oder wieder aus dem System entfernt werden. Neue Agenten lassen sich so bei Bedarf sehr leicht zu einem bestehenden System hinzufügen. Durch die Verwendung geeigneter Schnittstellen, wie etwa FIPA-ACL, ist dies sogar unter Verwendung unterschiedlicher Technologien innerhalb eines Systems möglich.

In den verschiedenen Projekten zur agentenorientierten Softwareentwicklung [Bosch et al., 2002, Cabac, 2010] hat sich ergeben, dass auch eine agentenorientierte Strukturierung zum Teil nicht ausreichend ist, um komplexere Systeme zu strukturieren. Daher wird eine weitere Ebene der Strukturierung für Agentensysteme benötigt.

Eine Möglichkeit, weitere Struktur in ein Multiagentensystem zu bringen, ist die Einführung von Artefakten als passive Elemente [Omicini et al., 2008]. Ein weiterer Softwareentwicklungsansatz, der Gegenstände als Strukturierungsmerkmal verwendet, ist der Werkzeug und Material Ansatz (WAM) [Züllighoven, 2004].

Hier werden die Gegenstände in Werkzeuge (aktive Gegenstände) und Materialien (passive Gegenstände) unterteilt. Der WAM-Ansatz wird unter anderem verwendet, um benutzerzentrierte Anwendungen zu konstruieren, die spezialisierte Nutzer in ihrer Arbeit unterstützen sollen. Das hier verfolgte Anwendungsgebiet der verteilten Softwareentwicklung bietet sich dafür an.

Für die Strukturierung von Agentensystemen wird daher ein System von Helfer- und Ressourcenagenten, HeRA (HElper and Resource Agents) genannt, entwickelt. Durch die Kapselung bestimmter Funktionalitäten und Ressourcen in eigenen Agenten können diese Strukturelemente jederzeit zur Laufzeit umorganisiert werden.

5.2.2 Prozess

Ein weiterer wichtiger Aspekt, der Änderungen unterworfen sein kann, sind die Prozesse, die innerhalb eines Systems ablaufen. Durch die ständige Überprüfung und Bewertung von Geschäftsprozessen in Organisationen, auch von Softwareentwicklungsprozessen, sind Änderungen hier in der Vergangenheit immer häufiger geworden. Daher hat es sich als sinnvoll erwiesen, die Prozesse, die sich zwischen den verschiedenen Teilen einer Anwendung abspielen, als

eigenes Konzept zu isolieren und damit separat änderbar zu machen. Hierfür verwendet man Workflow Management Systeme (WfMS).

Eine wichtige Herausforderung besteht daher darin, die Prozesse zu identifizieren, die in einem System ablaufen und sie in der Software adequat abzubilden. Die hierzu verwendete Software muss die nötige Flexibilität liefern, um sich an die vorgefundenen Prozesse und spezifischen Bedürfnisse anzupassen.

Wenn Geschäftsprozesse durch ein WfMS oder ähnliche Mittel gesteuert werden, müssen immer auch die sozialen und organisationalen Auswirkungen beachtet werden (Vgl. z.B. [Wieland et al., 2009, Gross und Pekkola, 2008]). Es besteht die Sorge, dass durch ein solches System die Autonomie der Mitarbeiter eingeschränkt und eine zu große Kontrolle ausgeübt wird. Dies ist nicht im Sinne der Mitarbeiter, oft ist es aber auch nicht im Sinne der Organisation, da die Auswirkungen einer solchen zu strengen Regulierung der Verlust der individuellen Kreativität und damit der Leistungsfähigkeit der Mitarbeiter sein kann.

Bei der Implementierung von Prozesssteuerungssystemen muss daher darauf geachtet werden, dass diese eine ausreichende Flexibilität im System erlauben. Ein WfMS kann positiv wirken, wenn es Menschen in ihrer Arbeit unterstützt, anstatt zu versuchen, sie lediglich als Ressourcen zu steuern, indem es den menschlichen Prozessteilnehmern Individualität und die Möglichkeit zur Anpassung der Prozesse an die spezifischen Erfordernisse lässt.

Es ist nicht Thema dieser Arbeit, die sozialen Aspekte von Informations- und Prozesssteuerungssystemen erschöpfend zu erörtern. Dennoch wird beim Entwurf der Aspekt der menschenzentrierten Sichtweise beachtet und der mündige Anwender, der nicht gesteuert, sondern in seiner qualifizierten Arbeit unterstützt werden muss, im Mittelpunkt gesehen. Dies wird nicht als Widerspruch zur Verwendung von WfMS gesehen, da hier gerade das Ziel besteht, das WfMS mit ausreichender Flexibilität auszustatten und die Gesamtsysteme so zu gestalten, dass sie dies berücksichtigen. Auch aus diesem Grund werden Agentensysteme für die Implementierung verwendet, da in diesen viele Konzepte zur Anpassung eines Systems durch autonome, interagierende Teilnehmer vorhanden sind.

5.2.3 Integration

Die Strukturierung von Multiagentensystemen durch Helferagenten wurde in [Lehmann und Markwardt, 2004, Lehmann et al., 2005, Willmott et al., 2005, Markwardt und Moldt, 2010] entwickelt. Parallel wurde die agentenorientierte Prozessinfrastruktur in [Reese et al., 2005, Reese et al., 2006a, Reese et al., 2006c, Markwardt et al., 2008a] vorangetrieben. In [Reese, 2010] werden Ansätze zur Integration einer prozessorientierten und einer agentenorientierten Systemstrukturierung diskutiert. Darin wird eine mehrstufige Integration von Prozess- und Strukturorientierung vorgeschlagen,

die schließlich in einer Architektur mündet, in der die beiden Konzepte gleichzeitig und gemeinsam verwendet werden können.

Dort wird allerdings nicht näher darauf eingegangen, auf welche Art die Strukturierung des Agentensystems vorgenommen wird. Dies wird in dieser Arbeit mit dem Konzept der Helfer- und Ressourcenagenten beantwortet. Entsprechend wird ein Architekturmodell analog zu dem aus [Reese, 2010] aufgestellt, das die Aspekte der Helferagenten mit der Prozessinfrastruktur vereint. Dies wird ausführlich in Abschnitt 5.4 behandelt.

5.3 Anforderungen

In diesem Abschnitt werden die Anforderungen für die verteilte Softwareentwicklungsumgebung und das zugrundeliegende Potato-System entwickelt. In den weiteren Kapiteln wird aus diesen Anforderungen ein Konzept zur Implementierung erarbeitet, von dem dann einige Teile, die Potato-Plattform sowie Beispiel-Agenten und -Workflows, prototypisch implementiert werden.

Die Entwicklungsumgebung soll in der Lage sein, prinzipiell jede Art von verteilter Softwareentwicklung zu unterstützen, sei es objektorientiert, funktional oder agentenorientiert, unabhängig von der verwendeten Methodik. Dafür muss sie zu jeder Art von Entwicklungsvorgehen entsprechend konfiguriert werden. Diese Konfiguration umfasst die Festlegung der verwendeten Prozessmuster, sowie der Hilfsmittel und Ressourcen, die jeweils erforderlich sind.

Da es nicht möglich ist, jedes mögliche Vorgehen zu modellieren, wird hier der Schwerpunkt auf die Unterstützung des Paose Ansatzes gelegt, mit dem Potato auch selbst entwickelt wird. Dadurch können gleichzeitig die hierbei verwendeten Konzepte und Vorgehensmodelle deutlich hervorgehoben werden.

Grundlage hierfür ist die Potato-Plattform, die die Entwicklung von Prozessen und Werkzeugen für verteilte Unterstützungssysteme bereitstellt. Potato verwaltet Helfer, Ressourcen und Workflows und stellt diese über Benutzeragenten Anwendern des Systems zur Verfügung. Die Anpassung an einen konkreten Anwendungskontext (wie z.B. Softwareentwicklung) erfolgt dann über die Definition dieser Entitäten.

Im Folgenden wird zunächst der Anwendungskontext umrissen, um deutlich zu machen, was die externen Anforderungen an das System sind. Es wird deutlich gemacht, welche Arten von Benutzern in welchen Konstellationen durch das System unterstützt werden sollen. Anschließend werden Anforderungen an die Architekur des Systems aufgestellt und die agentenorientierte Strukturierung von Potato motiviert. Es folgen die Anwendungsfälle des Systems sowie schließlich eine Aufstellung nicht-funktionaler Anforderungen.

5.3.1 Anwendungskontext

Die verteilte Entwicklungsumgebung hat die Aufgabe, Softwareentwicklung in verteilten, heterogenen Teams zu unterstützen. Die Benutzer besitzen unterschiedliche Qualifikationen, sind aber in der Regel Experten in ihrem jeweiligen Gebiet. Koordination der verschiedenen Aktivitäten im Gesamtkontext der zu bearbeitenden Aufgaben und Bereitstellung der Mittel zur Bearbeitung sind die Hauptanliegen der Software.

Daraus ergeben sich eine Reihe von Anforderungen an das zu erstellende Softwaresystem. Der jeweilige Arbeitsplatz der verschiedenen Benutzer muss es ihnen erlauben, die anfallenden Tätigkeiten effektiv und effizient auszuführen. Da diese Tätigkeiten sehr heterogen sind, müssen die verschiedenen Arbeitsplätze sehr unterschiedlich gestaltet sein. Machbar ist das durch eine flexible Gestaltbarkeit und Anpassbarkeit von Seiten des Benutzers.

Es kann davon ausgegangen werden, dass die Benutzer Experten in ihrem jeweiligen Anwendungsgebiet sind und daher entsprechend dem Leitbild des Arbeitsplatzes für selbstverantwortliches Expertenhandeln [Züllighoven, 2004] in diesen Tätigkeiten unterstützt, aber nicht gesteuert werden müssen. Eine Werkzeugumgebung, in der jeder Benutzer seine eigenen Werkzeuge auswählen und individuell konfigurieren kann, bietet sich hierfür an. Es ist allerdings auch zu berücksichtigen, dass bereits vorhandene, zufriedenstellend funktionierende Werkzeuge nicht ersetzt werden müssen, sondern unter Umständen lediglich integriert werden.

Arbeitsmittel und Ressourcen sollen den Benutzern jederzeit auf ihrem Arbeitsplatz verfügbar gemacht werden. Für die verteilte Zusammenarbeit ist es darüber hinaus erforderlich, diese zwischen Benutzern austauschen zu können.

Der andere Aspekt, der zu berücksichtigen ist, ist die Zusammenarbeit der verschiedenen Benutzer im Kontext einer Gesamtaufgabe. Architektonisch ergibt sich hieraus zunächst, dass das System als eine verteilte Software zu erstellen ist, bei der jeder Benutzer seinen eigenen Arbeitsplatz besitzt und die einzelnen Arbeitsplätze über eine Infrastruktur miteinander in Verbindung stehen.

Darüber hinaus müssen die einzelnen Teilaktivitäten einer Gesamtaufgabe miteinander verzahnt werden. Im Gegensatz zu den Einzelaktivitäten, bei denen es wichtig ist, den Akteuren in ihren Möglichkeiten eine freie Entfaltung zu ermöglichen, kann auf dieser Ebene ein höherer Steuerungsgrad wünschenswert sein. Erst dadurch wird eine effiziente Zusammenarbeit ermöglicht, die auf Projektleitungsebene gesteuert und überwacht werden kann. Aus diesem Grund soll die Softwareentwicklungsumgebung die Möglichkeit bereitstellen, die verschiedenen Teilaufgaben im Softwareentwicklungsprozess zu einem gemeinsamen Prozess zusammenzufassen und diesen Prozess zu koordinieren.

5.3.2 Systemabgrenzung

Die verteilte Softwareentwicklungsumgebung ist ein System, das verschiedene Benutzer verbinden und in ihrer Arbeit unterstützen soll. Hier werden die Systemgrenzen abgesteckt und die externen Partner des Systems benannt, um klarer abzugrenzen, was Teil des Systems ist und was nicht.

5.3.2.1 Zweck des Systems

Zunächst ist zu unterscheiden zwischen der Verteilten Softwareentwicklungsumgebung und dem POTATO-System. Der Zweck der verteilten IDE ist die Unterstützung verteilter Softwareentwicklungsprozesse. Dies umfasst das Projektmanagement und die Projektorganisation, sowie die verschiedenen Aktivitäten zur Abstimmung und zur Implementierung der Software. Die verschiedenen Parteien, die hieran beteiligt sind, sollen durch das System effizienter kollaborieren können.

Das POTATO-System ist ein Framework, das als Basis für die verteilte IDE dient. Sein Zweck ist es, verteilte Anwendungen zu ermöglichen, die sich durch koordinierte Interaktion zwischen den beteiligten Benutzern und den Einsatz von Hilfsmitteln und Ressourcen zur Ausführung der anliegenden Aktivitäten auszeichnen. Dafür muss es die Möglichkeit zur Verfügung stellen, neue Workflows zu definieren, sowie die damit verbundenen Benutzer und Rollen. Es muss diese Workflows instanziieren und ausführen, sowie die anliegenden Aufgaben an die zuständigen Benutzer zuweisen. Zusätzlich muss es Möglichkeiten bereit stellen, neue Helfer und Ressourcen im System zu definieren. Diese werden von Anwendern des Systems verwendet, um Aufgaben innerhalb und außerhalb der Workflowprozesse zu bearbeiten.

5.3.2.2 Benutzer

Benutzer der verteilten IDE sind alle Personen, die am Prozess der Softwareerstellung und -wartung beteiligt sind. Wer das im Einzelnen ist, hängt natürlich von der jeweiligen Projektorganisation ab und davon, welche Art von Modulen verfügbar sind.

Die Benutzer können sich an unterschiedlichen Orten befinden, zu unterschiedlichen Organisationen gehören und über unterschiedliche Qualifikationen verfügen. Im Allgemeinen werden als Benutzer des Systems hauptsächlich folgende Personengruppen vorkommen:

- Projektmanager
- Modellierer/Designer
- Entwickler
- Tester

- Projektpartner/Kunden

Die Einbindung von Kundenvertretern in das System erhöht die Transparenz und verbessert die Übereinstimmung zwischen Kundenwünschen und tatsächlichem Projektergebnis. Die Einbeziehung von Vertretern anderer Organisationen ist aber auch mit einer ganzen Reihe von Herausforderungen verbunden.

Benutzer greifen auf das System über ein Clientprogramm zu, das sich mit einer Agentenplattform verbindet, um dem Benutzer Zugriff auf das System zu ermöglichen. Durch mehrere verbundene Agentenplattformen, die auf verschiedenen Rechnern innerhalb eines (potentiell global verteilten) Netzwerks befinden, wird so ein Multiagentensystem geformt, an dem verschiedene Benutzer teilnehmen können.

5.3.2.3 Technisches Umfeld

Auf der technischen Seite ist die Entwicklungsumgebung einerseits eingebettet in die eigene Laufzeitumgebung, andererseits bestehen Verbindungen zu anderen Systemen, die für die Entwicklung genutzt werden.

Die Entwicklungsumgebung ist auf jedem Knotenrechner eingebettet in eine RENEW Laufzeitumgebung. Die Benutzungsoberflächen der Anwender können beispielsweise als Eclipse-Plugin in die Einzelplatzentwicklungsumgebung eingebettet sein. Desweiteren kann das System mit Dateien im Dateisystem, Datenbanken oder über spezialisierte Agenten mit anderen Programmen, wie etwa Versionsverwaltungssystemen in Verbindung stehen.

Eher lose ist die Verbindung zu bereits bestehenden Arbeitsplatzsystemen. Dies können IDEs sein, oder Office-Programme, mit denen Dokumente bearbeitet werden, die für die Prozesse von Bedeutung sind, die über das System abgewickelt werden. Diese Applikationen werden entweder außerhalb des Systems verwendet und Arbeitsergebnisse manuell in das System überführt, oder sie werden über spezialisierte Agenten gekapselt.

5.3.3 Architekturanforderungen

Das POTATO-System soll als verteilte Anwendung implementiert werden. Dies erfolgt in Form einer Multiagentenanwendung. Die verschiedenen Benutzer werden durch Agenten im System repräsentiert, ihre Interaktionen durch Protokolle und Workflows abgebildet.

Verschiedene Orte und Organisationen werden durch Agentenplattformen dargestellt, auf denen sich Agenten treffen können, um gemeinsam Aufgaben zu bearbeiten. Zu diesem Zweck können auch logische Plattformen gebildet werden, beispielsweise in Form eines Projektraumes, in dem sich Agenten zusammenfinden, die an einem gemeinsamen Projekt arbeiten.

Die Benutzer sollen in ihrer Arbeit durch Helfer unterstützt werden. Diese Helfer können in Form eines Werkzeuges einem Benutzer die Bearbeitung

von Materialien ermöglichen, sie können ihn aber auch auf andere Weise „intelligent“ in seiner Arbeit unterstützen. Dazu muss eine Plattform geschaffen werden, mit der es möglich ist, neue Helfer einfach zu erstellen und ins System einzubinden. Dadurch kann auf neue Erfordernisse eingegangen werden.

Die Arbeitsgegenstände, Ressourcen oder Materialien, werden ebenfalls explizit im System dargestellt und können von den Benutzern durch die Verwendung von Helfern erstellt, beobachtet und bearbeitet werden. Materialien sind dabei als vornehmlich passive Ressourcen und Dokumente zu verstehen, die durch Helfer bearbeitet werden, während Ressourcen auch den Zugriff auf Gegenstände außerhalb des Systems darstellen können, etwa einen Drucker.

Desweiteren soll es möglich sein, Workflowprozesse innerhalb des Systems zu definieren und ausführen zu lassen. Entsprechend berechtigte Benutzer können neue Prozessdefinitionen erstellen, wenn es im Projekt erforderlich ist, oder auch vordefinierte Prozesse instanziieren.

Durch Schachtelung und Verbindung verschiedener Workflows sollen die Prozesse auch auf verschiedenen Ebenen eingesetzt werden können, vom Haupt-Workflow eines Projektes bis zum Ad-hoc-Workflow für die Bearbeitung eines einzelnen Problems in der Software durch verschiedene Entwickler. Das System muss in der Lage sein, verschiedene Organisationen an der Bearbeitung eines Workflows zu beteiligen. Dies kann in einem zentralen WfMS oder über interorganisational implementierte Workflows (Vgl. [Reese et al., 2006c]) realisiert werden.

5.3.4 Use-Cases

In diesem Abschnitt werden Anwendungsfälle für die verteilte Softwareentwicklungsumgebung angegeben. Diese werden in zwei Gruppen unterteilt. Zunächst werden die fachlichen Anwendungsfälle erörtert, die den zu unterstützenden Softwareentwicklungsprozess betreffen. Hierdurch wird deutlich, welche Art von Funktionalität eine verteilte Softwareentwicklungsumgebung erfordert, wie sie mit Potato entwickelt werden könnte.

Anschließend wird die Framework-Ebene betrachtet. Auf dieser Ebene werden die Funktionen dargestellt, die das Potato-System für die Entwicklung allgemeiner verteilter Anwendungen bereitstellt.

Da das System auf einem generischen Multiagentensystem aufsetzt, sind natürlich auch die grundlegenden Funktionen eines MAS zugreifbar. Dies umfasst Funktionen wie das Definieren, Erzeugen, Migrieren und Beenden von Agenten und die Verwaltung von Agentenplattformen, Nachrichtenaustausch zwischen Agenten etc. Für diese Funktionalität wird auf die vorhandene Capa Implementation zurückgegriffen. Die entsprechenden Use-Cases werden daher nicht weiter ausgeführt.

5.3.4.1 Anwendungsfälle auf fachlicher Ebene

Auf der fachlichen Ebene ergeben sich die konkreten Anwendungsfälle aus der verwendeten Softwareentwicklungsmethodik (siehe auch Abschnitt 4.1). In den meisten Methodiken finden sich die folgenden Aufgaben wieder, die im System unterstützt werden müssen. Die genaue Ausgestaltung bezüglich verwendeter Prozesse, Artefakte und Hilfsmittel ist dann abhängig vom jeweiligen Vorgehensmodell. Abb. 5.1 zeigt die verschiedenen Aktivitäten des Entwicklungsprozesses, die hier betrachtet werden, sowie die Akteure, die daran beteiligt sind:

- Projektplanung und -steuerung
- Erfassung von Anforderungen
- Konzeption (Grobentwurf/Feinentwurf)
- Implementierung
- Test und Fehlerbehebung
- Auslieferung
- Wartung

Generell gilt, dass mehrere der hier angegebenen Aufgaben in einer Person vereint sein können. Auf der anderen Seite kann aber auch eine Aufgabe von mehreren Personen ausgeführt werden. Teilweise handelt es sich um komplexe interorganisationale Prozesse. Im Fall der verteilten Entwicklung befinden sich typischerweise die Personen teilweise an unterschiedlichen Orten. Selbst eine Person kann sich zu unterschiedlichen Zeitpunkten an verschiedenen Orten befinden, was bei der Gestaltung der Interaktionen beachtet werden muss.

Projektplanung und -steuerung Die erfolgreiche Durchführung eines Softwareprojekts erfordert ein großes Maß an Planung und Steuerung während des gesamten Projektlebenszyklus. Ein Projektmanager muss die Möglichkeiten erhalten, ein Projekt zu strukturieren und die anfallenden Aufgaben auf die beteiligten Personen und Gruppen zu verteilen. Er muss Zugriff auf die aktuellen Projektdaten haben, sowie Werkzeuge zur effektiven Kommunikation mit den Projektteilnehmern.

Es muss eine Prozessunterstützung geschaffen werden, die die anfallenden Aktivitäten des Softwareentwicklungsprozesses unterstützt. Zur Strukturierung von Projekten und zur Kontrolle des Arbeitsfortschrittes haben sich unter anderem Workflow Management Systeme als zweckmäßig erwiesen. Darin muss die Projektleitung den Prozess für das Projekt konfigurieren und seinen Fortschritt kontrollieren können. Für Teilprojekte können Unterworkflows angelegt

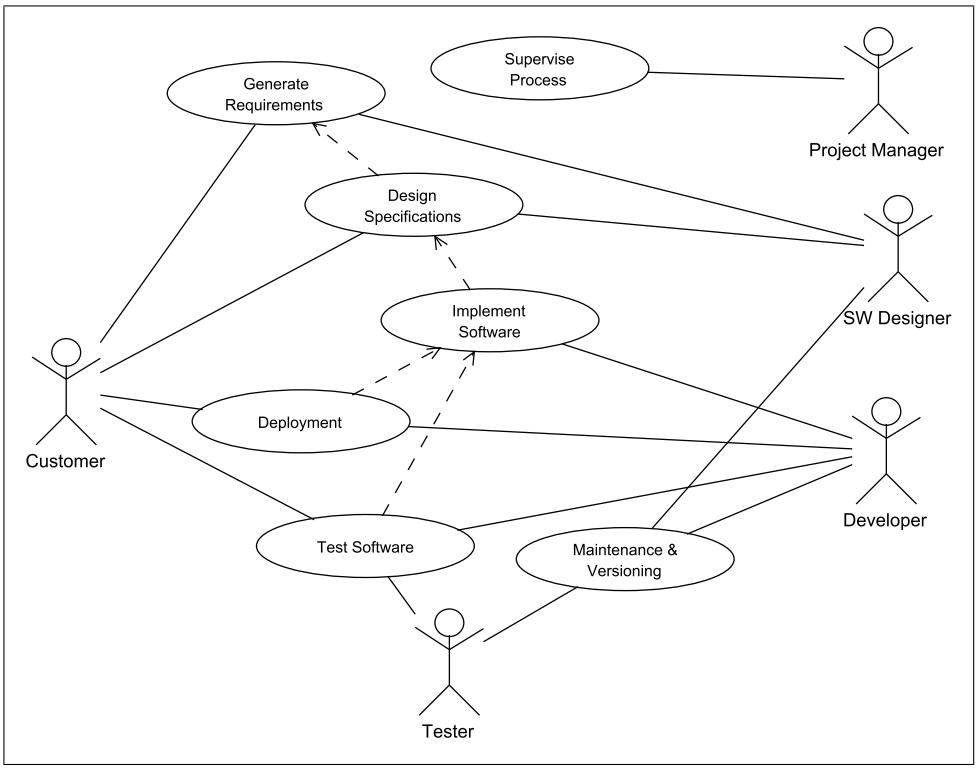

Abbildung 5.1: Fachliche Anwendungsfälle

werden, die im Sinne einer rollierenden Planung (vgl. [Müller-Lindenberg, 2005, S. 349]) die weitere Umsetzung des Projektes Schritt für Schritt konkretisieren.

Weitere Werkzeuge für das Projektmanagement erlauben beispielsweise eine Untersuchung der aktuellen Projektdaten, Durchführung virtueller Meetings oder die Verwaltung der zu einem Projekt gehörenden Dokumente. Auch die Funktionen zum Bug Tracking und Change Management liefern wichtige Informationen für das Projektmanagement.

Erfassung von Anforderungen Die Erfassung der Anforderungen ist ein wichtiger Schritt am Anfang eines Softwareprojektes. Der Auftraggeber muss hier möglichst genau spezifizieren, was das zu erstellende System leisten soll. Daher ist es zwar möglich, Anforderungen auf Seiten des Auftraggebers zusammenzustellen; eine endgültige Abstimmung und Beratung innerhalb der Anwenderorganisation und mit dem beaufragten Dienstleister, der die Implementierung vornehmen wird, ist aber außer in trivialen Fällen fast immer angeraten. Mit dem Thema der verteilten Anforderungsermittlung befasst sich beispielsweise [Gumm, 2009].

Hierfür muss die verteilte Entwicklungsumgebung Mittel bieten, Anforderungen zu erfassen und sich mit anderen Teilnehmern darüber zu beraten. Ein Beispiel für ein System zur verteilten Abstimmung in virtuellen Meetings ist

das D-Meeting System [Boulila et al., 2003].

Im konkreten Fall muss dann entschieden werden, wie weit diese Abstimmungsprozesse softwareunterstützt oder durch Face-to-face Meetings abgehandelt werden sollen. Oft bietet es sich selbst bei international verteilten Projekten an, zumindest am Anfang des Projektes ein Kickoff-Meeting zu veranstalten, um eine gewisse Vertrautheit der verschiedenen Projektteilnehmer herzustellen. Mangelnde Vertrautheit und dadurch mangelndes Vertrauen kann zu einem großen Problem in verteilten Projekten werden, vgl. z.B. [Kiel und Eng, 2003].

Für weite Teile der Anforderungsermittlung, wenn die grundlegenden Dinge geklärt sind, kann aber gut eine nicht persönliche, entfernte Kommunikationsart verwendet werden. Der Grad des persönlichen Kontakts reicht dabei von Wikis zur Erfassung von Anforderungen über E-Mail, Chat und Telefon bis hin zu Videokonferenzen, die nah an Face-to-face Meetings heranreichen.

Diese grundlegenden Kommunikationsmedien in der Entwicklungsumgebung zu replizieren ist in den meisten Fällen nicht sinnvoll. Spezifische Werkzeuge für die Anforderungsaufstellung können allerdings durchaus hilfreich sein. Beispiele hierfür sind Werkzeuge zum Erstellen von Diagrammen für bestimmte Zusammenhänge, Versionsverwaltung für Anforderungsdokumente, kollaborative Editoren etc.

Die Prozesse, die für die Anforderungsermittlung verwendet werden, sind natürlich sehr individuell für die jeweilige Organisation und können beliebig komplex sein. Da die Anforderungsermittlung Beteiligte aus verschiedenen Organisationen einbindet, sind hier unter Umständen organisationsübergreifende Prozesse erforderlich.

Im Paose-Ansatz werden die Anforderungen als Auflistung der verschiedenen Systemkomponenten und Interaktionen aufgenommen. Hieraus lässt sich ein Grobentwurfsdiagramm (Coarse Design Diagram) erstellen, das einem UML-Use-Case-Diagramm ähnelt [Cabac und Markwardt, 2009].

Konzeption Die Konzeption hat zum Ziel, aus den Anforderungen einen Entwurf für die Implementierung zu erzeugen. Hierzu werden die Anforderungsdokumente als Ausgangsbasis genommen, um konkrete Modelle zu erzeugen, die später bei der Implementation verwendet werden können.

Im Gegensatz zur Anforderungsanalyse liegt bei der Konzeption der geplanten Anwendung der größere Teil der Aufgaben beim beauftragten Dienstleister, der später auch die Implementation übernimmt. Er kennt die technischen Rahmenbedingungen meist besser als der Anwender und muss auf dieser Basis die Anforderungen in ein Konzept für die zu erstellende Software umsetzen. Auch hier sind Abstimmungen erforderlich, diese erfolgen aber auf einer technischeren Ebene und erfolgen unter Umständen mit anderen Personengruppen.

Die Unterstützung der Konzeption seitens der Entwicklungsumgebung gestaltet sich ähnlich wie die der Anforderungsermittlung. Auch hier müssen

strukturierte Dokumente erstellt werden, diesmal auf Basis der vorher erstellten Anforderungen. Da die Konzeptionsdokumente in der Regel formaler sind, können hier teilweise auch Verifikationsverfahren angewendet werden, um die Korrektheit des Entwurfs zu prüfen.

Kommunikation mit dem Auftraggeber ist auch hier erforderlich, meist aber in geringerem Umfang als bei der Anforderungsermittlung. Im Falle einer verteilten Konzeption müssen die beteiligten Teildesigner sich miteinander koordinieren, um sicherzustellen dass die Teilkonzepte kompatibel sind. Die erforderlichen Werkzeuge hierfür ähneln denen zur Anforderungsermittlung, allerdings ist die Konzeption konkreter und lösungsorientierter, so dass hier exaktere Modelle erforderlich sind. Die Weiterverwendung der Ergebnisse aus der Anforderungsermittlung zur Gestaltung des Softwareentwurfs soll durch die Verwendung gemeinsamer Materialien gewährleistet werden.

Im PAOSE-Ansatz beispielsweise werden die Interaktionen aus dem Grobentwurfsdiagramm überführt in Agenteninteraktionsprotokolle (AIP), die die Interaktionen zwischen den verschiedenen beteiligten Rollen ausführlicher darstellen. Ebenso kann das Rollendiagramm und teilweise auch die Ontologie aus dem Grobentwurf heraus entwickelt werden.

Implementierung Die genauen Anforderungen zur Unterstützung der Implementierungsaktivitäten sind stark von der verwendeten Technologie abhängig. Daher wird hier nur auf solche Anwendungen eingegangen, die im PAOSE Ansatz bei der Entwicklung mit RENEW und CAPA von Interesse sind.

Bei dieser Art der Entwicklung von Multiagentenanwendungen müssen vor allem die verschiedenen Agenten(-rollen) sowie ihre Interaktionen und Ontologien erstellt, bearbeitet und getestet werden können. Hierfür muss das System daher passende Werkzeuge zur Verfügung stellen. Viele verschiedene Werkzeuge existieren bereits in Form von RENEW-Plugins und -Ergänzungen. Der komplette PAOSE-Prozess wird durch verschiedene Modellierungswerkzeuge unterstützt, mit denen die Entwurfsartefakte erstellt, ausgeführt und debugged werden können.

Die Modelle der Entwurfsphase werden umgesetzt in ausführbare Modelle. Aus AIP-Diagrammen beispielsweise lassen sich die Skelette von Agentenprotokollen erzeugen, die dann angepasst und konkretisiert werden. Aus der Ontologiebeschreibung werden Javaklassen generiert, die in den Protokollen und Wissensbasen verwendet werden können. Die hierfür bereits vorhandenen Werkzeuge, die ausführlich in [CABAC, 2010] beschrieben sind, müssen in die Anwendung integriert werden.

Insbesondere im Bereich der Implementierung ist es wichtig, die richtige Balance zwischen der Steuerung der erforderlichen Entwicklungsprozesse und der kreativen Freiheit des einzelnen Entwicklers zu finden. Ansonsten besteht die Gefahr, dass Entwickler sich zu stark eingeengt fühlen, was die Produktivität beeinträchtigen und zu Akzeptanzproblemen führen kann.

Test und Fehlerbehebung Eine Möglichkeit, Software zu verifizieren, besteht darin, gewünschte Eigenschaften im Vorhinein über Korrektheitsbeweise der Spezifikation zu überprüfen. Durch Tests muss dann nur noch sichergestellt werden, dass die Software die Spezifikation erfüllt. Korrektheitsbeweise können allerdings Tests nicht ersetzen.

Zum Einen sind automatisierte Tests sinnvoll, um sicherzustellen, dass Softwarekomponenten sich entsprechend ihrer Spezifikation verhalten. Zum Anderen lässt sich dadurch feststellen, dass durch ungeplante Seiteneffekte eine zuvor korrekt arbeitende Komponente nicht mehr funktioniert. Dies hängt in der Regel damit zusammen, dass die Spezifikationen nicht korrekt umgesetzt oder nicht ausreichend genau spezifiziert worden sind.

Manuelle Tests sind darüber hinaus erforderlich, um Fehlersituationen aufzudecken, die bei der Formulierung der Tests übersehen wurden, vor allem aber, um Fehler aufzudecken, die nicht durch die Spezifikation erfasst werden. Dabei kann es sich um Lücken in der Spezifikation handeln, um Fehler im Entwurf oder auch Lücken in der Anforderungsermittlung. Oft werden Fehler, die automatische Tests finden könnten, erst durch manuelle Tests aufgedeckt, weil keine ausreichende Testabdeckung der Software vorliegt.

Für das manuelle Testen ist es erforderlich, lauffähige Versionen der produzierten Software, zum Beispiel in Form von Milestone Builds, zu erstellen. Automatisierte Tests können regelmäßig, zum Beispiel bei jedem Checkin oder einmal am Tag, auf Basis der aktuellen Codeversion durchgeführt werden. Die gefundenen Probleme müssen dann dokumentiert und an die zuständigen Entwickler kommuniziert werden.

Die Werkzeugunterstützung muss dem Tester also die zu testenden Softwarebestandteile zur Verfügung stellen, sowie die Konzeptionsdokumente, an denen diese bemessen werden. Das einfache Aufstellen von Testfällen, auch solchen, die im normalen Betrieb der Software selten vorkommen, ist wichtig für die Sicherstellung der Robustheit der Software. Angemessene Werkzeuge zur Dokumentation der Tests und zur Kommunikation der Ergebnisse an die Entwickler zur Behebung oder Klärung sind ebenfalls wichtig für die Testphase.

Zusätzlich können regelmäßige Builds in Verbindung mit automatisierten Tests helfen, unbeabsichtigte Seiteneffekte zu finden und eine hohe Softwarequalität zu erhalten. Hierfür müssen Möglichkeiten zur Konfiguration vorgesehen werden. Das System führt sie dann zu vorgesehenen Zeitpunkten aus und informiert über aufgetretene Probleme. Dies ist ein gutes Beispiel für einen Prozess, der autonom im System läuft und proaktiv eine Aufgabe ausführt. Eine entsprechende Komponente kann dann, wenn ein Fehler gefunden wird, eine Benachrichtigung aussenden oder einen vordefinierten Workflow starten.

Softwaretests sind ein Beispiel für einen Subprozess, der seinerseits neue Prozesse im System starten kann. Findet ein Tester (manuell oder automatisch) einen Fehler, startet das einen neuen Workflow für die Behebung dieses Fehlers, der je nach Gegebenheiten in der Organisation unterschiedlich ausfal-

len kann, z.B.: Der Fehler wird dokumentiert, kategorisiert und in eine Prioritätenliste eingereit. Daraus wird er zur Behebung an den Entwickler weitergeleitet. Anschließend wird er nachgetestet und schließlich als gelöst abgeschlossen. Prinzipiell können Probleme, die im Test gefunden werden, Rückgriffe zu allen früheren Entwicklungsphasen erforderlich machen, selbst zur Anforderungsermittlung, wenn sich herausstellt, dass bestimmte Anforderungen nicht sinnvoll umgesetzt werden können.

Auslieferung Wenn die Entwicklung abgeschlossen ist, muss die fertige Software auf dem Zielsystem ausgeliefert werden. Dafür muss ein geeignetes Installationspaket mit allen benötigten Programmteilen und, wenn erforderlich, Konfigurationseinstellungen zusammengestellt werden. Die Installation selbst, bei der unter Umständen noch weitere Einstellungen vorgenommen werden müssen, liegt dann außerhalb des Systems.

Dies ist im Prinzip ähnlich zu der Bereitstellung von Buildversionen für den Test. Allerdings kann für den Test unter Umständen noch ein bestimmter Testkontext angenommen werden, wie etwa eine Testdatenbank oder eine Sandbox, die noch nicht implementierte Funktionen oder auch Ausnahemefälle simuliert. Für die Auslieferung sind solche Funktionen natürlich nicht anzunehmen. Unter Umständen ist eine Parametrisierung erforderlich, wie etwa eine Seriennummer oder Parameter der Anwendungsumgebung. Im Grunde sollten aber alle diese Aspekte auch bereits im Test aufgetreten sein.

Wartung Die Wartung ist verwandt mit dem Bereich Testen und Fehlerbehebung, allerdings ist die Software zu diesem Zeitpunkt bereits ausgeliefert. Es muss daher ermöglicht werden, Konfigurationen in der Auslieferungsumgebung möglichst genau zu reproduzieren, um dort auftretende Fehler nachvollziehen zu können.

Auch für Erweiterungen um neue Funktionalität ist eine solche Umgebung wichtig. Hierfür gelten prinzipiell ähnliche Anforderungen, wie für die ursprüngliche Entwicklung des Systems, von der Anforderungsanalyse bis zur Auslieferung. Die Prozesse, die hier angewendet werden, unterscheiden sich zwar von denen zur Neuentwicklung, sind aber in den Grundzügen verwandt.

Ein später im Kapitel 8 vorgestellter Prototyp zeigt eine beispielhafte Implementierung eines Change Management Prozesses. Mit einem solchen Prozess, wenngleich üblicherweise komplexer, können typische Wartungssituationen bearbeitet werden.

5.3.4.2 Anwendungsfälle des POTATO-Systems

Auf dieser Ebene sind Anwendungsfälle zusammengefasst, die sich weniger aus konkreten fachlichen Anforderungen des Softwareentwicklungsprozesses ergeben. Stattdessen hängen sie mit der prozessorientierten Unterstützungsumge-

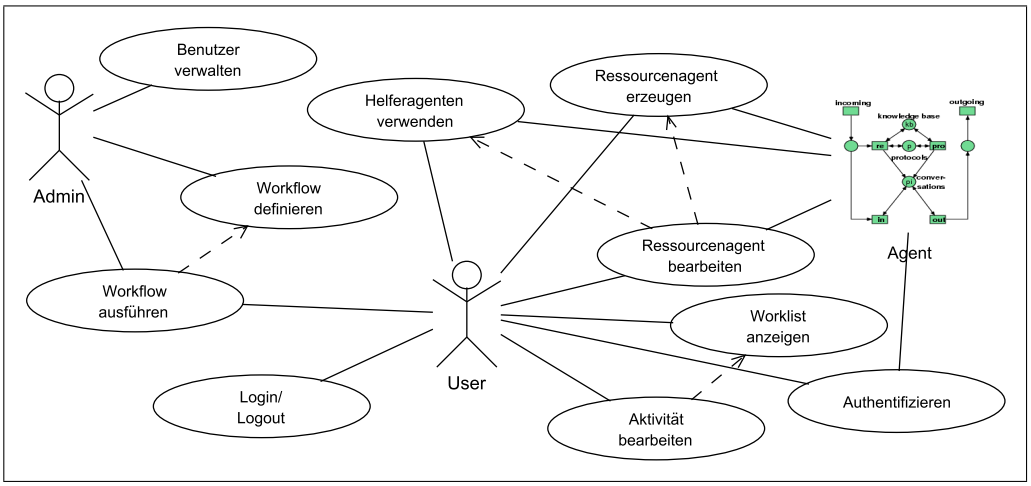

Abbildung 5.2: Anwendungsfälle der Werkzeugumgebung

bung zusammen, die die verwendete Methodik unterstützt und dementsprechend ausformuliert werden muss.

Abb. 5.2 zeigt eine Übersicht der verschiedenen Anwendungsfälle. Einige dieser Anwendungsfälle sind allgemeiner Natur, wie etwa die Benutzerverwaltung und die Anmeldung am System, einige beziehen sich auf die Verwendung von Helfer- und Ressourcenagenten, andere auf die Workflowunterstützung.

Benutzer verwalten Ein offenes, verteiltes System dieser Art kann nicht ohne eine Benutzerverwaltung auskommen. Es muss möglich sein, neue Benutzer im System einzurichten und ihnen Rollen und Rechte zuzuweisen. Dabei sollte ein Vorgehen gewählt werden, das verschiedenen beteiligten Organisationen Rechnung trägt, ohne dass sich ein Benutzer mehrfach anmelden muss.

Sicherheitsaspekte in Multiagentensystemen stellen allerdings ein extrem weitläufiges Thema mit vielfältigen Herausforderungen dar, so dass im Rahmen dieser Arbeit darauf nicht erschöpfend eingegangen werden kann. Diese Aspekte werden weitgehend ausgeklammert. Die Benutzerverwaltung dient hier vornehmlich dem Zweck, die Arbeitsplatzkonfiguration und die Aufgabenverteilung innerhalb der Prozessinfrastruktur zu ermitteln. Berechtigungskonzepte spielen nur eine untergeordnete Rolle. Die hier verwendete zentralisierte Benutzerverwaltung ist für einen größeren verteilten Kontext natürlich nicht ausreichend, hier müssten dann unter Umständen bestehende Authentifizierungssysteme wie etwa Kerberos [RFC4120, 2011] oder LDAP [RFC4511, 2011] integriert werden.

Login Der erste Schritt, wenn ein Benutzer das System benutzen will, ist die Anmeldung. Dabei werden die Rechte überprüft und der gespeicherte Arbeitsplatz des Benutzers wiederhergestellt. Die Agenten, die über den Verbindungsstatus eines Benutzers informiert sein müssen, werden benachrichtigt.

Dies ist zum Beispiel der Workitem Dispatcher, der die Worklists verwaltet. Dieser muss wissen, für welche Benutzer er Aufgabenlisten erstellen muss.

Logout Meldet sich ein Benutzer vom System ab, wird die Konfiguration seines Arbeitsplatzes gespeichert, um beim nächsten Login wieder abgerufen werden zu können. Wenn es erforderlich ist, wird anderen Agenten im System dann mitgeteilt, dass der Benutzer nicht mehr verfügbar ist, etwa dem Workitem Dispatcher.

Helfer und Ressourcen Für die verschiedenen Aktivitäten, die im Entwicklungsprozess ausgeführt werden, soll das System unterstützende Agenten bereitstellen. Diese werden in Form von Helfern, die dem Benutzer neue Funktionalität bereitstellen, und Ressourcen, die Materialien oder andere Ressourcen darstellen, angeboten. Unterschiedliche Benutzer im System benutzen für ihre Arbeit unterschiedliche Helfer und Ressourcen, daher muss die Bereitstellung flexibel anpassbar sein.

Eine zentrale Anforderung ist die Möglichkeit, verschiedene Helferagenten im System zu definieren und nach Bedarf an den einzelnen Arbeitsplätzen abzurufen. Die Helferagenten werden dann in eine Rahmenanwendung eingepasst und erweitern diese um ihre jeweilige Funktionalität.

Die Arbeitsgegenstände der Entwicklung, von Anforderungskatalogen über Designdokumente bis zu fertiger Software, werden als Materialien im System expliziert. Benutzer können Materialien erstellen, mit Hilfe von Helferagenten bearbeiten, an andere Benutzer versenden etc. Andere Ressouren, wie etwa Hardware (Drucker, Scanner), Datenbanken etc. werden als Ressourcenagenten ins System eingebracht.

Helferagenten auswählen Um neue Helferagenten in die Arbeitsumgebung eines Benutzers aufnehmen zu können, sind zwei Schritte erforderlich. Der Benutzer muss entscheiden welche Arten von Helferagenten er benötigt, anschließend müssen diese instanziiert, konfiguriert und dem Benutzer zugeordnet werden.

Dafür muss das System einen oder mehrere Agenten enthalten, die neue Helferagenten bereitstellen können. Auf Anfrage liefern diese Agenten eine Liste der verfügbaren Helferagenten zur Auswahl. Aus dieser Liste kann der Benutzer dann die gewünschten Helferagenten „bestellen“. Diese werden instanziiert und dem Benutzer verfügbar gemacht. Dazu fügen sie sich in die Benutzungsoberfläche des Benutzers ein und ergänzen diese so um die neue Funktionalität, die die Helfer anbieten.

Helferagenten benutzen Benutzer können Helferagenten für verschiedene Zwecke benutzen, um Materialien zu erzeugen und zu bearbeiten, um mit

anderen Benutzern zu kommunizieren oder zu anderen Zwecken. Die konkrete Ausgestaltung der Funktionalität ist abhängig vom jeweiligen Helfer.

Der Benutzer greift auf diese Funktionalität zu, indem er die vom Helferagenten bereitgestellte Benutzungsoberfläche verwendet. Diese kommuniziert mit dem Agenten im Multiagentensystem, so dass bestimmte Aktivitäten dann im Agenten ausgelöst werden, Änderungen am Zustand des Agenten vorgenommen, Nachrichten versendet oder Materialien verändert werden.

Ressourcenagent erzeugen Ressourcenagenten werden durch Helferagenten oder andere Agenten im System erzeugt. Handelt es sich um ein Material, kann der Agent mit einem Initialzustand erzeugt werden. Wird eine externe Ressource referenziert, muss der erzeugende Agent hierauf Zugriff haben. Nach der Erzeugung verfügt der Ressourcenagent über einen definierten internen Zustand und befindet sich an einem definierten Ort, z.B. auf dem Arbeitsplatz eines Benutzers.

Ressourcen verwenden Helferagenten dienen dazu, Ressourcen zu verwenden. Die genaue Art dieser Verwendung hängt von der Art der Ressource und des Helfers ab. Nicht jeder Helferagent kann jede Ressource verwenden, es kann eine n:m-Beziehung bestehen.

Wenn Helfer und Ressource miteinander kompatibel sind, können die entsprechenden Protokolle zur Verwendung benutzt werden. Der Benutzer verwendet die Benutzungsoberfläche des Helfers, um auf die Ressource zugreifen zu können.

Ressource verschieben Ressourcen können zwischen verschiedenen Orten ausgetauscht werden. Ein Benutzer kann eine Ressource an einen anderen Benutzer senden oder sie kann durch einen Prozess verschoben werden, beispielsweise im Rahmen der Bearbeitung eines Workflows. Anschließend kann auf die Ressource nur noch an ihrem neuen Ort zugegriffen werden.

Dabei kann zwischen der einfachen Zuordnung einer Ressource zu einem Agenten, also einem logischen Ort, und der tatsächlichen Verschiebung auf eine andere Agentenplattform, also einer Änderung des physikalischen Ortes unterschieden werden. Da Agenten über Nachrichten kommunizieren und diese auch zwischen verschiedenen Plattformen innerhalb eines Multiagentensystem ausgetauscht werden können, ist es nur in bestimmten Fällen erforderlich, den Ressourcenagenten selbst zu verschieben. In der Regel reicht es aus, die Zuordnung einer Ressource zu einem Benutzer, Helfer oder anderem Agenten zu ändern. Wenn ein Benutzer eine Ressource also an einen anderen Benutzer weitergibt, wird in der Regel nur der logische Ort geändert.

Prozesssteuerung Eine weitere zentrale Aufgabe der verteilten Softwareentwicklungsumgebung ist es, die verschiedenen Prozesse innerhalb des Ent-

wicklungsprozesses zu koordinieren. Daher muss es möglich sein, Workflowprozesse zu definieren, zu bearbeiten und ausführen zu lassen.

Workflow definieren Entsprechend berechtigte Benutzer können neue Workflowprozesse im System definieren und vorhandene Prozesse bearbeiten. Dies betrifft die Abfolge der Aktivitäten im Prozess ebenso wie die assoziierten Materialien, Helfer und Rollen.

Bei der Bearbeitung vorhandener Prozesse muss entschieden werden können, ob nur neu angelegte Prozessinstanzen die neue Definition verwenden oder ob existierende Prozessinstanzen auf die neue Definition umgestellt werden sollen und wie dies genau erfolgen soll. Ein Algorithmus hierfür wird in Abschnitt 7.4 vorgestellt.

Workflow ausführen Die definierten Prozesse werden im System instanziiert und ausgeführt. Dies kann direkt von einem Benutzer in Auftrag gegeben werden oder durch einen Agenten im System ausgelöst werden. So kann ein Helferagent zur Erfüllung seiner Aufgaben einen neuen Workflow starten oder ein Workflow startet einen neuen Unterworkflow.

Beim Start einer neuen Workflowinstanz können Parameter angegeben werden, die den Verlauf des Prozesses beeinflussen. Zum Beispiel kann ein Testagent einen neuen Bug-Handling-Prozess starten und eine Beschreibung des gefundenen Bugs als initialen Parameter an den Workflow übergeben. Daraus wird dann abgeleitet, welcher Prozess für diese Art von Fehler verwendet wird oder welche Benutzer für die Bearbeitung verantwortlich sind.

Worklist anzeigen Jedem Benutzer, der am Workflowmanagementsystem angemeldet ist, können Aktivitäten zur Bearbeitung zugewiesen werden. Diese werden je Benutzer in einer Worklist verwaltet. Der Benutzer hat die Möglichkeit, diese Worklist einzusehen, die anstehenden Aktivitäten zu betrachten und Aktivitäten zur Ausführung auszuwählen.

Bei der Anmeldung am System wird dem Benutzer die aktuelle Worklist zugesendet, bei jeder Änderung, die sich daran ergibt, erhält er eine Aktualisierung. So bleibt er stets auf dem Laufenden, welche Aufgaben zurzeit anstehen.

Aktivität annehmen Benutzer können Aufgaben aus ihrer Worklist zur Bearbeitung annehmen. Sie werden dann aus der Worklist entfernt und dem Benutzer als Aktivität zugewiesen. Hat ein Benutzer ein Workitem als Aktivität zugewiesen bekommen, wird es aus seiner Worklist und aus der Worklist von allen anderen Benutzern, die diese Aufgabe unter Umständen angeboten bekommen haben, entfernt.

Die Aufgabe kann dann bearbeitet und abgeschlossen werden oder auch wieder abgebrochen werden. Wird sie abgebrochen, wird sie wieder in den betroffenen Worklists angeboten.

Aktivität bearbeiten Wenn ein Benutzer eine Aktivität im Workflow zur Bearbeitung auswählt, erhält er alle benötigten Kontextinformationen, um die Aufgabe bearbeiten zu können. Zusätzlich zu den erforderlichen Falldaten, die sich aus der Workflowinstanz ergeben, erhält der Benutzer auch die nötigen Mittel an die Hand, um die Bearbeitung vorzunehmen. Zu diesem Zweck wird ein Helferagent für die Aktivität erzeugt und an den Aufgabenkontext angepasst bzw. mit den Falldaten konfiguriert.

Der Benutzer verwendet dann den Aktivitätsagenten (sowie eventuelle andere Helferagenten), um die Aufgabe zu bearbeiten. Nach der Bearbeitung wird dieser wieder an das WfMS zurückgesendet, um die Änderungen im Workflow abzulegen.

Workflow überwachen Eine wichtige Funktion von Workflow Management Systemen ist die Überwachung der laufenden Prozesse. Es muss, mit der entsprechenden Berechtigung, möglich sein, sich einen Überblick über die derzeit laufenden Prozesse zu verschaffen, die Prozessdaten einzusehen und, wenn erforderlich, steuernd in die Prozesse einzugreifen.

Erweiterbarkeit Es muss möglich sein, jederzeit neue Funktionalitäten im System bereitzustellen. Dies umfasst sowohl die Definition neuer Prozesse als auch neuer Werkzeuge. Wenn sich die Anforderungen an das System ändern, zum Beispiel durch neue Organisationsstrukturen oder geänderte Prozesse, können geänderte neue oder geänderte Workflows jederzeit ins System eingebracht werden. Ebenso können neue Helfer und Ressourcen jederzeit neu zum System hinzugefügt werden.

Strukturierung des Systems Es muss möglich sein, dem System zur Laufzeit neue Agenten (Anwender und Systemagenten) hinzuzufügen, neue Plattformen zu definieren und die Agenten auf den Plattformen neu zu strukturieren. Nur so kann wechselnden Konstellationen in den beteiligten Organisationen Rechnung getragen werden.

Direkte Kommunikation Wie bereits bei den fachlichen Anforderungen erwähnt, ist es in einer Reihe von Fällen erforderlich, dass sich verschiedene Anwender des Systems direkt miteinander austauschen können. Dies sollte systemseitig unterstützt werden, durch Möglichkeiten zur Kommunikation und zum kooperativen Bearbeiten von Aufgaben.

Beispiele hierfür reichen von einfachen Chatmöglichkeiten bis zu kooperierenden Werkzeugen zum verteilten Programmieren und Debuggen oder sozialer Interaktion. Die Unterstützungsumgebung muss die Möglichkeiten bereitstellen, solche Werkzeuge zu erstellen und zu verwenden.

5.3.5 Nicht-funktionale Anforderungen

Zusätzlich zu den genannten funktionalen Anforderungen an das POTATO System ergeben sich weitere, nicht-funktionale Aspekte, die bei der Implementierung berücksichtigt werden müssen.

Die entwickelte Software soll leicht verständlich sein, so dass Weiterentwicklungen leicht möglich sind. Dies ist bei Projekten mit häufig wechselnden Entwicklern besonders wichtig und kann durch klare Strukturierung sowie gute und vollständige Dokumentation erreicht werden.

Ein weiteres Kriterium bei der Entwicklung von Anwendungssoftware ist die Bedienbarkeit durch den Anwender. Um dies zu gewährleisten, wird bei der Entwicklung der Benutzungsschnittstelle darauf geachtet, sich an bestehende Guidelines und Best Practices der Schnittstellengestaltung zu halten. Die Bedienbarkeit wird weiterhin dadurch erleichtert, dass die Benutzungsschnittstelle direkt in eine Eclipse-Umgebung integriert wird, so dass nach Möglichkeit kein Bruch in den Verwendungsmustern entsteht für Anwender, die mit dieser IDE vertraut sind.

Die Performanz ist an dieser Stelle kein Hauptkriterium, da es sich in diesem Stadium lediglich um einen Prototypen handeln kann. Die Implementierung mit Referenznetzen ist nicht die effizienteste, wird aber hier verwendet, um die Modellierungsmöglichkeiten zu haben, die Netze in Netzen bieten. Für eine Produktiventwicklung müssten hier andere Maßstäbe angelegt werden.

5.4 Architekturmodelle

In diesem Abschnitt werden zunächst verschiedene Möglichkeiten für Architekturen diskutiert, die für die Entwicklung einer verteilten Softwareentwicklungsumgebung verwendet werden können. Diese Stufen orientieren sich an [REESE, 2010, S. 114ff] und wurden parallel dazu entwickelt. Während Reese die genauere Ausgestaltung des Agentenmanagementsystems weitgehend offen lässt, werden in dieser Arbeit konkretere Strukturierungsvorschläge für agentenorientierte Helfer- und Ressourcensysteme vorgeschlagen.

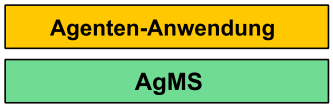

Abbildung 5.3: Architekturstufe I

5.4.1 Stufe I

In Stufe I wird die zu entwickelnde Anwendung direkt auf einer Basisarchitektur aufgebaut, in diesem Fall einem Multiagentensystem. In [REESE, 2010,

S. 114ff] wird parallel auf jeder Stufe als Alternative der Aufbau auf einem generischen Workflow Management System (WfMS) diskutiert. Diese Modelle wären zwar ebenfalls denkbar, werden aber für die hier diskutierten Zwecke nicht als zielführend erachtet und daher ausgeklammert.

Abb. 5.3 zeigt eine schematische Darstellung einer solchen Architektur. Das Agentenmanagementsystem wird hier als ein Block dargestellt, in der Realität kann natürlich auch dieses System aus verschiedenen Schichten bestehen. Bei dem Multiagentensystem CAPA wären noch Referenznetze (RENEW) und Java als untergeordnete Schichten implizit unter dem Agentenmanagementsystem zusammengefasst.

Auf diesem Agentenmanagementsystem wird dann direkt die gewünschte Anwendung (in diesem Fall die verteilte Softwareentwicklungsumgebung) konstruiert. Dieses Vorgehen wurde beispielsweise in den ersten Siedlerprojekten (vgl. [BOSCH et al., 2002]) gewählt.

Vorteil dieses Ansatzes ist, dass Anwendungen relativ schnell entwickelt werden können. Die Werkzeuge des PAOSE-Ansatzes unterstützen die Entwickler bei der Konstruktion der erforderlichen Ontologien, Agenten und Protokolle, so dass schnell eine lauffähige Anwendung entstehen kann.

Der Nachteil liegt darin, dass die Art, wie die Agenten zusammenarbeiten, in keiner Weise reguliert und vorstrukturiert ist. Strukturen und Interaktionsmuster entstehen adhoc im Laufe des Entwicklungsprozesses. Während das für kleinere Anwendungen durchaus noch überblickt werden kann, ist für komplexere Systeme mehr strukturelle Vorgabe erforderlich. Ebenso ist es zwar möglich, eine objektorientierte Anwendung direkt in Java zu implementieren, in der Regel wird man aber für komplexere Systeme auf ein Framework zurückgreifen.

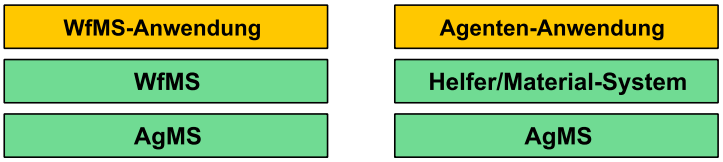

Abbildung 5.4: Architekturstufe II

5.4.2 Stufe II

Um der Anwendungsentwicklung mehr Strukturierung zu verleihen, wird auf Stufe II auf ein Framework zurückgegriffen, auf dem dann die Anwendung selbst aufsetzt. Die beiden Frameworks, die hier diskutiert werden, sind zum einen die agentenorientierte Prozessinfrastruktur, die eine WfMS-Strukturierung der Anwendung ermöglicht, zum anderen die Strukturierung anhand von Helfern und Ressourcen, die ein Vorgehen ähnlich dem WAM-Ansatz [ZÜLLIGHOVEN, 2004] unterstützt.

Abb. 5.4 verdeutlicht dieses Modell. Das verwendete Framework liegt als Abstraktionsschicht zwischen dem Agentensystem und der Anwendung und erlaubt damit, die Anwendung auf einer höheren Abstraktionsebene zu entwickeln.

Der Vorteil, der sich hier gegenüber Stufe I ergibt, ist die bessere Strukturierung, die zugleich auch ein bestimmtes Entwicklungsparadigma nahelegt. Das Framework kann eine Reihe von häufig verwendeten Funktionen bereits von sich aus bereitstellen, so dass man sich bei der Anwendungsentwicklung auf die fachliche Ausgestaltung konzentrieren kann.

Auf der anderen Seite ergeben sich aber auch Nachteile, dadurch dass die Systemstruktur bereits in weiten Teilen festgelegt ist. Aspekte der Anwendung, die nicht in das gewählte Paradigma passen, lassen sich nur schwer realisieren oder stehen im Widerspruch zur Architektur des Frameworks, so dass die Architektur logische Brüche aufweist. Die Entscheidung für ein Framework ist daher meist auch die Entscheidung gegen die anderen Frameworks.

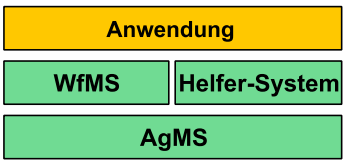

Abbildung 5.5: Architekturstufe III

5.4.3 Stufe III

Bei der Aufnahme der Anforderung für die verteilte Softwareentwicklungsumgebung hat sich ergeben, dass Prozessaspekte eine wichtige Rolle spielen, die über eine Prozessinfrastruktur implementiert werden sollen. Gleichzeitig sind aber an vielen Stellen auch Ressourcen und Helfer zu ihrer Bearbeitung ein wichtiger Aspekt. Diese Anforderung lässt sich auch in anderen Anwendungen wiederfinden.

In einer Architektur der Stufe III wird daher versucht, beiden Aspekten Rechnung zu tragen, indem die beiden in Stufe II erwähnten Frameworks parallel verwendet werden. Die Anwendung greift dann jeweils auf die Bestandteile zurück, die an der jeweiligen Stelle von Bedeutung sind. In Abb. 5.5 ist dargestellt, dass die beiden verwendeten Frameworks unabhängig nebeneinander auf dem Agentensystem aufsetzen und durch die Anwendung verwendet werden können. Überschneidungen zwischen den beiden Systemen sind nicht vorhanden.

Der Vorteil gegenüber Stufe II ist jetzt, dass mehr unterschiedliche Funktionen genutzt werden können. Auf der anderen Seite geht aber die Klarheit der Architektur verloren, da das System nicht mehr einem klaren Entwurfsprinzip

folgt. Darüber hinaus kann es zu Problemen mit der Integration unterschiedlicher Frameworks kommen. Teile der Software werden dann mit Workflows modelliert, andere Teile mit Helferagenten. Insbesondere an Stellen, an denen beide Aspekte benötigt werden, kommt es dann zu Brüchen in der Architektur, die die Anwendung jeweils im Einzelfall adressieren muss. Zusätzliche Funktionalität wird also durch mögliche Inkonsistenz in der Architektur erkauft.

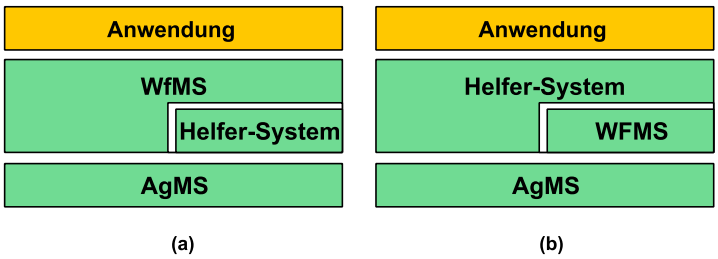

Abbildung 5.6: Architekturstufe IV

5.4.4 Stufe IV

Das Hauptproblem der Architekturstufe III ist die Inkonsistenz bei der Verwendung verschiedener Frameworks. In Stufe IV wird daher versucht, eine Integration vorzunehmen. Hierfür bieten sich zwei Möglichkeiten an, die in Abb. 5.6 dargestellt werden: Die Verwendung der Helferarchitektur als Basis für die Entwicklung der Prozessinfrastruktur (a) oder umgekehrt die Verwendung der Prozessinfrastruktur als Basistechnologie für die Helfer- und Ressourcenagenten (b). Wie aus der Abbildung ersichtlich ist, ist bei den beiden vorgestellten Varianten jeweils ein Framework „dominierend“, dessen Entwurfsprinzipien die Anwendungsentwicklungsschnittstelle an der Oberfläche bestimmt.

Das dominierende Framework baut dabei sowohl auf dem Agentensystem als auch auf dem zweiten verwendeten Framework auf. Die Anwendung verwendet lediglich die nach oben hin angebotene Schnittstelle des dominierenden Systems. Auf diese Weise können beide Aspekte berücksichtigt werden, während für die Anwendungsentwicklung nur eine einheitliche Schnittstelle berücksichtigt werden muss.

Der Vorteil gegenüber der Stufe III ist die Vereinheitlichung der Entwicklungsschnittstelle. Dadurch besteht für den Anwendungsentwickler nicht mehr das Problem, dass unterschiedliche Konzepte miteinander vermischt werden und der Entwurf dadurch inkonsistent wird. Hierin liegt aber gleichzeitig auch der Nachteil, da durch die asymmetrische Integration auch wieder Teile der Funktionalität verborgen werden. Der Funktionsumfang in Stufe IV ist geringer als der in Stufe III, da effektiv nur die Funktionen angeboten werden, wie sie in Stufe II bereits vorhanden waren, also Workflowfunktionalität oder Helfer- und Ressourcenfunktionalität.

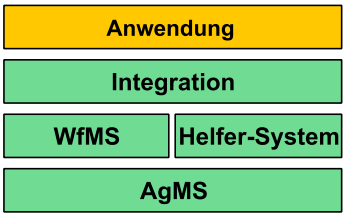

Abbildung 5.7: Architekturstufe V

5.4.5 Stufe V

Auf Stufe V schließlich wird die Integration der WfMS- und Helferkonzepte einheitlich vorgenommen. Abb. 5.7 zeigt das Helfersystem und das Workflow Management System parallel gleichberechtigt nebeneinander. Die Integration der beiden Frameworks erfolgt über eine separate Integrationsschicht, die in Stufe III manuell durch den Anwendungsentwickler vorgenommen werden musste. Diese Schicht bietet dem Entwickler nun eine einheitliche, integrierte Schnittstelle an, mit der er sowohl Prozesse als auch Helfer und Ressourcen in einem Modell spezifizieren kann.

Der Vorteil gegenüber Stufe IV besteht darin, dass in Stufe V alle Konzepte in vollem Umfang genutzt werden können. Im Gegensatz zu Stufe III wird die Integration durch das neue Framework übernommen. Diese Stufe ist damit die mächtigste der vorgestellten Entwicklungsstufen.

5.4.6 Aktueller Stand und weiteres Vorgehen

Nachdem die verschiedenen Stufen für eine Architektur aufgeführt worden sind, muss eine Designentscheidung getroffen werden, welche Stufe für die Implementierung des Potato-Frameworks gewählt wird.

Wünschenswert wäre natürlich die komplette Flexibilität, die eine Integration der Stufe V bieten würde. Allerdings bestehen hierfür noch zu viele offene Fragen, da es sich hierbei um eine völlig neue Art von Objekten handeln würde, aus denen das System besteht. Jeder Teilnehmer des Systems kann sowohl als Prozess als auch als Strukturelement aufgefasst werden. Diskussionen hierzu finden sich in [Tell und Moldt, 2005].

In der Vergangenheit wurden Implementationen anhand der verschiedenen Architekturstufen vorgenommen. In den Siedler-Lehreprojekten [Bosch et al., 2002, Cabac et al., 2005] wurde die Entwicklung direkt auf dem Multiagentensystem aufgesetzt. Hier wurde nach der obigen Terminologie also eine Architektur der Stufe I verwendet.

In weiteren Projekten wurden sowohl das Helfersystem [Lehmann und Markwardt, 2004, Lehmann et al., 2005, Willmott et al., 2005, Markwardt und Moldt, 2010] als auch die Prozessinfrastruktur [Reese et al., 2005, Reese et al., 2006a, Reese et al., 2006c, Markwardt et al., 2008a]

als neue Architekturschicht entsprechend einer Stufe II-Architektur entwickelt. Diese Systeme wurden zunächst unabhängig voneinander eingesetzt. Die Verwendung beider Systeme nebeneinander in einer Stufe III-Anwendung [Markwardt et al., 2008b] ergab dann die erläuterten Probleme der inkompatiblen Paradigmen. Ansätze zu einer Integration der Stufe IV finden sich in [Markwardt et al., 2009b, Markwardt et al., 2009a, Wagner, 2009c, Moldt et al., 2010, Wagner et al., 2011]. Ausgehend davon wird in dieser Arbeit die weitere Integration vorangetrieben. Hierbei wird konkret eine Architektur der Stufe IV(a) gewählt. Das Potato-System wird realisiert als agentenorientiertes WfMS, aber unter Verwendung der Helfer- und Ressourcenagenten aus dem HeRA-System. Verwender des Potato-Systems können dann ihre Prozesse definieren, die im jeweiligen Anwendungskontext erforderlich sind und zusätzlich Helferagenten und Ressourcen, die innerhalb der Prozesse Verwendung finden.

Für den Anwendungsfall der Verteilten Softwareentwicklung sind verschiedene Teilprozesse denkbar, die teilweise als wiederholbare Workflows, teilweise als kontextabhängige Workflows für ein bestimmtes Projekt definiert werden. Innerhalb dieser Workflows werden dann die verschiedenen Ressourcen, von Anforderungserhebungen über Designdokumente bis hin zu Klassen und compilierten Programmen verwaltet und von den verschiedenen beteiligten Parteien mit verschiedenen Helferagenten bearbeitet.

Zur Gestaltung der verschiedenen Helfer und Prozesse innerhalb eines Systems können Techniken des Process Mining verwendet werden, mit denen die vorherrschenden Verwendungsmuster einer Software analysiert werden können, um so Strukturen zu ermitteln, die in neuen Helfern oder Prozessen formalisiert werden können. Zum Thema Process Mining in Multiagentensystemen vergleiche auch [Cabac et al., 2006, Cabac und Knaak, 2007].

5.5 Referenzmodell

Der Vorteil der Verwendung von Referenznetzen zur Systemmodellierung liegt darin, dass die verschiedenen interagierenden Bestandteile als einzelne Netze modelliert und anschließend gemeinsam simuliert werden können. Obwohl es sich um eine verhältnismäßig anschauliche Art der Modellierung handelt, ist die Semantik eindeutig und klar definiert, so weit, dass die Modelle direkt den ausführbaren Code darstellen. In diesem Abschnitt wird ein vereinfachtes Modell der Grundkomponenten des Potato-Systems vorgestellt, an dem das Zusammenspiel dieser Komponenten erläutert werden kann. Die tatsächliche Implementierung ist keine direkte Verfeinerung dieses Modells, orientiert sich aber an dessen Strukturen. Das komplette Modell mit allen dazugehörigen Referenznetzen findet sich in Anhang A.

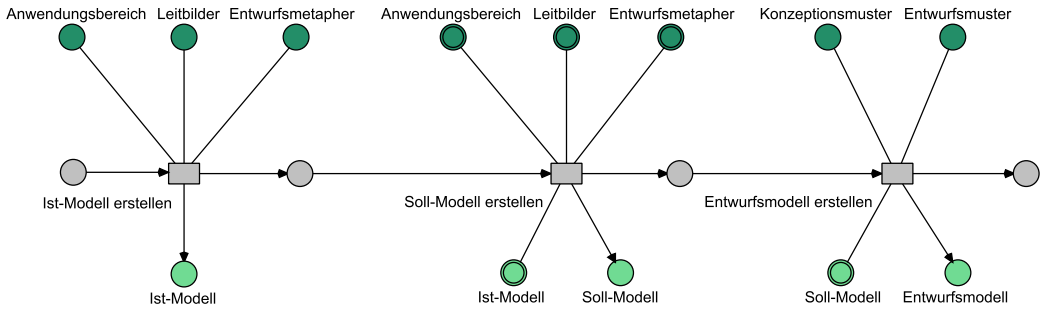

Abbildung 5.8: Modell eines Entwicklungsprozesses (Ausschnitt) (nach: [Tell, 2005, S. 88])

5.5.1 Softwareentwicklung

Softwareentwicklung ist ein komplexer Prozess, der aus einer Reihe verschiedener Aktivitäten besteht, verschiedene Arten von Dokumenten und Ressourcen verwendet und durch verschiedene Akteure vorangetrieben wird. Eine Modellierung des Softwareentwicklungsprozesses mit Referenznetzen findet sich in [Tell, 2005, S. 86ff]. Abb. 5.8 zeigt einen Teil des Prozesses.

Die einzelnen Blöcke stellen die Aktivitäten des Entwicklungsprozesses dar, wobei die Transition in der Mitte jeweils die Durchführung der Aktivität selbst modelliert. Die Stellen im oberen Bereich stellen die Umgebung dar, in der die Aktivität ausgeführt wird, die Stellen im unteren Bereich die verschiedenen Dokumente, die erzeugt und in späteren Aktivitäten wieder verwendet werden[1]. Welche Elemente die Umgebung einer Aktivität ausmachen und welche Dokumente im Prozess verwendet werden, ist natürlich von der jeweiligen Ausprägung des Prozesses abhängig.

Auch die Anordnung der Aktivitäten ist hier nur als Beispiel zu verstehen. In einem konkreten Entwicklungsprozess können eine Reihe unterschiedlicher Aktivitäten vorkommen und auch Zyklen sind möglich, wenn Modelle in Iterationen erweitert und verfeinert werden.

Dieses Grundmodell lässt sich nun um weitere Aspekte anreichern, die das Potato-System charakterisieren. Dadurch wird eine Aktivität im Entwicklungsprozess nicht mehr nur durch eine einfache Transition dargestellt, sondern der Kontext genauer ausdifferenziert. Als erstes werden die Akteure, die die verschiedenen Aufgaben bearbeiten, sowie die Hilfsmittel und Ressourcen als eigene Entitäten explizit gemacht. Anschließend werden in einem separaten Modell die Abläufe und die Rolle der verschiedenen Bearbeiter in den Abläufen in Form einer Prozessinfrastruktur modelliert. Schließlich werden die beiden Modelle integriert zu einem Referenzmodell für das Potato-System.

[1] Die doppelt umrandeten Stellen sind virtuelle Stellen, die eine Referenz auf die Originalstellen darstellen und hier aus Layoutgründen verwendet werden.

5.5.2 Helfer und Ressourcen

Das erste Grundmodell, das hier vorgestellt wird, modelliert die Konzepte von Helfern und Ressourcen. Akteure repräsentieren Benutzer und autonome Dienste im System, die Ressourcen und Helfer verwenden können, um ihre Aufgaben auszuführen. Eine Helferfabrik erzeugt neue Helfer für die Behandlung von Ressourcen.

5.5.2.1 Akteure

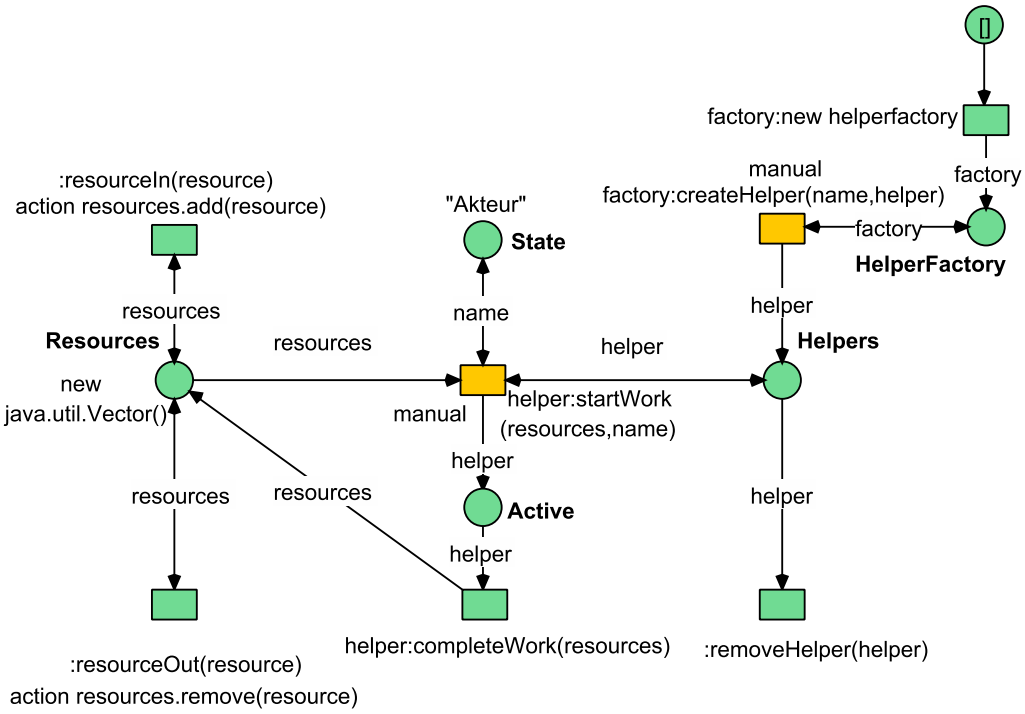

Abbildung 5.9: Akteur

Aufgaben werden durch Akteure bearbeitet (siehe Abb. 5.9). Akteure besitzen einen internen Zustand (hier symbolisiert durch einen Namen, der ihnen bei der Erzeugung gegeben wird). Dies stellt unter anderem auch das Wissen eines menschlichen Akteurs, Erfahrungen, Ressourcen außerhalb des Systems oder eine bestimmte Strategie dar, die verfolgt wird. Die Ausprägung dieses Zustandes bestimmt unter anderem, auf welche Weise eine Aufgabe durch diesen Akteur behandelt wird.

Helfer können dem Akteur über die Helferfabrik hinzugefügt werden. Im Modell, das nur einen Akteur beinhaltet, wird die Helferfabrik direkt im Akteur erzeugt, in einem verteilten System ist dies ein Dienst, auf den verschiedene Akteure zugreifen können.

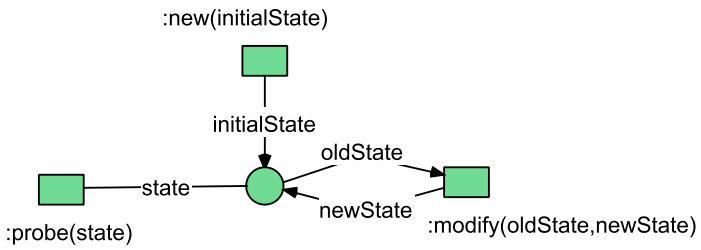

Abbildung 5.10: Ressource

Ressourcen können entweder über die vorhandenen `:resourceIn()`- bzw. `:resourceOut()`-Transitionen hinzugefügt und entfernt werden oder durch Helfer erzeugt und bearbeitet werden, die Zugriff auf die Ressourcen des Akteurs eingeräumt bekommen.

Der Akteur kann die Helfer nutzen, um Aktionen auszuführen. Dies wird dargestellt durch die `helper:startWork()`-Transition. Der Akteur entscheidet (über die Wahl der Bindung beim Schalten der Transition), welche Helfer er verwenden will, was genau diese tun, ist in jedem Helfer selbst implementiert. Eine Tätigkeit ist damit beeinflusst durch drei Faktoren: Den Akteur und dessen Kontext, den verwendeten Helfer und die vorhandenen Ressourcen.

Für die Dauer der Tätigkeit übernimmt der Helfer die Kontrolle über die Ressourcen des Akteurs, die daher aus der entsprechenden Stelle entfernt werden, bis sie durch die `helper:completeWork()`-Transition wieder freigegeben werden.

5.5.2.2 Ressourcen

Ressourcen stellen Arbeitsgegenstände im System dar und kapseln den Zustand eines solchen Materials. Die Ressourcen innerhalb eines Agenten können auf Anforderung von außen hin bearbeitet werden. Dafür wird über einen Kanal auf die Ressource zugegriffen, die daraufhin ihren Zustand anzeigen oder ändern kann.

Spezifische Helfer und Ressourcen unterscheiden sich zum einen im Inhalt, also dem Zustand der Ressource, zum anderen in den Zugriffsmustern, die der Helfer erlaubt. Hier ist dieser Zugriff sehr abstrakt dargestellt über einen `:probe()`-Kanal für die Ansicht und einen `:modify()`-Kanal für die Bearbeitung, die auf der Ressource durchgeführt werden soll (siehe Abb. 5.10). Wie dies in eine konkrete Zustandsänderung der Ressource umgesetzt wird, macht die spezifische Helfer-Ressourcen-Kombination aus.

5.5.2.3 Helfer und Helferfabrik

Neue Helfer werden dem Akteur über die Helferfabrik hinzugefügt. Diese kennt die verschiedenen Arten von Helfern und kann neue Instanzen davon erzeugen

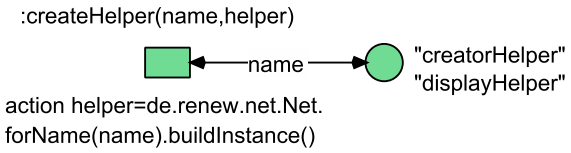

Abbildung 5.11: Helferfabrik zum Erzeugen neuer Helfer

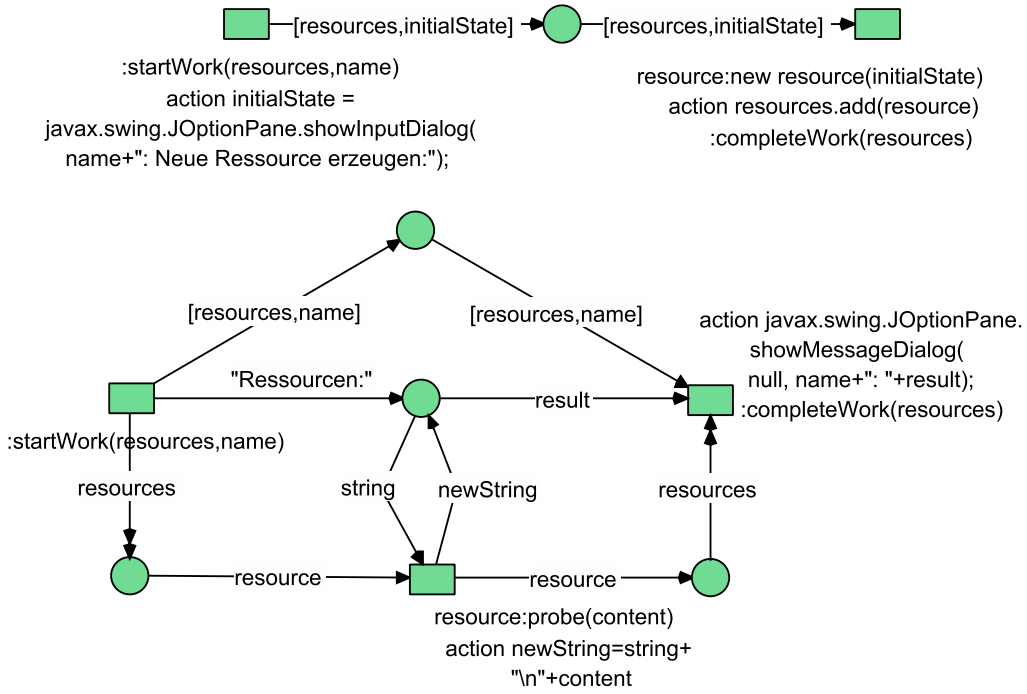

Abbildung 5.12: Helfer zum Erzeugen und Anzeigen von Ressourcen

und dem Akteur verfügbar machen. Auf diese Weise kann neue Funktionalität an einer zentralen Stelle abgelegt werden, um sie systemweit zur Verfügung zu stellen (siehe Abb. 5.11).

Helfer sind zuständig für die tatsächliche Ausführung von Aktionen und zum Erzeugen und Bearbeiten von Materialien. Im Modell sind exemplarisch zwei Helfer enthalten, einer zur Erzeugung einer neuen Ressource, einer zum Anzeigen der vorhandenen Ressourcen.

Die Helfer werden über die Transitionen `:startWork()` und `:completeWork()` mit dem Akteur verbunden, der damit den Helfer aktiviert und mit dem nötigen Kontext (Akteur und Ressourcen) versorgt. Zwischen diesen Transitionen hat der Helfer vollen Zugriff auf die Ressourcen. Im Referenzmodell werden einige einfache Java-Dialoge angezeigt, um die Funktionsweise zu veranschaulichen (siehe Abb. 5.12).

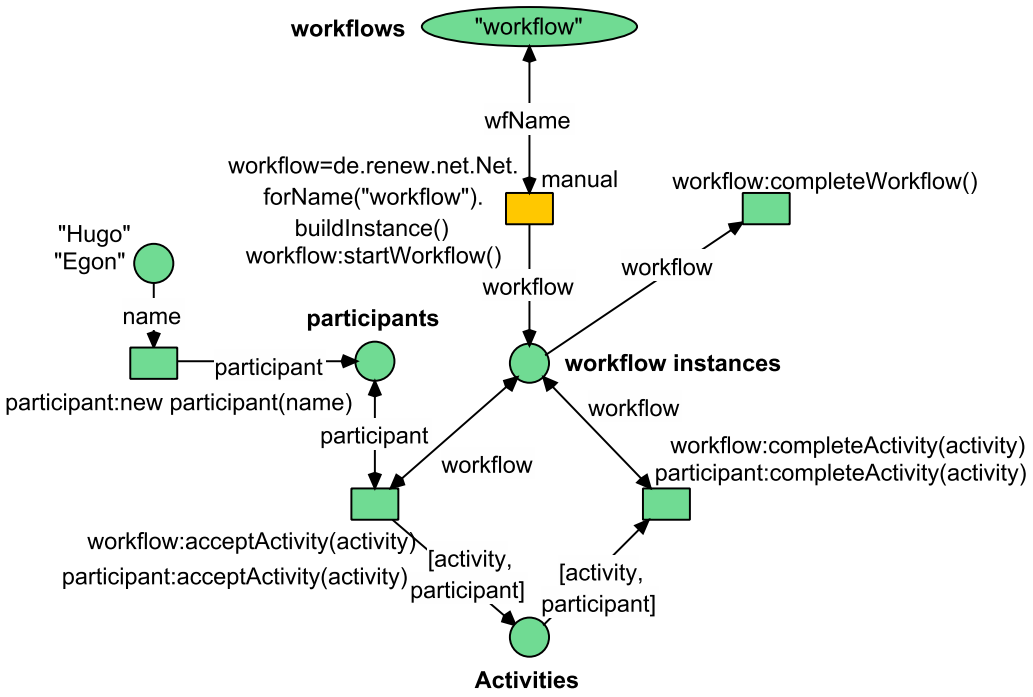

Abbildung 5.13: Das Workflow Management System

5.5.3 Prozesse

Als nächstes wird das Workflow Management System (WfMS) modelliert. Der Zweck dieses Systems ist es, Abläufe zu koordinieren und Teilnehmern des Systems ihre Aufgaben zur Erledigung zuzuteilen.

In diesem einfachen Modell wird das WfMS als ein Netz modelliert, anstatt verschiedene Bestandteile, die später als separate Agenten modelliert werden, in separate Netze zu zerlegen. Dadurch wird der Fokus der Modellierung mehr auf die Interaktion der Workflow-Teilnehmer mit dem WfMS gelegt. In diesem Bereich erfolgt später auch die Integration mit dem Helfer- und Ressourcensystem.

5.5.3.1 Das Workflow Management System

Das WfMS verwaltet Prozessdefinitionen, Workflow-Teilnehmer und aktive Workflow-Instanzen. Es ist zuständig für die Zuweisung von Aufgaben an die Teilnehmer und achtet auf die korrekte Abwicklung (siehe Abb. 5.13).

Die Stelle „workflows“ enthält Referenzen zu den bekannten Workflow-Mustern. In diesem Modell ist in dieser Stelle nur ein Workflow-Muster mit dem Namen „workflow“ vorhanden, in weiteren Verfeinerungen des Modells könnten neue Workflowdefinitionen hier abgelegt werden, um das System um

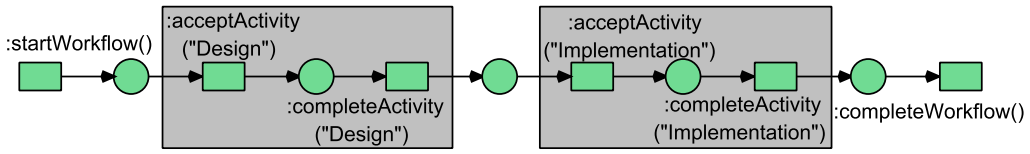

Abbildung 5.14: Beispiel-Workflow

weitere Varianten zu erweitern.

Beim Start des Netzes werden die Teilnehmer als Netzinstanzen erzeugt und in einer hierfür vorgesehenen Stelle abgelegt. Das stellt die Anmeldung von Benutzern am System dar, die nun bereit sind, Aufgaben zu übernehmen. Über eine Transition können Instanzen von den bekannten Workflownetzen erzeugt werden, die dann in der Stelle für aktuell ausgeführte Workflow-Instanzen abgelegt werden. Workflow-Teilnehmer und Workflow-Instanzen sind jeweils eigene Netzinstanzen, die hier referenziert werden.

Wenn eine Aufgabe im Workflow verfügbar ist, kann sie über die `:acceptActivity()`-Transition einem Teilnehmer zugewiesen werden. In einem WfMS würde diese Zuweisung über die Aufgabenliste erfolgen, hier erfolgt die Zuweisung über die Auswahl der Bindungen beim Schalten der Transition. Die Zuweisung der Aufgaben an die Teilnehmer erfolgt willkürlich. Entsprechende Regeln könnten im Workflow vermerkt und in einer weiteren Verfeinerung über einen neuen Parameter im synchronen Kanal implementiert werden.

Die Aufgabe und der zugeordnete Teilnehmer werden dann als Aktivität auf die entsprechende Stelle gelegt, auf der sich alle aktuell bearbeiteten Aufgaben befinden. Durch das synchrone Schalten der `:completeActivity()`-Transitionen im Workflow und im Teilnehmer wird die Aufgabe als erledigt registriert und von dort wieder entfernt. Die `:completeWorkflow()`-Transition signalisiert den Abschluss einer Workflow-Instanz und entfernt die Instanz aus dem WfMS.

5.5.3.2 Workflow

Der Workflow, der hier im Beispiel verwendet wird, ist sehr simpel und besteht lediglich aus zwei sequentiellen Aufgaben (vgl. Abb. 5.14). Um Raum für die Ausführung der Aufgaben zu haben sind sie jeweils aufgeteilt in eine `:acceptActivity()`- und eine `:completeActivity()`-Transition, zwischen denen die tatsächliche Bearbeitung beim Workflow-Teilnehmer stattfindet. Die Transitionen `:startWorkflow()` und `:completeWorkflow()` dienen zur Synchronisation mit dem WfMS.

Komplexere Workflows sind damit möglich, würden aber unter Umständen schnell unübersichtlich werden. Zudem ist es mit diesem Modell nicht möglich, eine einmal begonnene Aufgabe abzubrechen. In der implementierten Version

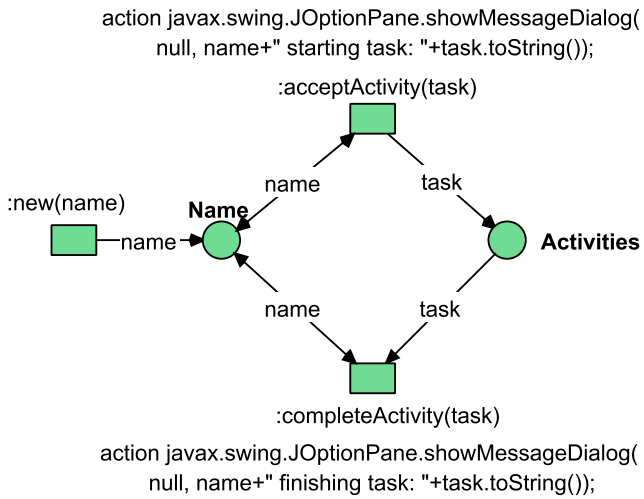

Abbildung 5.15: Workflow-Teilnehmer

der Prozessinfrastruktur wird daher als Kurzschreibweise eine spezielle Art von Transition, die Task-Transition verwendet, die es erlaubt, einen Schaltvorgang wieder abzubrechen. Zusätzlich kann hierbei bereits durch die Aktivierung einer Aufgabe eine Aktion ausgelöst werden (etwa ein Update der Worklisten). Mehr dazu in Abschnitt 7.2.2.

5.5.3.3 Workflow-Teilnehmer

Der Workflow-Teilnehmer (Abb. 5.15) verfügt über einen internen Zustand (auch hier dargestellt durch ein Feld für den Namen) und kann Aufgaben im Workflow zur Bearbeitung annehmen. Zur Verdeutlichung, was im System passiert, wird jeweils eine Nachricht ausgegeben, wenn eine Aufgabe angenommen oder abgeschlossen wird.

In diesem Modell wird die tatsächliche Bearbeitung der Workflow-Aufgaben nicht modelliert. Es liegt also außerhalb dieses Modells, was tatsächlich getan wird, um die Aufgabe zu erledigen. Stattdessen wird lediglich erfasst, dass die Bearbeitung vollständig ist. Der nächste Schritt besteht daher darin, die beiden vorgestellten Modelle zu kombinieren, so dass Helfer im Workflow verwendet werden können, um Aufgaben zu bearbeiten.

5.5.4 Kombination - POTATO

Im POTATO-System sollen sowohl Helfer und Ressourcen verwendet werden können, um Aufgaben zu bearbeiten, als auch eine Prozessinfrastruktur, die die Entwicklungsprozesse koordiniert. Ein Helfer wird verwendet, um den Benutzer gegenüber dem WfMS zu vertreten. Dieser nimmt also die Rolle des Benutzers ein, nimmt Aufgaben im Workflow zur Bearbeitung an und bearbeitet sie. Die

Bearbeitung der Aufgaben auf der anderen Seite wird ebenfalls durch Helfer ausgeführt, die hier Aktivitätsagenten heißen, da sie für einzelne Aktivitäten im Workflow speziell erzeugt werden.

5.5.4.1 Workflow

Zu diesem Zweck wird der Beispiel-Workflow erweitert zu einem einfachen Softwareerstellungs-Workflow. Die erste Aufgabe ist die Erstellung eines Designs (bspw. für ein neues Software-Modul). Im Gegensatz zum einfachen WfMS wird diese Aufgabe nicht einfach nur textuell beschrieben, sondern es wird ein Helfer für die Aktivität erzeugt, sobald die Aufgabe angenommen wird. Dieser erzeugt während seiner Ausführung eine neue Ressource für das Designdokument.

Diese Design-Ressource wird im Workflow gespeichert[2]. Es folgt dann eine Aktivität zur Beurteilung des Designs. Diese erhält die vorher erzeugte Ressource übergeben und kann sie dann bei der Ausführung nutzen. Je nachdem, ob das Design-Dokument akzeptiert wird, wird der Workflow abgeschlossen oder eine neue Aktivität zum Überarbeiten der Ressource aufgerufen. Den kompletten Workflow zeigt Abb. 5.16. Dieses Netz ist verhältnismäßig groß, da für jede Aktivität zwei Transitionen mit verschiedenen Anschriften benötigt werden. In Pia wird zu diesem Zweck eine Task-Transition verwendet, die Annehmen, Abschließen und Abbrechen einer Aufgabe in einem spezialisierten Netzelement zusammenfasst (siehe Abschnitt 7.2.2).

5.5.4.2 Akteur

Der Akteur aus dem HeRA-System wird für das Beispiel geringfügig modifiziert, so dass er bei der Erzeugung bereits eine Helferfabrik zugewiesen bekommt. Diese Helferfabrik wird dann mit dem WfMS bekannt gemacht und kann daher dem Akteur einen speziellen Workflow-Helfer zur Verfügung stellen. Dieser kennt das WfMS und kann sich dort als Stellvertreter für den Akteur anmelden.

5.5.4.3 Workflow-Helfer

Dieser spezielle Helfer dient als Schnittstelle zwischen dem Akteur und dem WfMS. Er wird als Helfer in den Akteur integriert und meldet sich in dessen Namen beim WfMS an, füllt also die Rolle des Workflow-Teilnehmers aus. Daher ist dieser Helfer eine Mischung aus einem Helfer und dem Teilnehmer aus der Prozessinfrastruktur (siehe Abb. 5.17).

Wenn er sich in einem aktiven Status befindet (also zwischen `:startWork()` und `:completeWork()`), kann er Aktivitäten im Workflow annehmen. So lange

[2]Im Netz werden zwei Stellen mit der Bezeichnung „Designdocument“ verwendet; die doppelt umrandete Stelle ist eine virtuelle Stelle, die eine Referenz zur Originalstelle darstellt und hier aus Layout-Gründen verwendet wird.

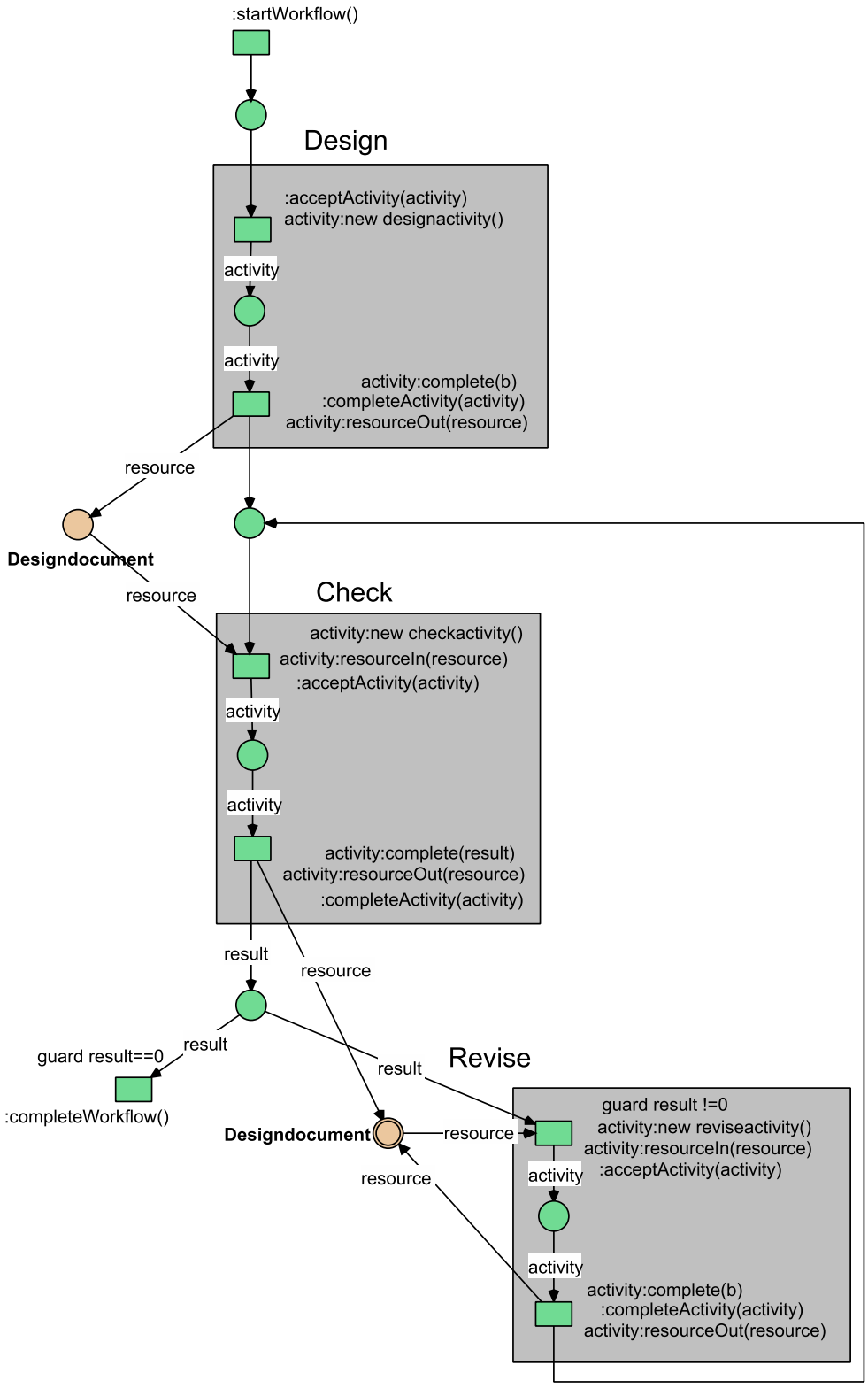

Abbildung 5.16: Beispiel-Workflow für POTATO

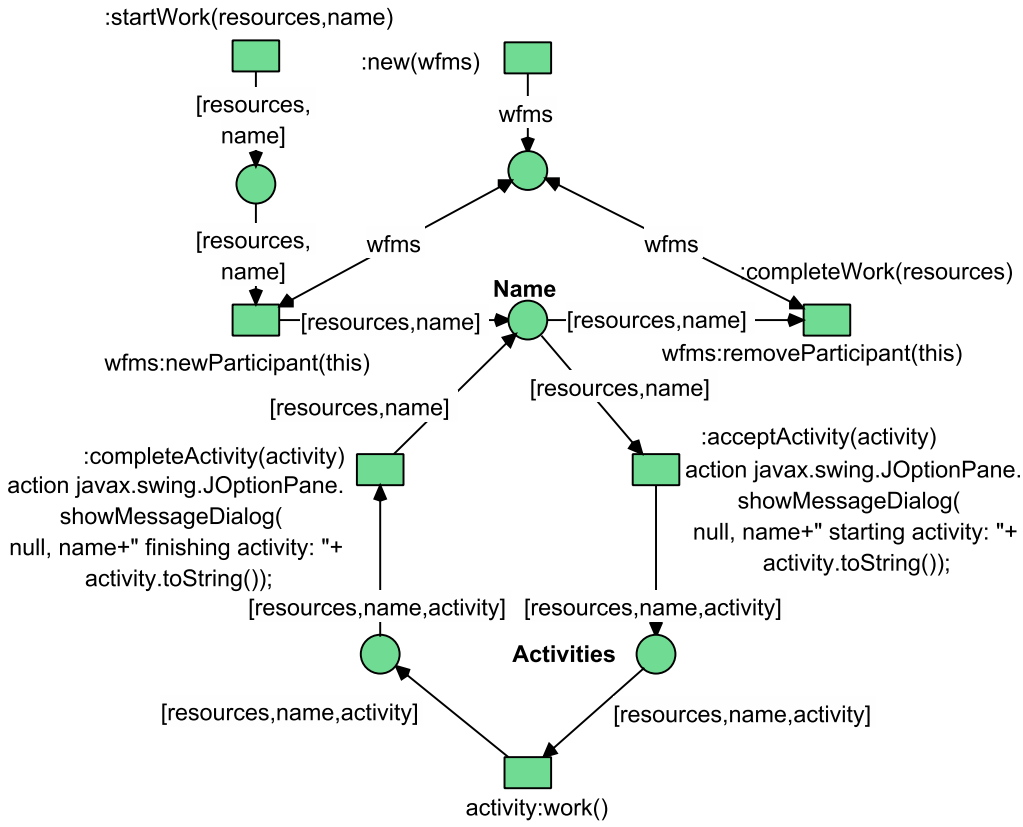

Abbildung 5.17: Workflow-Helfer

eine Aktivität aktiv ist, kann die Bearbeitung nicht beendet werden. Wie der Workflow-Teilnehmer in der Prozessinfrastruktur führt er die Aktivität aus, indem er die entsprechende `:work()`-Transition der Aktivität schaltet.

5.5.4.4 Aktivitäten

Die Aktivitäten im POTATO-System werden durch spezialisierte Helfer ausgeführt, die für jeweils eine Aktivität verantwortlich sind. Wie Helfer können sie Ressourcen erzeugen und bearbeiten, zusätzlich können sie diese Ressourcen mit dem Workflow austauschen. Dadurch können die Ressourcen, die zu einer Prozessinstanz gehören, direkt im Workflow gespeichert werden, um an Folgeaktivitäten weitergereicht zu werden.

Abb. 5.18 zeigt eine Aktivität zum Überarbeiten einer Ressource im Workflow. Im Beispiel-Workflow wird diese Aktivität verwendet, wenn das Design als nicht korrekt abgelehnt wird und überarbeitet werden muss. Über die `:resourceIn()`-Transition bringt der Workflow die zu bearbeitende Ressource in die Aktivität ein, bevor sie an den Workflow-Teilnehmer weitergegeben wird.

Über die `:work()`-Transition wird die tatsächliche Bearbeitung durchgeführt (hier über ein einfaches Dialogfeld) und anschließend die Änderung an der Ressource durchgeführt, bevor die Aktivität abgeschlossen werden kann. Der Workflow liest dann die geänderte Ressource wieder aus, die dann im weiteren Verlauf des Workflows weiter verwendet werden kann.

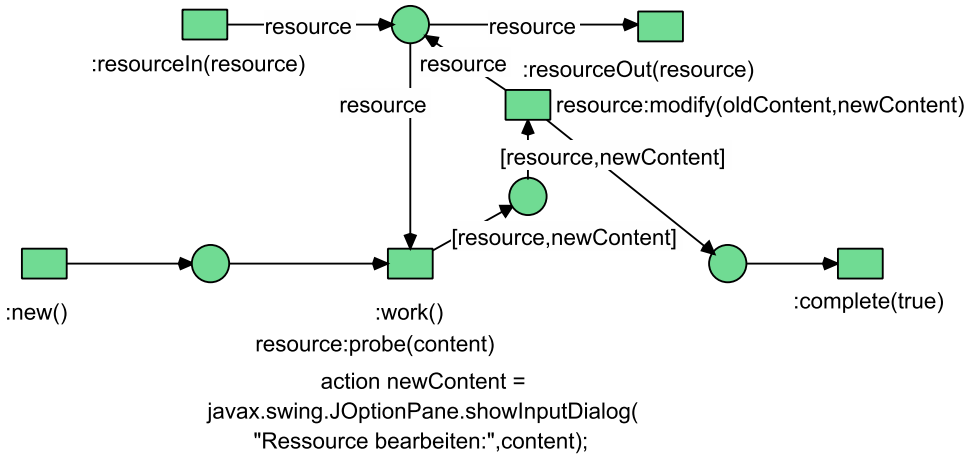

Abbildung 5.18: Aktivität zum Bearbeiten einer Ressource im Workflow

5.5.5 Anmerkungen zum Referenzmodell

Das vorliegende ausführbare Modell stellt die Zusammenhänge zwischen den verschiedenen Bestandteilen des POTATO-Systems und der Teilsysteme HERA und PIA dar. Dabei werden natürlich nicht alle Details wiedergegeben, sondern

lediglich die generelle Idee hinter dem System und das Zusammenspiel der verschiedenen Bestandteile deutlich gemacht.

Die weitere Entwicklung des POTATO-Systems ist keine Erweiterung dieses Modells, sondern eine eigenständige Entwicklung auf der Basis des MULAN/CAPA-Agentenframeworks. Die Grundideen, die hier dargestellt werden, dienen aber als Orientierung für den folgenden Entwurf und die prototypische Implementation.

5.6 Grobentwurf

Ziel dieses Abschnittes ist es, den allgemeinen Rahmen abzustecken, in dem die Implementierung der verteilten Entwicklungsumgebung erfolgen soll. Dies umfasst die verwendeten Technologien und Frameworks, sowie Grundsatzentscheidungen bezüglich der Strukturierung des Systems.

5.6.1 Agenten

Wie bereits eingangs erwähnt, sollen das POTATO System und die darauf aufbauende verteilte Entwicklungsumgebung als Multiagentensystem implementiert werden. Für die Implementierung von Multiagentensystemen gibt es eine Reihe von Frameworks und Umgebungen, die in Abschnitt 3.2 erwähnt wurden. Für die konkrete Implementierung werden hier der PAOSE-Ansatz und die Agentenplattform CAPA gewählt. Im PAOSE-Ansatz wird der Schwerpunkt auf die Modellierung der verschiedenen Rollen und ihrer Interaktionen gelegt, welche sich dann jeweils als Agenten und Protokolle in CAPA wiederfinden lassen.

In dieser Arbeit wird dieser agentenorientierte Softwareentwicklungsansatz nicht nur exemplarisch durch seine Verwendung deutlich gemacht, er soll auch in der Entwicklungsumgebung als beispielhaft zu unterstützender Prozess auftauchen. Während also prinzipiell verschiedene Entwicklungsansätze denkbar sind und durch entsprechende Prozesse und Werkzeuge unterstützt werden können, wird bei der konkreten Entwicklung und der Formulierung von Beispielen auf das hier selbst verwendete Vorgehen Bezug genommen.

Zur Unterstützung der prozessorientierten Aspekte von POTATO wird die Prozessinfrastruktur aufgegriffen, die in [REESE et al., 2005, REESE et al., 2006a, REESE, 2010, WAGNER, 2009c, WAGNER et al., 2011] beschrieben wird und die für die Verwendung hier weiter entwickelt worden ist.

5.6.2 Verwendete Technologien

Eine wichtige Entscheidung, die vor der Implementierung eines Systems getroffen werden muss, ist die Frage nach den verwendeten Technologien, Programmiersprachen, Frameworks und Bibliotheken. Diese Frage muss separat für den

Server-Anteil des Systems, also das Multiagentensystem und den Client-Teil, also die Benutzungsoberfläche, mit der ein Endanwender später arbeitet, beantwortet werden.

5.6.2.1 Multiagentensystem

Die Entwicklung des POTATO-Systems und der darauf aufbauenden Entwicklungsumgebung soll mit Hilfe des PAOSE-Entwicklungsansatzes erfolgen. Dieser Ansatz wurde mit dem RENEW-Werkzeug und der darauf aufbauenden MULAN/CAPA-Agentenplattform entwickelt und verwendet diese auch für die Entwicklung. Daher werden diese Werkzeuge auch für die Entwicklung von POTATO verwendet werden.

Es gibt Gründe, die gegen eine Produktentwicklung auf dieser Plattform sprechen würden. Insbesondere sind dies die Performanz und die sehr spezialisierte Einarbeitung, die für eine Entwicklung mit diesen Werkzeugen erforderlich ist. In Rahmen dieser Arbeit können diese Gründe aber vernachlässigt werden: Performanz ist kein zentrales Kriterium bei dieser Art von Prototyp, ein Produktivsystem kann mit den hier gewonnenen Erfahrungen auf einer anderen Plattform reimplementiert werden. Die benötigten Kenntnisse sind bereits vorhanden und können direkt verwendet werden. Auf der anderen Seite bietet die Entwicklung mit Referenznetzen eine spezielle Sicht auf die Softwareentwicklung, die nur auf diese Weise erlangt werden kann.

5.6.2.2 Workflow

Für die Definition und Ausführung von Prozessen innerhalb der Entwicklungsumgebung sollen Workflows verwendet werden. Der Entwicklung von [REESE, 2010] folgend, wird dafür eine agentenbasierte Prozessinfrastruktur verwendet. Dabei handelt es sich um ein Workflow Management System (WfMS), das mit Hilfe von PAOSE in CAPA entwickelt worden ist. Die Möglichkeiten der Modellierung und Ausführung von Workflows innerhalb von RENEW können damit direkt innerhalb des Systems genutzt werden.

5.6.2.3 Benutzungsoberfläche

Bei der Entwicklung der Benutzungsoberfläche gibt es verschiedene mögliche GUI-Frameworks. Da das Multiagentensystem in Java implementiert ist, bietet es sich an, hier eine Beschränkung auf Java-basierte Frameworks vorzunehmen.

Eine Gegenüberstellung verschiedener GUI-Frameworks würde hier zu weit führen. Für die Entwicklung eines Prototypen wurde die Rich Client Platform (RCP) [RCP, 2011, MCAFFER und LEMIEUX, 2005] der Eclipse Foundation [Eclipse, 2011] gewählt. Zum einen ist damit bereits eine leistungsstarke Einzelplatz-IDE direkt vorhanden, zum anderen bietet sie ein Framework für Applikationen, welches durch Plugins flexibel erweitert werden kann.

Die Eclipse-Entwicklungsumgebung selbst ist eine Menge von Plugins innerhalb der Rich Client Plattform, es können aber auch andere Arten von Anwendungen realisiert werden. Mit dem Standard Widget Toolkit (SWT) [SWT, 2011] können darin grafische Benutzungsoberflächen erstellt werden. Das System ist unter einer OpenSource Lizenz verfügbar. Ein Vorteil bei der Verwendung von RCP liegt darin, dass durch die Plugin-Struktur und die zahlreichen existierenden Eclipse-Plugins das POTATO-System von den bereits existierenden Funktionen zur lokalen Softwareentwicklung mit Eclipse profitieren kann.

5.6.3 Implementationstiefe

Ziel dieser Arbeit ist es nicht, eine komplette verteilte Entwicklungsumgebung oder auch nur ein vollständiges Framework zu implementieren. Vielmehr werden die konzeptionellen Grundlagen hierfür erörtert und ein Entwurf entwickelt, aus dem ein solches Framework entwickelt werden kann. Die grundlegenden Konzepte und Zusammenhänge werden prototypisch implementiert, sowie das Vorgehen bei der Erstellung herausgearbeitet.

Grundlegende Funktionen sollen darin vorhanden und nutzbar sein, um eine sinnvolle Evaluation des Ergebnisses zu ermöglichen. In diesem Abschnitt wird daher festgelegt, welche Funktionen in den einzelnen Bereichen zu implementieren sind.

5.6.3.1 HeRA

Die Unterstützungsumgebung HeRA ermöglicht es dem Benutzer, verschiedene Helfer und Ressourcen zu erzeugen, zu verwalten und zu verwenden. Der Benutzeragent ermöglicht ihm den Zugriff auf diese Funktionen. Die prototypische Implementation muss eine verwendbare Benutzungsoberfläche enthalten, mit der ein Anwender bei der Helferfabrik nach neuen Agenten suchen und diese in seinen Arbeitsplatz laden kann.

Ebenso werden Helferagenten implementiert, die geladen werden können und die sich in die Benutzungsoberfläche einpassen. Diese Helfer stellen die diskutierten Konzepte dar und müssen insbesondere dem Benutzer die Möglichkeit geben, Ressourcenagenten zu erzeugen und zu bearbeiten.

Zusätzlich wird ein Helferagent implementiert, um auf das WfMS zuzugreifen. Dies umfasst die Anmeldung am System, Starten von Workflows, Anzeige der Worklist, und das Auswählen von Aktivitäten zur Bearbeitung. Weitere Helferagenten erlauben dann die Bearbeitung von Aufgaben im Workflow.

5.6.3.2 Workflow Management System

Das Workflow Management System bildet die verbindende Struktur, die die Zusammenarbeit der Anwender im System reguliert. Es muss die typischen

Aufgaben eines WfMS erfüllen können. Neben den oben genannten Benutzerinteraktionen muss es dafür sorgen, dass die Prozesse korrekt ausgeführt werden, die richtigen Bearbeiter für die jeweiligen Aufgaben ausgewählt werden und die Bearbeitung der Aufgaben ermöglicht wird.

Es soll mit dem Prototypen zumindest möglich sein, einen einfachen Workflow zu instanziieren, bei dem verschiedene Benutzer Aufgaben zugewiesen bekommen und diese mit Hilfe des Systems bearbeiten können.

Spezielle Möglichkeiten zum Monitoring der Prozesse werden nicht implementiert, ebensowenig wie administrative Eingriffe in laufende Prozesse. Hierfür kann bei Bedarf auf die Möglichkeiten des MulanViewers und anderer Renew-Werkzeuge zurückgegriffen werden.

5.7 Zusammenfassung

In diesem Kapitel wurde das Potato-System als Basis für die Entwicklung einer verteilten Softwareentwicklungsumgebung vorgestellt. Als grundlegende Strukturierungsschwerpunkte für die Entwicklung von Multiagentensystemen wurden die Aspekte von Struktur und Prozess identifiziert.

Für den Strukturaspekt wird eine Orientierung an den Artefakten des Anwendungsbereiches vorgeschlagen. Dafür werden Ideen des Werkzeug und Material Ansatzes auf Multiagentensysteme übertragen und erweitert. Der Prozessaspekt wird durch ein agentenorientiertes Workflow Management System unterstützt.

In Kapitel 4 wurden verteilte Softwareentwicklungsprozesse untersucht. Hieraus wurden nun Anforderungen für die Unterstützung verteilter Softwareentwicklung aufgestellt. Als Hauptaspekte wurden auch hier die Unterstützung von Aktivitäten durch geeignete Hilfsmittel und Ressourcen, sowie die Organisation dieser Aktivitäten durch eine Prozesssteuerung identifiziert.

Auf Basis dieser Anforderungen wurden analog zu den Betrachtungen in [Reese, 2010] verschiedene Architekturmodelle diskutiert, um die Struktur- und Prozessaspekte in einer Architektur zu vereinen. Das Modell, das in dieser Arbeit verfolgt wird, verwendet das System der Helfer- und Ressourcenagenten als einen Baustein in der Umsetzung eines agentenorientierten Workflow Management Systems. Die Prozesssteuerung setzt damit auf der Werkzeugunterstützung auf und verwendet die durch dieses System bereitgestellten Agenten.

Um diesen Architekturentwurf zu verdeutlichen und zu konkretisieren, wurde er in Form eines Referenznetzmodells umgesetzt. Dieser einfache Prototyp zeigt die verschiedenen Bestandteile des Potato-Systems und ihr Zusammenwirken. In den folgenden Kapiteln dient es als Orientierung für die Konzeption des HeRA- und Pia-Systems, sowie für deren Integration im Potato-System.

Kapitel 6

Das agentenorientierte Unterstützungssystem HeRA

Die zentrale Frage, die sich bei der Entwicklung agentenbasierter Anwendungssysteme stellt, ist die Frage nach der Strukturierung des Systems. Welche Arten von Agenten enthält das System und auf welche Art und Weise interagieren diese miteinander. Das HeRA-System versucht auf diese Frage eine konzeptionell ansprechende und zugleich anwendungsnahe Lösung zu finden.

Neben der Strukturierung nach FIPA-Spezifikation [FIPA, 2005], die sich in Mulan/Capa wiederfindet, erfolgt hier eine weitere Unterscheidung verschiedener Agententypen. Eine Möglichkeit der Systemstrukturierung ist die Unterscheidung in aktive und passive Komponenten. [Omicini et al., 2008] verwendet Artefakte in Multiagentensystem, um passive Einheiten darzustellen.

Ein anderer Ansatz findet sich im - in der objektorientierten Entwicklung erfolgreichen - Werkzeug und Material Ansatz (WAM) [Züllighoven, 2004]. Bei diesem Ansatz werden Gegenstände aus dem Anwendungsbereich in Software in Form von (Software-)Werkzeugen und Materialien umgesetzt. Durch die Nähe der gewählten Metaphern und Konzepte zu physikalischen Gegenständen aus dem Arbeitsumfeld der Anwender soll die Verwendung der Software erleichtert werden.

Das HeRA-System (Helfer- und Ressourcen-Agenten) wendet die Agentenmetapher auf dieses Konzept an, indem die Gegenstände zu handelnden Entitäten (Agenten) gemacht werden. Statt Werkzeugen werden dann Helfer verwendet, die den Anwender in seinen Aufgaben unterstützen. Ressourcen werden als Oberbegriff für Materialien und andere Ressourcen im System (z.B. Drucker, Scanner, Dateien, Datenbanken) verwendet und ebenfalls als Agenten mit zusätzlicher Autonomie ausgestattet.

Während die Übertragung und Erweiterung der Werkzeug-Metapher im Agentenkontext unmittelbar auf der Hand liegt, ist die Interpretation von Materialien als Agenten nicht sofort offensichtlich. Eine andere Anwendung dieser Übertragung von „eigener Intelligenz“ und „eigener Verantwortung“ für

die Interaktion mit der Umgebung auf bislang als passiv angesehene Elemente findet sich in [Reese, 2010, S. 175ff]. Dort werden Workflows als Agenten modelliert, die damit die Auftraggeber, Benutzer und die Umgebung von zahlreichen komplexen Behandlungen entlasten können, indem sie selbst über die eigene Ausführung wachen.

Im Folgenden werden zunächst die verschiedenen Elemente des WAM-Ansatzes darauf untersucht, wie sie im Kontext eines agentenorientierten Unterstützungssystems verwendet werden können. Nicht alle Details dieses Ansatzes können übernommen werden, sondern er wird als Inspiration verwendet, um aktive und passive Strukturelemente in einem agentenorientierten System zu verwenden. Im Folgenden wird aufgezeigt, welche Eigenschaften dieses Ansatzes sich geeignet übertragen und nutzen lassen.

Anschließend werden die Anforderungen aufgestellt, die an das HeRA-System gestellt werden. Diese ergeben sich zum Teil aus den vorher diskutierten Parallelen zum WAM-Ansatz, zum Teil aus den Erfordernissen der verteilten IDE bzw. des Potato-Systems, die im letzten Kapitel aufgestellt worden sind.

Schließlich werden diese Anforderungen in einen agentenorientierten Entwurf überführt, der, dem Paose-Ansatz folgend, die Ontologie sowie die verschiedenen beteiligten Rollen und Interaktionen beschreibt, die das HeRA-System ausmachen.

6.1 Werkzeuge und Materialien im Paose-Ansatz

Wie die meisten agentenorientierten Softwareentwicklungsansätze legt der Paose-Ansatz den Schwerpunkt auf die Modellierung der verschiedenen Akteure (Agenten) und der Prozesse (Interaktionen), die zwischen diesen Akteuren ablaufen. Für die Entwicklung von Unterstützungssystemen und eine stärkere Fokussierung auf Anwender eines Multiagentensystems vertritt der Autor die Meinung, dass es sinnvoll ist, zusätzlich Ansätze zu integrieren, die mehr auf die individuelle Benutzerunterstützung ausgerichtet sind.

Der Paose-Ansatz erlaubt verschiedene Perspektiven auf die entwickelte Software. Ansätze zu einer anwendungsnäheren Betrachtung finden sich in [Tell, 2005, Tell und Moldt, 2005]. Dort werden auch mögliche Berührungspunkte mit dem WAM-Ansatz aufgezeigt, ohne aber eine konkrete Umsetzung der dort vorgefundenen Konzepte in die Mulan-Umgebung aufzuzeigen. Dies wird nun durch das HeRA-System nachgeholt.

Der Werkzeug und Material Ansatz (siehe Abschnitt 3.5) liefert hierfür verschiedene Metaphern, die im Rahmen dieser Arbeit in den Paose-Ansatz integriert werden sollen. Im Folgenden wird erörtert, wie die Konzepte des WAM-Ansatzes im Rahmen einer Multiagenten-Entwicklung verwendet werden können. Insbesondere werden die Konzepte Werkzeug, Material, Aspekt,

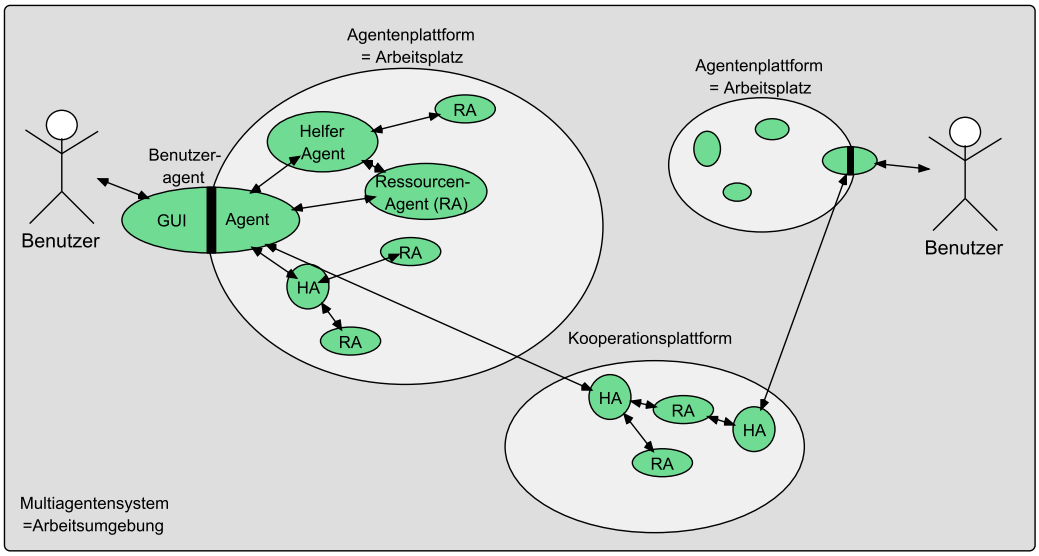

Abbildung 6.1: HeRA-Konzepte in einem Multiagentensystem

Automat, Arbeitsplatz und Arbeitsumgebung beleuchtet.

Zur Verdeutlichung wird ein Whiteboardsystem als Beispielanwendung verwendet, das Benutzer verwenden können, um Gedanken auszutauschen und Ideen zu entwickeln. Dieses Whiteboard soll über das Agentensystem verteilt zugreifbar sein. Eine Beschreibung der prototypische Umsetzung dieses Whiteboardsystems findet sich in Abschnitt 8.3.

6.1.1 Bestandteile des HeRA-Systems

Das HeRA-System liefert Strukturierungsmöglichkeiten für Multiagentensysteme, die für die Unterstützung der individuellen und gemeinsamen Arbeit verschiedener Benutzer entworfen werden. Abb. 6.1 veranschaulicht, wie die verschiedenen Komponenten miteinander in Beziehung stehen.

Anwender greifen über einen Benutzeragenten auf das HeRA-System zu. Dieser Benutzeragent besteht aus zwei Teilen: einer GUI, die der Benutzer für die Interaktion mit dem System verwendet, sowie einem Agenten, der den Benutzer innerhalb einer Agentenplattform repäsentiert. Die Agentenplattform enthält eine Reihe von Agenten, mit denen der Benutzer interagieren kann, um seine Aufgaben zu bearbeiten. Helferagenten unterstützen ihn, indem sie ihm die Verwendung von Ressourcen im System ermöglichen oder andere Funktionen bereitstellen, während Ressourcenagenten passive Systemkomponenten darstellen, die der Benutzer ansehen, bearbeiten, verwenden oder auch mit anderen Benutzern auf anderen Plattformen austauschen kann.

Zentral hierbei ist das Konzept des Helferagenten, der bestimmte Funktionalitäten oder auch den Zugriff auf bestimmte Arten von Ressourcen kapselt. Der Benutzeragent selbst hat keine Kenntnis darüber, welche Art von

Aufgaben im System vorkommen. Es wäre möglich, ihn mit unterschiedlichen Protokollen für verschiedene Aufgaben auszustatten, und ihn so um neue Verhaltensweisen zu erweitern.

Wenn viele verschiedene Arten von Ressourcen und Aufgaben im System vorkommen, führt dies dazu, dass der Benutzeragent schnell sehr umfangreich wird. Stattdessen werden in HeRA Helferagenten verwendet, die bestimmte Fähigkeiten zur Verfügung stellen. Durch diese Kapselung von Funktionalität in einem spezifischen Agenten können neue Verhaltensweisen auf einfache Weise zum System hinzugefügt und wieder daraus entfernt werden.

Die Verwendung von Ressourcen über Helferagenten entspricht einem Model-View-Controller Entwurfsmuster [Krasner und Pope, 1988]. Der Ressourcenagent stellt das Modell dar, der Helferagent dient als Controller, der die Zugriffslogik in seinen Protokollen implementiert. Die GUI-Erweiterung, die der Helferagent zur Darstellung im Benutzeragenten bereitstellt, ist der View, über den der Benutzer auf den Controller zugreifen kann.

Die Helfer- und Ressourcenagenten, über die ein Benutzer verfügt, befinden sich auf seiner Agentenplattform, die seinen Arbeitsplatz darstellt. Er kann mit Agenten auf anderen Plattformen kommunizieren und Ressourcen austauschen. Spezielle virtuelle oder physikalische Plattformen können zudem im Multiagentensystem erstellt werden, auf denen sich Agenten treffen können, um miteinander zu interagieren. Die Menge der Agentenplattformen im System bildet die Arbeitsumgebung.

6.1.2 Beispiel: Whiteboard

Im Folgenden wird ein Beispiel verwendet, um die in den folgenden Abschnitten beschriebenen Konzepte zu erläutern. Das Beispiel, das auch in den folgenden Kapiteln Anwendung findet, ist ein Whiteboard. Darunter ist hier ein einfaches Kollaborationswerkzeug zu verstehen, das verschiedene Benutzer verwenden können, um gemeinsam ein Dokument zu bearbeiten oder ein Meeting abzuhalten.

Das Whiteboard besitzt eine Fläche, auf der Benutzer schreiben und die sie lesen können. Lesen kann jederzeit erfolgen, Schreiben nur im wechselseitigen Ausschluss. Bei einem physikalischen Whiteboard ist dies im Allgemeinen implizit gegeben, da nur eine Person gleichzeitig Stift und Platz an der Tafel hat, um zu schreiben. Im Agentensystem muss dies durch die Agenten realisiert werden. Es wird davon ausgegangen, dass dieser wechselseitige Ausschluss zur Strukturierung der Diskussionen erwünscht ist.

6.1.3 Werkzeuge

Wie in Abschnitt 3.5 erläutert, dient ein Werkzeug der interaktiven Untersuchung und Manipulation von Materialien. Die Art der Benutzung hängt von den spezifischen Eigenschaften des Materials ab. Im HeRA-System werden

Werkzeuge durch Agenten modelliert. Diese sollen aber nicht nur passiv als Werkzeug benutzt werden, sondern auch die Möglichkeit haben, den Benutzer aktiv in seinen Aufgaben zu unterstützen. Daher werden diese Agenten als Helferagenten (oder kurz Helfer, engl. Helper) bezeichnet. Ein Helferagent erlaubt einem Benutzer den Zugriff auf Ressourcen im Rahmen der definierten Verwendungsformen.

In bisherigen PAOSE-Projekten [WILLMOTT et al., 2005, CABAC et al., 2005] wurde in der Regel ein Benutzeragent verwendet, der einem Anwender des Systems erlaubte, Aktionen innerhalb des Systems vorzunehmen. Hierfür war es erforderlich, dass der Benutzeragent alle diese Interaktionsformen kennt, die ein Benutzer vornehmen kann und über die hierfür erforderlichen Protokolle verfügt. Durch die Verwendung von Helferagenten, die die Protokolle zur Verwendung von Ressourcen kennen, kann dieses Wissen aus dem Benutzeragenten ausgelagert und in einem hierauf spezialisierten Agenten gekapselt werden. Der Benutzeragent muss dann nur noch allgemeine Protokolle zum Umgang mit Helferagenten kennen. Dadurch kann neue Funktionalität auf einfache Weise zum System hinzugefügt werden, indem neue Helferagenten in das System eingebracht werden. Dies erlaubt eine modularere Gestaltung des Systems und einfachere Benutzeragenten.

Für das Whiteboard-Beispiel sind verschiedene Möglichkeiten der Modellierung möglich. Das Whiteboard selbst wird als ein Material modelliert, auf das durch verschiedene Helfer zugegriffen werden kann. Es wäre denkbar, separate Helfer für das Lesen und das Beschreiben des Whiteboards zu implementieren, da ein Stift als Werkzeug selbst nicht lesen kann. Die Werkzeugmetapher wird hier nicht exakt übernommen. Stattdessen wird ein Helfer verwendet, der die gesamte Interaktion eines Benutzers mit dem Whiteboard regelt.

Tatsächlich ist der Whiteboard-Helfer-Agent mehr als nur ein Stift, da er ebenfalls die Funktionalität zum Lesen des Whiteboards beinhaltet, sowie zur Vereinfachung auch die Möglichkeit, ein neues Whiteboard zu erstellen. Es wäre möglich, Lesen, Schreiben und Erzeugen auf unterschiedliche Werkzeuge aufzuteilen, darauf wird aber hier verzichtet, da es das Beispiel unnötig verkomplizieren würde.

6.1.4 Materialien

Materialien sind zuvorderst mehr oder weniger passive Objekte, die im Rahmen des Arbeitsprozesses durch Werkzeuge bearbeitet und schließlich in ein (End-)produkt überführt werden.

Im Grundmodell eines Multiagentensystems nach MULAN sind passive Entitäten nicht oder nur innerhalb der Wissensbasis eines Agenten vorgesehen. Materialien können daher auf verschiedene Arten implementiert werden. Sie können als spezielle Attribute der Plattform implementiert und über die Plattform verwaltet werden, als einfache Objekte innerhalb der Agenten, die über Nachrichten ausgetauscht werden oder als eigene Agenten modelliert werden,

die eigenständig die zulässigen Verwendungsformen überprüfen können. Im Folgenden werden verschiedene Varianten der Umsetzung diskutiert.

6.1.4.1 Verwaltung von Materialien durch die Agentenplattform

Da Materialien quasi per Definition keine Autonomie besitzen, sondern sich nur passiv verhalten, können sie als einfache Objekte modelliert werden. Die Plattform wacht dann darüber, wo sie sich befinden und wer welche Operationen mit ihnen ausführen kann. Diesen Ansatz verfolgen etwa [Omicini et al., 2008, Tell und Moldt, 2005], indem Artefakte als eigener Objekttyp in Multiagentensystemen neben Agenten eingeführt werden.

Der Vorteil dabei ist eine zentrale, einheitliche Behandlung. Der Ort, an dem sich ein Material aufhält, kann dann jederzeit durch die Plattform, auf der es sich befindet, bestimmt werden. Der größte Nachteil dieser Lösung ist, dass die Agentenplattform dafür um zusätzliche Funktionalitäten und Konzepte zur Verwaltung von Materialien erweitert werden müsste.

6.1.4.2 Materialien innerhalb von Agenten

Eine andere Möglichkeit besteht darin, Materialien innerhalb der Agenten zu verwalten. Die Agenten besitzen dann eine interne Repräsentation des Materials in ihrer Wissensbasis und können es wie jedes andere Objekt verwenden. Sollen Materialien zwischen Agenten ausgetauscht werden, kann dies über Nachrichten erfolgen.

Ein Beispiel für dieses Vorgehen stellt das Lehre-Projekt zur agentenorientierten Implementierung des Siedlerspiels [Willmott et al., 2005, Cabac et al., 2005] dar. Hier wurden Materialien (z.B. Baumaterial, Ereigniskarten und das Spielfeld) jeweils von einem Agenten verwaltet (der Bankagent für die Baumaterialstände aller Spieler und der Inselagent für den Zustand des Spielfeldes). Diese Materialien wurden allerdings nicht zwischen den Agenten verschoben, sondern immer zentral innerhalb eines Agenten vorgehalten. Lediglich Statusabfragen über Nachrichten wurden vorgenommen.

Der Vorteil dieses Ansatzes ist die einfache Benutzung von Materialien innerhalb eines Agenten. Nachteilig ist, dass Materialien als primäres Strukturierungsmerkmal an Bedeutung verlieren. Auch der Austausch über Nachrichten kann sich als unflexibel erweisen.

Das Beispiel des Whiteboards zeigt darüber hinaus, dass es Materialien gibt, die nicht direkt einem Agenten zugeordnet werden können, sondern von mehreren Agenten gleichzeitig verwendet werden können. Dies ist auch schwieriger zu realisieren, wenn das Material sich innerhalb eines der beteiligten Agenten befindet. In einem solchen Fall kann dann beispielsweise ein Agent als Verwalter des Materials auftreten, über den dann die Zugriffe anderer Agenten geregelt werden.

6.1.4.3 Materialien als Agenten

Nach dem Ansatz des „Alles ist ein Agent“ können auch Materialien als Agenten repräsentiert werden. Sie kapseln dann alle Informationen und Benutzungsweisen, die das Material ausmachen. Im Gegensatz zu einem normalen Agenten sind sie aber wesentlich weniger autonom, sondern an die Gesetzmäßigkeiten der Benutzung des Materials gebunden, ähneln also eher Objekten als Agenten.

Auf diese Weise kann der Aufenthaltsort eines Materials festgelegt werden als die Plattform, auf der sich der Agent befindet, dennoch können bei Bedarf mehrere Helferagenten darauf zugreifen, sofern der Materialagent dies erlaubt. Lässt ein Material nur die Bearbeitung durch einen Agenten zur Zeit zu, kann das vom Materialagenten durch entsprechende Protokolle selbst durchgesetzt werden.

6.1.4.4 Materialien und Plattformen

Die Modellierung von Materialien als Eigenschaften einer Plattform wie als Agenten hat als gemeinsames Merkmal, dass ein Material sich auf einer Plattform befindet. Auf dieser Plattform kann darauf zugegriffen werden und in der Regel nicht von außerhalb. Auf einer Plattform können aber alle Agenten gleichermaßen auf das Material einwirken.

Die Möglichkeit, Agenten als Plattformen zu betrachten und ineinander zu schachteln, erlaubt es, den wechselseitigen Ausschluss zu realisieren, indem das Material für den Schreibvorgang *in* den betreffenden Agenten verschoben wird, so dass die anderen Agenten keinen Zugriff mehr darauf haben, weil sie sich nicht mehr auf der gleichen Plattform befinden. Der betreffende Agent nimmt das Material also zur Bearbeitung an sich und gibt es hinterher wieder für die Öffentlichkeit heraus.

Diese Vorgehensweise setzt allerdings voraus, dass das Multiagentensystem es erlaubt, Agenten und Plattformen ineinander zu schachteln bzw. als austauschbare Konzepte zu verstehen. Entwicklungen in dieser Richtung finden sich in [Schleinzer, 2007]. Für HeRA wurde allerdings auf diese Vorgehensweise verzichtet, da die Funktionalität in Capa noch experimentell ist und in anderen Multiagentensystemen überhaupt nicht zur Verfügung steht.

6.1.4.5 Materialien in HeRA

Für die Implementierung von Materialien in HeRA muss eine der vorgestellten Möglichkeiten verwendet werden. Um eine einheitliche agentenorientierte Umgebung zu erzielen, wurde die Lösung gewählt, bei der Materialien als Agenten implementiert werden. In HeRA wird statt des Begriffs Material der allgemeinere Ausdruck Ressource verwendet. Wenngleich Ansätze zur Schachtelung von Plattformen in anderen Plattformen bzw. Agenten in Agenten existieren [Schleinzer, 2007], erlaubt der derzeitige Implementationsstand von Capa

die zuletzt vorgestellte Vorgehensweise nicht, bei der Ressourcenagenten sich in anderen Agenten befinden können.

Diese Architektur dient aber als Leitbild; die Einbettung von Agenten innerhalb anderer Agenten kann durch Agentenverhalten simuliert werden. Ein Ressourcenagent, der sich als „innerhalb“ eines Benutzer- oder Helferagenten betrachtet, kann auf Anfragen von anderen Helferagenten anders reagieren als einer, der frei auf einer Plattform liegt.

In vielen Fällen sind Materialien passive Systemeinheiten. Die Modellierung als Agenten und damit autonome, proaktive Einheiten, scheint daher auf den ersten Blick übertrieben komplex. Es gibt allerding auch komplexere Materialien, die selbsttätig über ihre Verwendung entscheiden können sollen, und die zusätzlich zu reinen Daten auch einen komplexeren Zugriffspfad auf diese Daten verwalten. Für diese Art von Materialien ist eine Darstellung als Agent sinnvoll. Die Modellierung von Werkzeugen und Materialien als Einheiten gleicher Art ergibt zudem einen einheitlicheren Systementwurf. Simplere Materialien, die nicht mehr leisten, als Daten zu kapseln, werden durch Agenten repräsentiert, die ebenfalls kein besonders hohes Ausmaß an Komplexität und Autonomie verfügen und eher einem Objekt ähneln.

In WAM werden viele verschiedene Arten von Materialien verwendet, die bestimmte Funktionen erfüllen können. Behälter-Materialien etwa können andere Materialien aufnehmen. Ein solcher Ressourcenagent kann die gleichen Protokolle verwenden, um andere Ressourcen aufzunehmen, wie ein Benutzer- oder Helferagent und so als Plattform fungieren.

Das Whiteboard ist als Ressource auf einer Kollaborationsplattform angesiedelt. Um auf das Whiteboard zugreifen zu können, muss ein Agent sich auf der Plattform befinden. Der Helferagent eines Benutzers würde hierfür zum Beispiel einen Proxyagenten auf der Kollaborationsplattform erzeugen.

Die Zugriffsregeln erlauben nun, bestimmte Operationen auf dem Whiteboard auszuführen, wie etwa beliebiges nebenläufiges Lesen, Schreiben aber nur im wechselseitigen Ausschluss. Diese Zugriffsregeln werden durch den Whiteboard-Ressourcenagenten durchgesetzt.

Prinzipiell können auf einem (physikalischen) Whiteboard mehrere Personen gleichzeitig schreiben, sofern sie über mehrere Stifte verfügen. In dieser Implementation wird dies über eine Queue geregelt, in die „Stift“-Agenten Schreibanforderungen eintragen können, die dann durch das Whiteboard abgehandelt werden.

6.1.5 Aspekte

Aspekte beschreiben ein Zugriffsmuster, das ein Material erlaubt, bzw. das ein Werkzeug beherrscht. Im agentenorientierten Kontext bedeutet das nichts weiter als die Kenntnis bestimmter Interaktionsprotokolle. Agenteninteraktionsprotokolle können verwendet werden, um einen Aspekt zu beschreiben. Die Agentenprotokolle von Helfer und Material bestimmen dann, wie genau der

Aspekt jeweils umgesetzt wird. In Abb. 6.1 entsprechen Aspekte den Pfeilen, die die Interaktion zwischen Helfer- und Ressourcenagenten darstellen.

Das Whiteboardmaterial implementiert die Aspekte „writable“, „readable“ und „observable“, während das Werkzeug entsprechend die Zugriffsmuster zum Lesen und Schreiben implementiert. Die Protokolle werden bewusst einfach gehalten. Lesen ist ein einfaches Query-Response-Protokoll. Beobachten bedeutet, dass das Werkzeug über Änderungen des Materialzustands (also des Whiteboardinhaltes) automatisch durch „inform“-Nachrichten unterrichtet wird. Für das Schreiben wird lediglich eine „request“-Nachricht mit dem neuen gewünschten Inhalt an das Whiteboard geschickt, das dann antwortet, ob die Änderung erfolgreich war oder nicht.

6.1.6 Automaten

Eine weitere Art von Konzepten aus dem WAM-Ansatz bilden die Automaten. Wie oben beschrieben, lassen sich diese in kleine und große Automaten unterteilen, die auf unterschiedliche Weise in Paose Verwendung finden können.

Kleine Automaten können problemlos durch Agenten abgebildet werden, die mit Eingabedaten versehen ihr internes Protokoll abarbeiten oder sogar als ein einzelnes Protokoll innerhalb eines Agenten. Teilweise können diese Automatenagenten auch ohne Benutzereingreifen aktiv werden. Beispiele wären Agenten, die automatisierte Buildvorgänge und Tests durchführen. Diese Art von Automaten findet man häufig bereits in Agentensystemen, zum Teil mit einem größeren Grad an Autonomie, als man das bei einem Automaten im strengeren Sinne erwarten würde.

Besondere Beachtung verdienen die großen Automaten, da diese eine Möglichkeit der Prozesssteuerung in WAM darstellen [Zülighoven, 2004]. Die Metapher hierfür ist die einer Produktionsstraße, die verschiedene Teilprozesse zu einem Gesamtprozess zusammenfasst. Eine solche Vergegenständlichung kann vorgenommen werden, indem Prozessbeschreibungen als Workflowagenten (vgl. [Markwardt et al., 2008b]) expliziert werden. Dies wird ausführlicher im Zusammenhang mit der agentenorientierten Prozessinfrastruktur in Kapitel 7 erörtert. Die dort vorgestellte Prozessinfrastruktur ist ein Beispiel für einen Automaten, der als ein Subsystem aus mehreren interagierenden Agenten implementiert wird.

6.1.7 Arbeitsplätze

Ein Arbeitsplatz ist ein Ort, an dem sich Werkzeuge, Materialien und Automaten zum Zugriff durch einen Benutzer befinden. Dies impliziert ein Ortskonzept, das in Multiagentensystemen durch Agentenplattformen modelliert werden kann.

Zusätzlich ist der Arbeitsplatz dadurch gekennzeichnet, dass er einem Benutzer zugeordnet ist, der über die dort vorhandenen Gegenstände verfügen

kann. Hierfür wird ein neuer Typ von Agent eingeführt, der Benutzeragent (User Agent oder UA). Dieser stellt den zentralen Zugriffspunkt eines Benutzers zum System dar und verwaltet die Helfer und Ressourcen, über die dieser verfügt. Der Benutzeragent und die lokale Agentenplattform eines Benutzers bilden damit den Arbeitsplatz eine Benutzers im System.

Nachdem ein Benutzer einen neuen Whiteboard-Helfer zu seinem Arbeitsplatz hinzugefügt hat, öffnet dieser ein Fenster in der GUI des Benutzers und präsentiert ihm eine Liste der verfügbaren Whiteboards, die er finden kann. Der Benutzer kann dann entscheiden, diese zu beobachten, zu bearbeiten oder ein neues zu erstellen. Im Beispiel des Whiteboards ist das Material nicht auf dem lokalen Arbeitsplatz des Benutzers sondern auf einer beliebigen Agentenplattform in der Arbeitsumgebung.

6.1.8 Arbeitsumgebung

Die Arbeitsumgebung eines Benutzers stellt den weiteren Kontext seiner Arbeit dar, die über seinen individuellen Arbeitsplatz hinausgeht. Dies beschreibt weitere Orte, Helfer und Ressourcen, auf die der Benutzer zugreifen kann, sowie andere Benutzer im System und ihre Arbeitsplätze.

Die Arbeitsumgebung eines Benutzers besteht aus allen am Multiagentensystem beteiligten Agentenplattformen. Über entsprechende Helfer kann er mit Benutzern auf anderen Plattformen interagieren, Gegenstände mit anderen Plattformen austauschen und auf Dienste zugreifen, die von anderen Agenten innerhalb des Multiagentensystems angeboten werden.

Zusätzlich können in der Arbeitsumgebung virtuelle Kollaborationsräume eingerichtet werden. In diesen Räumen können sich Benutzer treffen, um Gegenstände gemeinsam zu nutzen. Durch die generelle Nachrichtenkommunikation zwischen Agenten kann sich ein Benutzer sowohl auf seiner lokalen Plattform (Arbeitsplatz) befinden, als auch in verschiedenen virtuellen Räumen.

Die Whiteboardressource wird als Koordinationsmittel auf einer eigenen Kollaborationsplattform erstellt. Die Whiteboard-Helfer-Agenten können auf die Ressource dort zugreifen. Der Ressourcenagent könnte auch entscheiden, den Zugriff auf Helferagenten zu beschränken, die sich auf der gleichen Plattform befinden.

6.1.9 Kooperation

Es gibt viele Möglichkeiten, die Kooperation zwischen Benutzern eines Multiagentensystem zu unterstützen. In der Regel wird dies über geeignete Helfer- und Ressourcenagenten realisiert werden.

Beispiele für solche Kooperationshelfer finden sich in Kapitel 8. Hier werden ein Chat-Agent und das bereits erwähnte Whiteboard-System als Beispiel für Kooperationsmittel vorgestellt.

Ein zentrales Thema dieser Arbeit ist aber auch die Entwicklung einer agentenorientierten Prozessinfrastruktur. Hier wird eher der Ansatz eines Steuerungsautomaten verwendet. Die Verwendung eines Workflowagenten als Kooperationsmaterial erlaubt eine höhere Flexibilität, als das üblicherweise in Workflow Management Systemen der Fall ist [Reese et al., 2005]. Dies wird in Kapitel 7 ausführlicher diskutiert.

6.2 Anforderungen

Die generellen Ideen zu einem Helfer- und Materialkonzept werden im Folgenden in konkrete Anforderungen umgesetzt, aus denen sich dann ein Entwurf ableiten lässt. Diese Anforderungen geben an, was das HeRA-System aus Sicht eines Entwicklers, wie auch eines Benutzers einer mit HeRA realisierten Anwendung leisten muss.

6.2.1 Benutzeragent

Der Benutzeragent stellt den Arbeitsplatz eines Anwenders dar, der mit dem HeRA-System arbeitet. Er stellt auf der einen Seite eine Benutzungsoberfläche zur Verfügung, durch die auf das Multiagentensystem zugegriffen werden kann. Auf der anderen Seite ist er selbst Teil des Multiagentensystem und verfügt über Protokolle für die Verwaltung von Helfer- und Ressourcenagenten.

Die Benutzungsoberfläche muss es dem Anwender erlauben, einen Überblick über die im System und auf dem Arbeitsplatz verfügbaren Helfer zu erlangen. Sie stellt Funktionen zum Auflisten der verfügbaren Helfer bereit, die die Helferfabrik bereitstellt, und erlaubt es, diese anzufordern. Zusätzlich bietet sie die Flexibilität, Helferagenten mit ihrer GUI in die Haupt-GUI des Benutzeragenten zu integrieren. Auf diese Weise können Helfer den Benutzeragenten um neue Funktionen erweitern. Die Art der Erweiterung ist natürlich abhängig von der Implementierung des Benutzeragenten, die Helferagenten müssen daher auf diesen abgestimmt sein.

Für die Interaktion mit dem Multiagentensystem verfügt der Benutzeragent über eine Reihe von Protokollen: Er lokalisiert selbstständig die Helferfabrik im System und fragt eine Liste der verfügbaren Helferagenten an. Auf Anwenderanfrage wird ein Protokoll zum Abrufen eines neuen Helferagenten aus dieser Liste gestartet. Nach der Erzeugung dieses Helfers durch die Helferfabrik erfolgt dessen Registrierung beim Benutzeragenten. Dabei werden auch die vom Helferagenten bereitgestellten GUI-Komponenten in die GUI des Benutzeragenten eingefügt. Wenn nötig werden die dafür notwendigen Plugins über das Netzwerk nachgeladen.

6.2.2 Helferfabrik

Eines der Ziele bei der Verwendung von Helferagenten für die Implementierung der Funktionen im HeRA-System ist die einfache Erweiterbarkeit. Zu diesem Zweck erfolgt die Erzeugung neuer Helferagenten durch einen spezialisierten Agenten, die Helferfabrik. Auf diese Weise können neue Funktionen an zentraler Stelle durch Erweiterung dieses einen Agenten hinzugefügt werden.

Die Helferfabrik muss ein Verzeichnis von Helferagenten verwalten und dieses Verzeichnis interessierten Benutzeragenten auf Nachfrage mitteilen. Wenn ein Benutzeragent einen neuen Helfer anfordert, ist es Aufgabe der Helferfabrik, dafür zu sorgen, dass der Helferagent erzeugt wird. Dafür muss er dem Plattformagenten alle nötigen Informationen für die Erzeugung eines neuen Agenten übermitteln. Er sorgt ferner dafür, dass der neu erzeugte Helferagent sich beim anfordernden Benutzeragenten registriert.

Die Helferfabrik kann, sofern das erforderlich ist, Modifikationen und Voreinstellungen an den Helferagenten vornehmen, beispielsweise kann ein Helfer mit der Adresse eines benötigten Serveragenten für eine bestimmte Funktion konfiguriert werden. In Verbindung mit der Prozessinfrastruktur wird dies verwendet, um Helferagenten im Workflow mit den Informationen vorzukonfigurieren, die für die Bearbeitung einer bestimmten Aufgabe erforderlich sind.

Nicht alle Benutzer in einem verteilten System haben notwendigerweise Zugriff auf alle Funktionen. Die Helferfabrik muss daher auch die Zugriffsberechtigung für die erzeugten Helfer kontrollieren. Auch kann die Helferfabrik die erzeugten Helferagenten in Abhängigkeit der Benutzerrolle unterschiedlich konfigurieren. Ein Helferagent, der für einen Benutzer mit Administrationsberechtigung erzeugt wurde, verfügt dann über zusätzliche Funktionen und Protokolle, die ein Helferagent für einen weniger privilegierten Anwender nicht besitzt. Da Sicherheitsaspekte im Rahmen dieser Arbeit zunächst ausgeklammert werden, wird dieser Aspekt allerdings im Weiteren nicht ausgeführt.

6.2.3 Helfer

Helferagenten stellen Funktionalität im HeRA-System zur Verfügung. Daher sind hier sehr verschiedene Implementationen möglich. Alle Helferagenten müssen jedoch über die grundlegenden Protokolle zum Registrieren bei einem Benutzeragenten verfügen. Zusätzlich enthält HeRA einfache Basisprotokolle für die Erzeugung, Beobachtung und Modifikation von Materialien. Diese müssen im Einzelfall an die spezielle Situation angepasst oder ersetzt werden.

Für die Integration der Benutzungsschnittstelle eines Helfers in die des Benutzeragenten ist eine Schnittstelle erforderlich, die zwischen den Entwicklern des Benutzeragenten und der Helferagenten abgestimmt sein muss. Bei der Registrierung eines Helfers beim Benutzeragenten stellt der Helferagent die Informationen bereit, die benötigt werden, um seine GUI in die des Benutzeragenten zu integrieren. Entwickler, die Anwendungen auf der Basis des

HeRA-Systems entwickeln, können diese Schnittstelle verwenden, um GUIs für ihre Helfer zu erstellen.

6.2.4 Ressourcenagenten

Ressourcenagenten kapseln anwendungsspezifische Daten und Ressourcen, wie etwa Konzepte, Dokumente, Datenbanken oder Dateien. Der genaue Aufbau eines Ressourcenagenten ist daher Sache des Anwendungsentwicklers. Das HeRA-System muss lediglich Protokolle zur Erzeugung von Ressourcenagenten bereitstellen und auch diese sind abhängig von den dafür verwendeten Helferagenten. In der Regel besitzt ein Ressourcenagent auch keine eigene GUI-Repräsentation, die Darstellung eines Materials ist Aufgabe der zugehörigen Helferagenten.

Ressourcen werden regelmäßig zwischen verschiedenen Benutzern und Helfern ausgetauscht. Daher ist es nötig, dass Ressourcenagenten in der Lage sind, ihren Aufenthaltsort zu ändern. Die Capa-Plattform stellt dazu Migrationsprotokolle zur Verfügung, die verwendet werden können, um einen Agenten von einer Plattform auf eine andere zu migrieren. Oft ist es allerdings nicht nötig, den Agenten zu migrieren, da er auch von anderen Plattformen aus über Agentennachrichten erreicht werden kann und der tatsächliche Aufenthaltsort des Agenten damit nicht entscheidend ist.

6.2.5 Plattformen

Agenten im HeRA-System residieren auf Agentenplattformen, die miteinander über ein Netzwerk zu einem Multiagentensystem verbunden sind. Plattformen bieten damit eine Möglichkeit zur Strukturierung des Systems. So kann ein Anwender seine eigene Agentenplattform in das Multiagentensystem einbringen, die seinen Arbeitsplatz repräsentiert und auf der sich der Benutzeragent sowie die verbundenen Helfer- und Ressourcenagenten befinden.

Zusätzlich können weitere Subsysteme durch eigene Agentenplattformen (physikalisch oder virtuell als Teil einer anderen Agentenplattform) im System repräsentiert werden. Beispiele hierfür wären Kollaborationsplattformen, auf denen sich Agenten treffen können, um an einem gemeinsamen Projekt zu arbeiten und dort auch Projektressourcen abzulegen, oder Subsysteme, wie etwa ein Workflow Management System, die auf einer eigenen Plattform operieren.

6.3 Entwurf - Agenten als Werkzeuge

Aus den oben formulierten Anforderungen wird jetzt ein Entwurf erstellt, der eine konkrete Implementierung des HeRA-Systems erlaubt. Darin werden die beteiligten Agenten/Agentenrollen, Interaktionen und die Ontologie beschrie-

	Benutzeragent	Helferagent	Ressourcenagent	Helferfabrik	Sonstige
Helferagenten auflisten	X			X	DF
Helferagent erzeugen	X	X		X	AMS
Helferagent registrieren	X	X		(X)	
Helfer-/Benutzeragenten-Verbindung		X			
Material erzeugen	(X)	X	X		AMS
Material beobachten	(X)	X	X		
Material bearbeiten	(X)	X	X		

Tabelle 6.1: Rollen und Interaktionen der Werkzeugumgebung

ben. Die konkrete prototypische Umsetzung dieses Entwurfs wird dann in Kapitel 8 beschrieben.

Wie in Abschnitt 4.3 beschrieben, ist eine Entwicklungsumgebung eine Sammlung verschiedener Werkzeuge zur Unterstützung des Software-Entwicklungsprozesses. Das HeRA-System bietet die Möglichkeit, solche Werkzeuge und die damit zu bearbeitenden Ressourcen als Agenten zu definieren und damit die Grundlagen für die Erstellung eines agentenorientierten IDE. Die Grundzüge dieses Systems finden sich in [Lehmann und Markwardt, 2004, Lehmann et al., 2005].

Die Funktionen innerhalb des HeRA-Systems sollen durch Agenten erbracht werden. Dafür sind zunächst drei Arten von Agenten erforderlich: Eine Arbeitsumgebung für Benutzer, sowie Arbeitsmittel und Ressourcen, die darin eingebunden werden können. Diese sollen realisiert werden als Benutzeragenten und Helfer- sowie Ressourcenagenten. Die Agenten, sowie die Interaktionen, an denen sie jeweils beteiligt sind, sind in Tabelle 6.1 im Überblick dargestellt. Diese Übersicht kann als Basis für eine Matrixorganisation des Entwicklungsteams bei einem Paose-Vorgehen dienen.

6.3.1 Rollen

Das HeRA-System ist ein Agentensystem, in dem Benutzern durch verschiedene kooperierende Agenten die Funktionalität zur Bearbeitung von Arbeitsgegenständen ermöglicht wird. Der Benutzer wird in diesem System repräsentiert durch einen Benutzeragenten. Dieser befindet sich auf einer Agentenplattform und stellt zusammen mit den verbundenen Helfer- und Ressourcenagenten den Arbeitsplatz des Benutzers dar.

Der Benutzeragent kooperiert mit Helfer- und Ressourcenagenten, die Hilfsmittel und Arbeitsgegenstände repräsentieren. Die Zusammenstellung der Helfer- und Ressourcenagenten eines Benutzers bestimmt, was dieser an seinem Arbeitsplatz tun kann.

Der Benutzeragent erschließt dem Benutzer die Funktionalität des Systems und gibt die Anforderungen des Benutzers an das System weiter. Der Benutzeragent stellt daher auch eine Benutzungsoberfläche zur Verfügung, die die Helferagenten um die ihnen jeweils eigenen Funktionen erweitern können.

Für die Benutzungsoberfläche wurde entschieden, eine RCP-Anwendung zu verwenden (siehe auch Abschnitt 5.6.2.3), die direkt oder über Remote Method Invocation (RMI) mit dem Agentensystem in Verbindung steht. Über den Plugin-Mechanismus von Eclipse können GUI-Erweiterungen von Helferagenten eingebunden werden. Die Benutzungsoberfläche des Agenten stellt seinen Arbeitsplatz dar, auf dem sich die aktuell verwendeten Helfer und Ressourcen befinden.

Die verschiedenen Funktionalitäten, die dem Benutzer zur Verfügung stehen, werden durch verschiedene Helferagenten erbracht. Diese Agenten können bei Bedarf zur Erbringung ihrer jeweiligen Dienstleistung erzeugt werden. Um die Helferagenten zu erzeugen, gibt es den sogenannten Helferfabrikagenten. Benutzer sollen die Möglichkeit haben, neue verfügbare Helfer zu finden und in ihren Arbeitsplatz zu laden. Um das zu ermöglichen, wird die Rolle der Helferfabrik eingeführt, die verantwortlich ist für die Verwaltung von Helferagentendefinitionen und auf Anfrage für die Erstellung neuer Helferagenten sorgt.

Es wird ein generischer Helferagent implementiert, der die nötigen Funktionen umfasst, um erzeugt und in einen Benutzeragenten eingebunden werden zu können. Spezifische Helfer erweitern diesen dann um eigene Funktionalität. Insbesondere kann ein Helferagent eine Benutzungsoberfläche bereitstellen, die in diejenige des Benutzeragenten eingefügt werden kann.

Helfer operieren auf Ressourcen, insbesondere Materialien, die Artefakte des Arbeitsprozesses darstellen. Im Falle einer Softwareentwicklungsumgebung sind das unter anderem Pflichtenhefte, Quellcode, Testpläne oder auslieferungsfähige Software. Andere Arten von Ressourcen können etwa Hardwareressourcen (Drucker, Scanner), Dateien, Datenbanken etc. sein. Um eine hohe Flexibilität bei der Gestaltung von Ressourcen zu erhalten, und um innerhalb der Metapher konsistent zu bleiben, werden Ressourcen ebenfalls als Agenten modelliert. Ressourcenagenten können von Helferagenten erzeugt und bearbeitet werden und zwischen Helfern und Benutzern ausgetauscht werden.

6.3.1.1 Benutzeragent

Der Benutzeragent ist die Schnittstelle eines Benutzers zum Agentensystem. Daher besteht er aus zwei wesentlichen Teilen, einer Benutzungsoberfläche und einem Petrinetzagenten. Diese stehen miteinander in Verbindung, so dass die

Aktionen des Benutzers Funktionen innerhalb des Netzagenten auslösen können. Umgekehrt können Aktionen innerhalb des Netzagenten, wie etwa eingehende Nachrichten von anderen Agenten, in der Benutzungsoberfläche angezeigt werden.

Benutzungsoberfläche Zentrale Aufgabe des Benutzeragenten ist es, einem menschlichen Benutzer Zugriff auf das Agentensystem zu ermöglichen. Dafür wird eine Benutzungsschnittstelle zu Verfügung gestellt.

Prinzipiell sind verschiedene Arten von Benutzungsschnittstellen möglich. Hier muss eine Designentscheidung gefällt werden, da die Benutzungsschnittstellen der Helferagenten später hierzu kompatibel sein müssen. Denkbar sind natürlich auch verschiedene Implementationen, etwa eine Java-basierte und eine Web-Schnittstelle, dann muss aber jeder Helferagent mehrere verschiedene GUIs bereitstellen.

In dieser Arbeit wurde die Implementierung als Plugin in der Entwicklungsumgebung Eclipse [Eclipse, 2011] gewählt, da so für die normale Einzelplatzentwicklung der volle Leistungsumfang von Eclipse genutzt werden kann. Mit der Rich Client Platform (RCP) [RCP, 2011] steht zudem eine flexibel erweiterbare Plattform zur Verfügung, die für technisch nicht so versierte Benutzer (z.B. Kunden, die bestimmte Aspekte der Software testen sollen) minimalistisch eingerichtet werden kann und dann nur die benötigten Funktionen enthält.

Die Benutzungsschnittstelle muss zwei grundlegende Funktionalitäten erfüllen. Zum einen zeigt sie die im System angebotenen Helfertypen an, und erlaubt es, Helfer aus dieser Liste auszuwählen, um sie in den Benutzeragenten zu integrieren. In dem in Kapitel 8 vorgestellten Prototypen wird hierfür ein View verwendet, der eine einfache Liste von Helferagenten anzeigt, aus denen dann per Doppelklick die gewünschten Helfer ausgewählt werden können.

Zum anderen muss die Benutzungsoberfläche so flexibel gestaltet sein, dass die Helferagenten ihre eigenen GUI-Bestandteile integrieren können. Dies wird dadurch erreicht, dass der Helferagent ein Plugin angibt, das über OSGi [OSGi, 2011] in das Eclipse Pluginsystem geladen wird, und eine Klasse, die das `IHelperAgent` Interface implementiert. Auf diese Weise kann der Helfer beliebige Initialisierungen vornehmen und auf Daten in der Umgebung des Benutzeragenten zugreifen.

An dieser Stelle muss noch einmal darauf hingewiesen werden, dass diese Arbeit Sicherheitsaspekte nicht behandelt. Das Importieren und die Ausführung von Code, der von anderer Stelle geladen wird, birgt potentielle Risiken. Strategien, wie diesen Risiken begegnet werden kann, werden hier nicht diskutiert.

Netzagent Die Benutzungsoberfläche ist mit dem Multiagentensystem durch einen Netzagenten verbunden, der den Benutzer als Agenten repräsen-

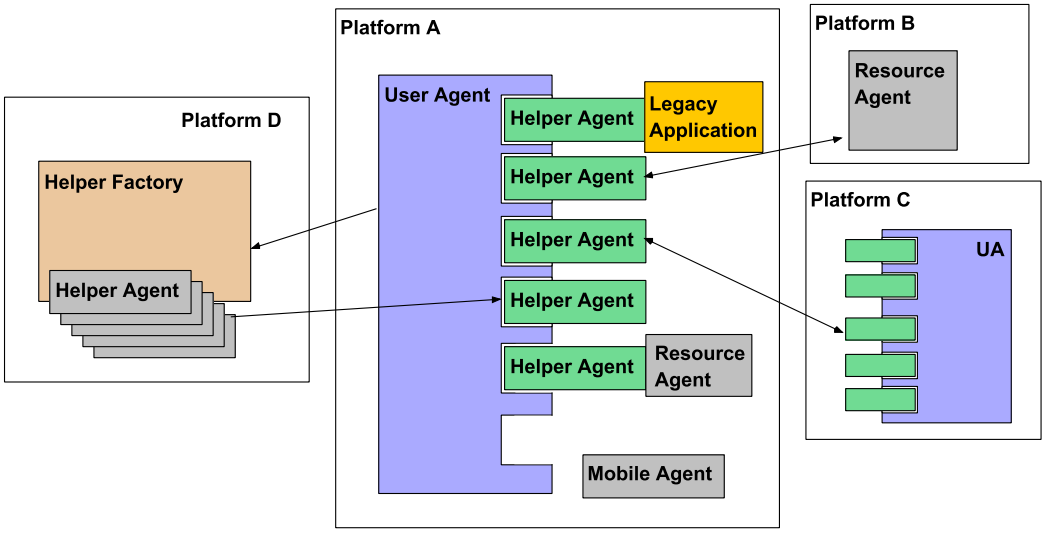

Abbildung 6.2: Helferagenten

tiert. Die Kommunikation zwischen diesen beiden Teilen des Benutzeragenten erfolgt durch das `remoteDC`-Plugin, das es erlaubt, Agenten in einem MULAN/CAPA Agentensystem entfernt über RMI [RMI, 2011] zu steuern (ausführlicher beschrieben in Abschnitt 8.2.2.1).

Der Netzagent enthält die eigentliche Funktionalität des Agenten. Er verwaltet das Wissen des Agenten in seiner Wissensbasis, wie etwa die verbundenen Helfer- und Ressourcenagenten oder die Adresse der Helferfabrik und führt die Agentenprotokolle zum Laden und Registrieren von Helferagenten aus. Über die `remoteDC` kann die GUI auf die Werte in der Wissensbasis zugreifen, um sie anzuzeigen, oder auch neue Protokolle starten.

6.3.1.2 Helferagent

Der Helferagent ist dafür zuständig, die im System anfallenden Aufgaben tatsächlich auszuführen. Helferagenten werden bei Bedarf durch den Benutzeragenten angefordert und durch die Helferfabrik erzeugt und, wenn erforderlich, auf die Plattform des Benutzeragenten migriert. Sie können die Benutzungsoberfläche des Benutzeragenten um eigene Elemente erweitern.

Helferagenten können auf verschiedene Arten operieren, wie in Abb. 6.2 dargestellt wird. Meist dienen Helfer dazu, Materialien zu bearbeiten, die sich auf der Benutzerplattform befinden. Hierfür müssen sie mit den Protokollen der jeweiligen Materialien vertraut sein, so dass Helfer und Materialien über eine vereinbarte Schnittstelle aufeinander abgestimmt sein müssen. Da Agenten untereinander mit Nachrichten kommunizieren, ist es natürlich auch möglich, dass sich eine bearbeitete Ressource auf einer anderen Agentenplattform befindet. Dies ist für den Benutzer in der Regel transparent.

Helferagenten können Ressourcenagenten nicht nur anzeigen und bearbeiten, sie werden auch verwendet, um neue Ressourcenagenten zu erzeugen. Hierfür muss der Helferagent natürlich, ähnlich wie die Helferfabrik, mit den Details für die Erzeugung eines Ressourcenagenten vertraut sein.

Ein Helferagent kann aber auch ohne Material eine für den Benutzer nützliche Dienstleistung erbringen. Er kann eine vorhandene Legacy Applikation kapseln und so Zugriff darauf in der Anwendung ermöglichen. Der Chat-Agent aus Abschnitt 8.2 erbringt seine Funktionalität, indem er direkt mit Helferagenten kommuniziert, die mit anderen Benutzern verbunden sind.

Helferagenten können den Benutzeragenten um eigene GUI-Elemente erweitern. Hierfür übermitteln sie bei der Registrierung ein `RegisterHelper`-Objekt an den Benutzeragenten. Darin ist neben dem `AgentIdentifier` des Helferagenten auch eine Beschreibung seiner Benutzungsoberfläche. Für den allgemeinen, vom GUI-Framework unabhängigen, Fall enthält dieses `HelperInterface` einen Namen und ein Passwort für den Zugriff auf den Agenten via `remoteDC`.

Für die Verwendung einer RCP-GUI wird eine Unterklasse verwendet, `RCPHelperInterface`. Objekte dieser Klasse enthalten zusätzlich eine `PluginURL` und eine `HelperAgentClass`. Die `PluginURL` wird verwendet, um das benötigte Eclipse-Plugin in die Benutzungsoberfläche zu laden. Hierfür verwendet Eclipse die OSGi-Plattform [OSGi, 2011]. Anschließend wird ein Objekt der als `HelperAgentClass` angegebenen Klasse erzeugt, die das Interface `IHelperAgent` implementieren muss. Daran wird die `start()`-Methode aufgerufen, um den Helferagenten zu initialisieren. Dieses Vorgehen erlaubt dem Helferagenten, dem Benutzeragenten quasi beliebige Funktionen hinzuzufügen.

6.3.1.3 Helferfabrik

Der Helferfabrikagent verwaltet eine Liste möglicher Helferagenten, die auf Benutzeranfrage hin instanziiert werden können. In einem größeren Projekt kann es mehrere Helferfabriken geben, beispielsweise lokal auf dem Rechner eines Benutzers für Standardhelfer und spezialisierte Versionen für das spezielle Projekt. Wenn verschiedene Organisationen für ein Projekt kollaborieren, können bestimmte Helfer allen Projektbeteiligten angeboten werden, während andere organisationsintern sind. Zusammen mit einem Workflow können beispielsweise die Agenten für die verschiedenen Workflowaktivitäten in einer eigenen Helferfabrik bereitgestellt werden. Die Verwendung von Helferagenten in WfMS wird in Kapitel 7 ausführlicher behandelt.

Wenn ein Benutzeragent einen neuen Helfer anfordert, wird zunächst auf der Agentenplattform der Helferfabrik ein neuer Agent erzeugt und mit passenden Parametern initialisiert. Auf diese Weise hat die Helferfabrik einen großen Einfluss auf die genaue Ausgestaltung der erzeugten Helferagenten. Bei der Verwendung im WfMS etwa werden Helferagenten für die zu bearbeitenden Workflowaufgaben vorkonfiguriert. Ebenso ist es möglich, die Helferagenten auf

der Basis der Benutzerrollen und -berechtigungen zu konfigurieren. Wenn ein Benutzer beispielsweise eine bestimmte Funktion nicht ausführen darf, dann können die entsprechenden Protokolle komplett aus dem Helferagenten entfernt werden.

Falls erforderlich kann der Helferagent dann auf die Plattform des Benutzeragenten migrieren. Anschließend sorgt die Helferfabrik dafür, dass der Helferagent ordnungsgemäß mit dem Benutzer verbunden wird.

6.3.1.4 Ressourcenagent

Ein Ressourcenagent kapselt eine Ressource, auf die von Helferagenten zugegriffen werden kann. Insbesondere umfasst das Materialien, die durch Helfer bearbeitet, untersucht und umgeformt werden können. Er verhält sich dabei eher wie ein Objekt als wie ein Agent, d.h. die Autonomie eines Ressourcenagenten ist stark eingeschränkt.

Für den Zugriff auf eine Ressource stehen jeweils eigene Protokolle zur Verfügung, die Ressourcen- und Helferagent kennen müssen. Diese Protokolle regeln die zulässigen Interaktionen mit der Ressource.

Ein Ressourcenagent kann Ressourcen außerhalb des Agentensystems referenzieren, wie beispielsweise Dateien oder Datenbankverbindungen, er kann aber auch lediglich durch den Zustand seiner Wissensbasis charakterisiert sein.

6.3.2 Interaktionen

Die verschiedenen Agenten des HeRA-Systems stehen miteinander über verschiedene Protokolle in Verbindung, um ihre Aufgaben zu erfüllen. Dies umfasst insbesondere das Auffinden, Erstellen und die Verwendung von Helfer- und Ressourcenagenten.

Der Vorgang für das Erzeugen eines neuen Helferagenten besteht aus mehreren Teilen, die Abb. 6.3 zeigt. Außer der tatsächlichen Erzeugung des Helferagenten gehört auch die Registrierung beim Benutzeragenten mit dazu, da beides in HeRA in einem Schritt durchgeführt wird. Bei der Einführung von Aktivitätsagenten in Kapitel 7 wird dies aber getrennt, so dass Erzeugung und Registrierung als zwei unterschiedliche Interaktionen betrachtet werden können.

6.3.2.1 Verfügbare Helferagenten finden

Bevor der Benutzeragent einen Helferagenten anfordern kann, muss er zunächst in Erfahrung bringen, welche Helfer im System vorhanden sind. Die Protokollnetze des Benutzer- und Helferfabrikagenten zeigt Abb. 6.4 Dafür fragt er den allgemeinen Verzeichnisdienst der Plattform (Directory Facilitator = DF), welche Agenten im System einen Helferfabrikdienst anbieten. Der Plattform-DF

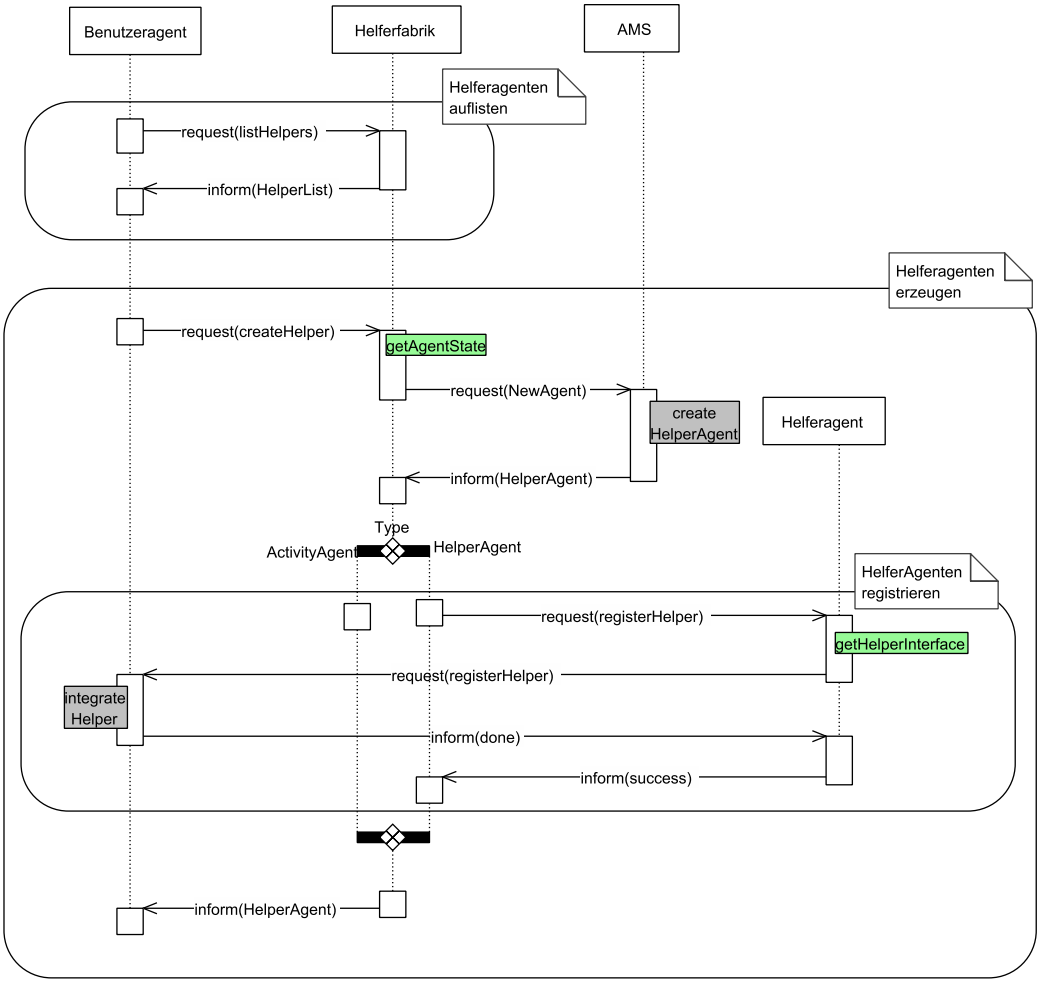

Abbildung 6.3: Interaktion: Helferagenten auflisten, erzeugen und registrieren

kann andere DF-Agenten innerhalb des Systems kontaktieren, um weitere Helferfabrikagenten zu lokalisieren.

Bei diesen fragt der Benutzeragent dann an, welche Helfer sie zur Verfügung stellen. Unter Umständen muss dazu die Helferfabrik ein Unterprotokoll aufrufen, um die Berechtigungen des Benutzers zu prüfen, falls nicht alle Helfer allgemein freigegeben sind. Auf diesen Schritt wird im Prototypen verzichtet. Die Anfrage kann Einschränkungen und Suchbegriffe enthalten, um nur bestimmte Typen auflisten zu lassen. Die Helferfabrik erzeugt dann eine Liste der infrage kommenden Helfer und sendet sie zurück an den Benutzeragenten.

6.3.2.2 Neuen Helfer erzeugen

Um einen neuen Helfer zu erzeugen, wendet sich der Benutzeragent an eine Helferfabrik, die den gewünschten Helfertyp bereitstellt. Diese sorgt dafür, dass der Helferagent auf der Plattform erzeugt und mit den richtigen Parametern

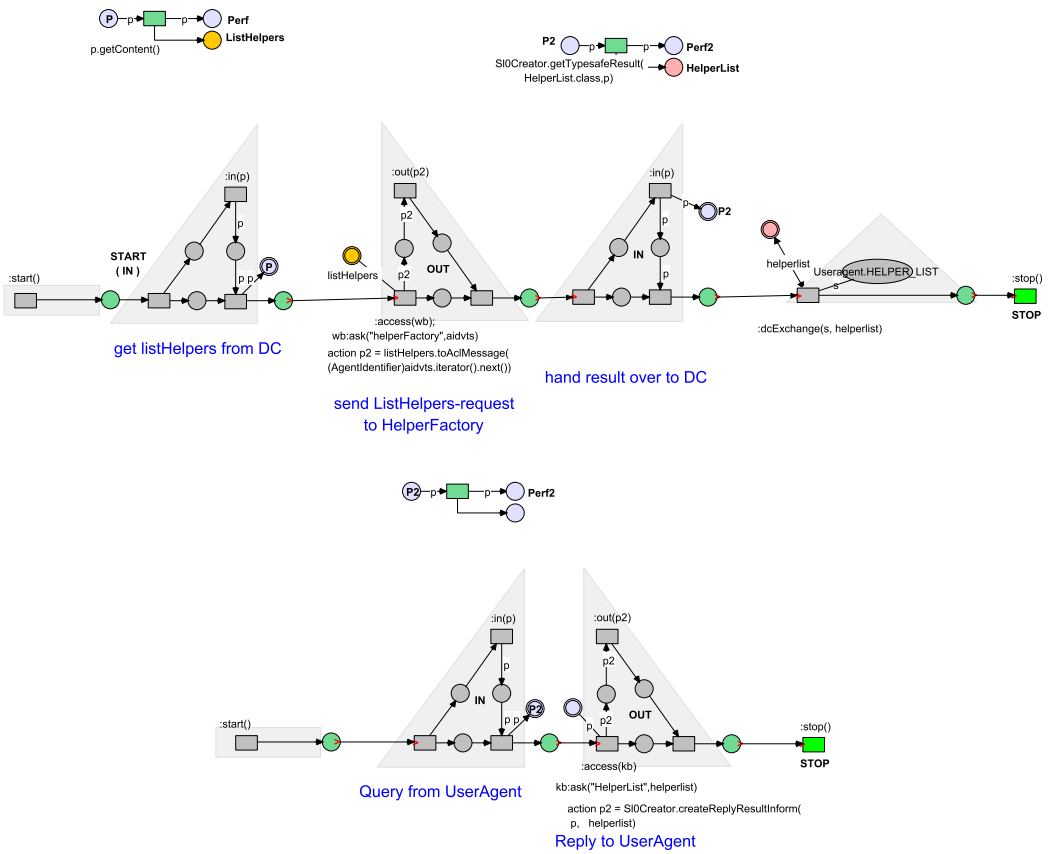

Abbildung 6.4: Protokollnetze des Benutzeragenten und der Helferfabrik zum Auflisten verfügbarer Helferagenten

initialisiert wird.

Für die Erzeugung eines Agenten wird der Agent Management System Agent (AMS Agent) der Plattform aufgerufen. Diesem wird der Name und die Wissensbasis des neuen Agenten übergeben, woraus er dann den neuen Agenten erzeugt. Die Helferfabrik verfügt daher über Vorlagen, wie die Wissensbasen der jeweiligen Helferagenten aussehen. Optional können über Parameter, die an die Helferfabrik übergeben werden, noch Modifikationen an dieser Vorlage vorgenommen werden, um den Helfer an die speziellen Anforderungen anzupassen.

Nun kann es unter Umständen erforderlich sein, den neu erzeugten Helferagenten auf die Agentenplattform des Benutzers zu migrieren. Hierfür werden die entsprechenden Protokolle der CAPA-Plattform verwendet. Abschließend fordert die Helferfabrik den neuen Helferagenten auf, sich beim Benutzeragenten zu registrieren und quittiert die Erzeugung des neuen Helferagenten.

6.3.2.3 Helferagenten registrieren

In der Regel wird ein Helferagent nach der Erzeugung einem Benutzeragenten zugewiesen. Hierfür erhält er, üblicherweise von der Helferfabrik, die Aufforderung, sich bei einem bestimten Benutzeragenten zu registrieren. Diesem sendet er eine Beschreibung seiner Benutzungsschnittstelle, die dann, wie oben beschrieben, in die Benutzungsschnittstelle des Benutzeragenten integriert wird.

6.3.2.4 Helfer-/Benutzeragenten-Zuordnung

Sämtliche Funktionalität, auf die ein Benutzer zugreifen kann, wird durch Helferagenten realisiert. Wenn daher zwei Benutzer miteinander interagieren, geschieht dies durch ihre Helferagenten. Die Interaktion `GetUserAgentMapping` ermöglicht es einem Helferagenten, den Benutzer zu ermitteln, mit dem er interagiert, beispielsweise um den Benutzer anzuzeigen, mit dem eine Chat-Konversation geführt wird. Dies ist eine einfache Query-Response-Interaktion zwischen zwei Helferagenten.

6.3.2.5 Neuen Ressourcenagenten erzeugen

Ressourcenagenten werden in der Regel durch Helferagenten erzeugt. Bei der Erzeugung eines Helferagenten stellt die Helferfabrik die Wissensbasis des Helferagenten zur Verfügung. Analog dazu erzeugt hier der Helferagent die Wissensbasis des Ressourcenagenten und sendet sie an den AMS Agenten zur Erzeugung der neuen Ressource. Die neue Ressource ist initial ungebunden, der Helfer kann sich aber direkt als Beobachter oder Bearbeiter registrieren.

6.3.2.6 Material beobachten

Helfer dienen dazu, Materialien zu untersuchen und zu bearbeiten. Ein Helfer kann sich bei einem Materialagenten (und bestimmten anderen Ressourcenagenten) als Beobachter registrieren, um über Aktualisierungen des Materialzustandes auf dem Laufenden gehalten zu werden. Dies kann durch eigene Manipulationen, durch interne Prozesse des Materials oder durch andere Helferagenten hervorgerufen werden, die Zugriff auf das Material haben.

Der Helferagent sendet hierfür eine `request-whenever(MaterialChange)`-Nachricht an das Material. Der Materialagent speichert dann den Helfer in seiner Liste von zu benachrichtigenden Agenten für den Fall einer Änderung.

6.3.2.7 Materialänderung

Wenn ein Material geändert wird und eine Beobachtung zulässt, wird dieses Protokoll aufgerufen, das an alle eingetragenen Beobachter ein Update des Zustandes versendet. Was die Beobachter daraufhin tun, ist natürlich vom jeweiligen Agenten abhängig. Auch die Art und Weise der Benachrichtigung

ist abhängig vom jeweiligen Material, in einigen Fällen kann der komplette Zustand des Materials versendet werden, in anderen nur eine Beschreibung der aktuellen Änderung.

6.3.3 Ontologie

Für die Verwaltung von Helfern und Materialien sind eine Reihe von Ontologiekonzepten von Bedeutung. Diese drehen sich einerseits um die Beschreibung der Konzepte in HeRA, wie etwa Ressource, Material, Helferagent, Dienste und Schnittstellen von Helferagenten, Helfer-/Benutzer-Zuordnung. Zusätzlich gibt es Ontologieobjekte für die verschiedenen Aktionen, die im System ausgeführt werden. Das umfasst beispielsweise die Anforderungen für das Auflisten, Erzeugen und Registrieren von Helferagenten oder für das Erzeugen und Beobachten von Ressourcenagenten.

Diese Elemente werden in einer Protégé-Ontologie [Protégé, 2011] beschrieben und teilweise in Verbindung mit den vorhandenen Konzepten des Multiagentensystem gebracht. Aus dieser Ontologie werden dann Java-Klassen erzeugt, die in den Wissensbasen der Agenten und in den Nachrichten der Agenten untereinander verwendet werden können.

6.3.4 Whiteboard-Beispiel

Das eingangs erwähnte Whiteboardsystem wird hier jetzt noch einmal in Hinblick auf den Entwurf diskutiert. Zusätzlich zu den ausgeführten Ontologieobjekten, Agenten und Interaktionen, die für das HeRA-System generell erforderlich sind, benötigen die hiermit implementierten Helfer und Materialien natürlich auch ihre eigenen spezifischen Einheiten. Für das Whiteboard-Beispiel, das bereits in Abschnitt 6.1.2 beschrieben worden ist, sind demnach folgende Elemente zu beachten und teilweise zu implementieren:

- Eine Anwendungsontologie
- Die Arbeitsumgebung
- Der Whiteboard-Ressourcenagent
- Der Whiteboard-Helferagent
- Die Interaktionen zwischen diesen Agenten

6.3.4.1 Ontologie

Die Ontologie umfasst die Begriffe für den Inhalt des Whiteboards und für die Operationen der verschiedenen Aspekte. In der prototypischen Implementation ist der Inhalt eines Whiteboards einfacher Text, der als `WhiteboardContent`

gekapselt wird. Hier könnten in weiteren Ausbaustufen komplexere Typen (z.B. Grafiken) verwendet werden.

Mit diesem Inhalt kann eine `WhiteboardChange`-Aktion ausgeführt werden. Dies ist hier ein einfaches Überschreiben des bisherigen Inhalts mit dem neuen Inhalt. In einer anspruchsvolleren Implementierung wären komplexere Aktionen sinnvoll, wie etwa Differenzen zu einer Basisversion, Einfügen von Objekten, Zusammenführen nebenläufiger Änderungen etc.

6.3.4.2 Arbeitsumgebung

Das Whiteboard stellt ein verteiltes Material dar, auf das verschiedene Benutzer über ihre jeweiligen Helfer zugreifen können. Die einzelnen Helferagenten sind über die Benutzeragenten mit den Plattformen der jeweiligen Benutzer verbunden. Das Material kann sich an einem beliebigen Ort im System befinden.

Um den gemeinschaftlichen Zugriff auf das Whiteboard zu ermöglichen, werden Agentennachrichten verwendet, die die Helfer- und Ressourcenagenten miteinander austauschen. So lange ein Benutzer einen Helferagenten besitzt, der mit dem Material verbunden ist, kann er den Inhalt des Whiteboards verfolgen und versuchen, auf das Whiteboard zu schreiben.

6.3.4.3 Ressource

Die zentrale Ressource in dieser Anwendung ist natürlich das Whiteboard. Für die Beispielimplementierung genügen die Basisoperationen Lesen und Schreiben. Aufwändigere Operationen wie Einfügen von Grafiken, Bearbeiten bestimmter Abschnitte, Formatierungen etc. sind für das Verständnis des Grundprinzips nicht wichtig.

Der Whiteboard-Ressourcenagent wird von einem Whiteboard-Helfer auf Anfrage eines Benutzers erzeugt und existiert dann zunächst unabhängig im System und kann von beliebigen Helferagenten angesprochen werden. Hierfür muss der Ressourcenagent einerseits natürlich seinen Inhalt in der Wissensbasis verwalten, andererseits eine Liste der registrierten Helferagenten, die sich als Beobachter eingetragen haben. Das Whiteboard verwendet hierfür die oben genannten Interaktionen für die Registrierung von Beobachtern und Änderungsbenachrichtigungen.

Der Whiteboard-Ressourcenagent kommuniziert mit den Whiteboard-Helferagenten. Diese müssen die Protokolle kennen, mit denen das Whiteboard angesprochen werden kann. Hier wird deutlich, dass Ressourcen und Helfer aufeinander abgestimmt sein müssen, es aber verschiedene Helfer geben kann, die auf einer Ressource operieren, sofern sie die entsprechende Schnittstelle beherrschen.

6.3.4.4 Helfer

Im Umgang mit einem (physikalischen) Whiteboard gibt es hauptsächlich zwei Funktionen: Betrachten, was auf dem Whiteboard geschrieben steht, und Veränderung dieses Inhaltes. Während für das Lesen kein besonderes Werkzeug benötigt wird (lediglich die Augen), wird für das Schreiben ein Stift benötigt. Teilweise verfügt ein Whiteboard auch über mehrere Stifte, die nebenläufig verwendet werden können.

Von diesen Eigenschaften ausgehend muss nun der Entwurf des Helferagenten als Softwarewerkzeug erfolgen. Lesen und Schreiben können als Operationen eines einzelnen Werkzeuges oder über unterschiedliche Werkzeuge implementiert werden. Werden unterschiedliche Werkzeuge zum Lesen und Schreiben verwendet, kann der wechselseitige Ausschluss über das Weiterreichen des Werkzeuges (Stift) zwischen den verschiedenen Benutzern intuitiv modelliert werden. Auf der anderen Seite ergibt sich durch verschiedene verwendete Werkzeuge eine höhere Komplexität der Interaktionen. Für dieses Beispiel wird daher nur ein einzelner Helfer zum Lesen und Schreiben implementiert.

Der Whiteboard-Helferagent ermöglicht dem Benutzer den Zugriff auf alle Aspekte des Whiteboards. Dies umfasst sowohl Funktionen zum Auffinden oder Erstellen eines neuen Whiteboards, als auch die Anzeige und Bearbeitung von existierenden Boards. Das Erstellen eines neuen Materials verwendet das oben bereits beschriebene Protokoll. Der Whiteboard-Helfer muss hierfür natürlich wissen, wie eine neue Whiteboard-Ressource aussieht.

Die Rolle des Helfers bei der Bearbeitung ist vor allem eine vermittelnde: Auf der einen Seite muss er die Protokolle des Whiteboards kennen, um darauf zugreifen zu können, zum anderen muss er die Funktionalitäten in sinnvoller Art und Weise dem Benutzer über den Benutzeragenten zugreifbar machen. Abb. 6.5 zeigt die verschiedenen Interaktionen, die zwischen Whiteboard-Helfer und Ressource stattfinden und damit die Benutzungsschnittstelle der Ressource definieren. Zum einen ist dies die Schreib-Interaktion, mit der Daten auf das Whiteboard geschrieben werden können, zum anderen der Beobachtermechanismus, der aus einer Anmeldung als Beobachter und den folgenden Benachrichtigungen bei Statusänderungen besteht.

Der Helferagent stellt über den Benutzeragenten eine Benutzungsschnittstelle zur Verfügung. Wird eine Operation dieser Schnittstelle aufgerufen, entscheidet der Helferagent, was zu unternehmen ist und sendet gebenenfalls Nachrichten an den Ressourcenagenten, um den Zustand abzufragen oder Zustandsänderungen hervorzurufen. Die Ergebnisse dieser Operation werden dann wieder in der Benutzungsschnittstelle angezeigt.

6.3.4.5 Interaktionen

Für die Registrierung des Helferagenten sowie die Erzeugung und Beobachtung des Ressourcenagenten im Whiteboard-Beispiel werden die oben angegebenen

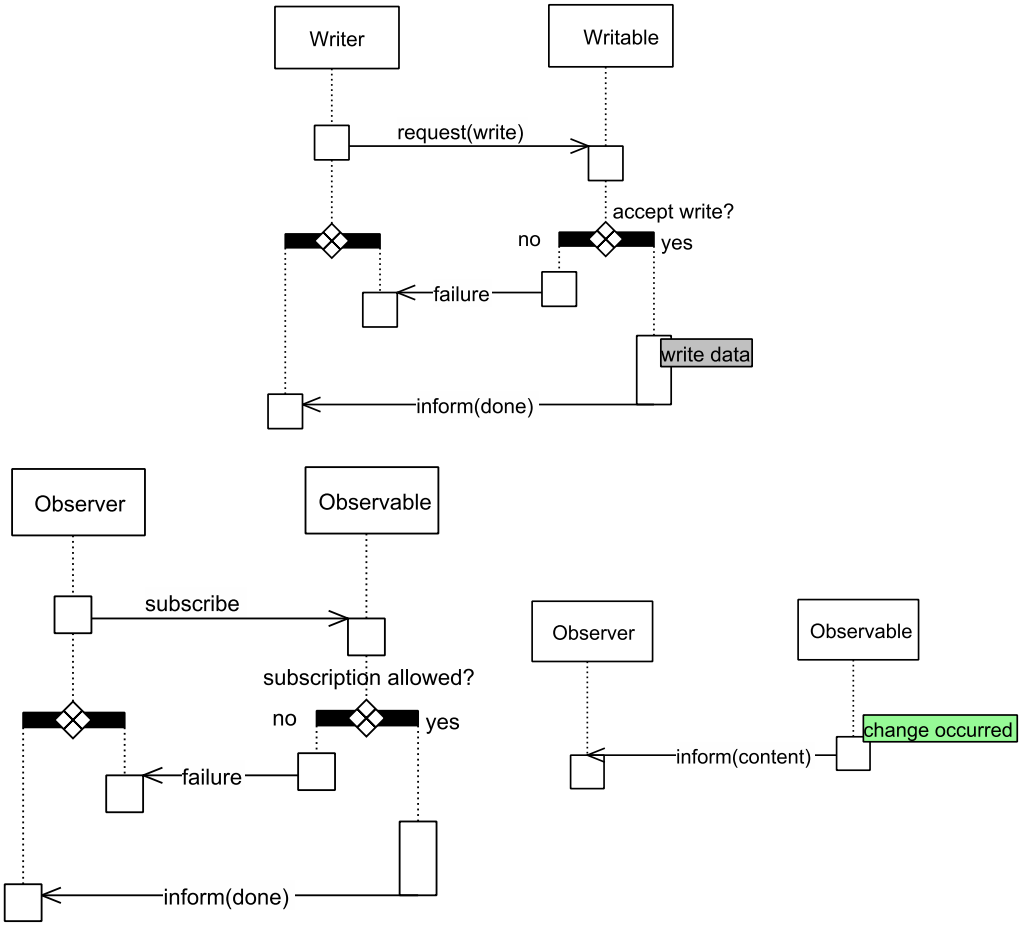

Abbildung 6.5: Interaktionen der Whiteboard-Anwendung

Standardinteraktionen des HERA-Systems verwendet. Die einzige Interaktion, die neu implementiert werden muss, ist die zum Schreiben auf dem Whiteboard.

Um zu schreiben, sendet der Helferagent eine Nachricht mit dem Inhalt `request(whiteboardChange)` an den Ressourcenagenten. Dieser kann die Änderung annehmen und seinen Zustand entsprechend ändern, oder er kann die Änderung ablehnen. Die Beobachtung des Inhalts erfolgt über den Beobachter-Mechanismus. Der Helfer registriert sich als Beobachter für Änderungen des Whiteboards und wird dann automatisch über Änderungen auf dem Laufenden gehalten. Eine explizite Leseoperation ist daher nicht erforderlich.

6.3.4.6 Implementierung

Die prototypische Implementierung des HERA-Systems wird ausführlicher in Kapitel 8 beschrieben. Dort wird auch in Abschnitt 8.3 die hier beschriebene Whiteboard-Anwendung umgesetzt.

6.4 Zusammenfassung

In diesem Kapitel wurde das HeRA-System als einer der Bestandteile des Potato-Systems beschrieben. Hierfür werden Entwurfskonzepte aus dem objektorientierten Werkzeug und Material Ansatz verwendet und auf Multiagentensysteme übertragen. Das HeRA-System definiert konkrete Rollenmuster, nach denen Agenten miteinander interagieren, um gemeinsam eine nützliche Funktionalität für Anwender des Systems zu erbringen.

Hierfür werden einige der Konzepte aus dem WAM Ansatz aufgegriffen und in eine agentenorientierte Form überführt. Damit können die Vorteile des WAM Ansatzes, ebenso wie die der Agentenorientierung genutzt werden. Agenten erlauben es, statt einfacher Werkzeuge, die durch einen Benutzer verwendet werden, adaptive und proaktive Helferagenten zu implementieren, die den Benutzer in seiner Arbeit selbstständig unterstützen können.

Ressourcenagenten erlauben es, Arbeitsmaterialien zu entwerfen, die autonom über ihre Verwendung entscheiden können und so verschiedene Interaktionsmuster mit Hilfe dieser Ressourcen unterstützen können. Die Verwendung mobiler Agenten erlaubt darüber hinaus den Austausch von Ressourcen und Helfern zwischen verschiedenen Plattformen innerhalb eines verteilten Multiagentensystems.

Der Benutzeragent als Arbeitsplatz eines Benutzers kann beliebig konfiguriert und um neue Funktionalität erweitert werden, indem nach Bedarf Helfer- und Ressourcenagenten hinzugefügt werden. Dies erlaubt die Konstruktion vielfältiger und flexibler Agentenanwendungen.

Entsprechend dem Paose-Vorgehen wurde das HeRA-System durch einen agentenorientierten Entwurf und eine Spezifikation der beteiligten Agentenrollen, der Interaktionen zwischen diesen Rollen sowie der verwendeten Ontologie konzipiert. Die verwendeten Agentenrollen orientieren sich am Referenznetzmodell aus Kapitel 5 und werden am Beispiel einer Whiteboard-Anwendung dargestellt.

Durch das HeRA-System wird die Architektur von Multiagentensystemen verbessert, indem zusätzliche Kriterien zur Strukturierung geschaffen werden. Mulan stellte bereits eine ähnliche Strukturierung für Referenznetzsysteme auf, die nun durch weitere Strukturen verfeinert wird. Anstatt Agenten durch die Angabe von Wissensbasis und Protokollnetzen zu definieren, werden ganze Agentenanwendungen durch die Angabe der darin vorhandenen Helfer- und Ressourcenagenten entwickelt.

Im folgenden Kapitel werden Konzepte für die Prozessunterstützung betrachtet. Die Verwendung der Helfer- und Ressourcenagenten des HeRA-Systems erweitert Pia zu dem in Kapitel 5 konzipierten Potato-System.

Kapitel 7

Prozessunterstützung in Multiagentensystemen - Pia

In Kapitel 6 wurde das HeRA-System vorgestellt, mit dem Helfer- und Ressourcenagenten erzeugt werden können, die Benutzer bei der Erledigung ihrer Aufgaben unterstützen. Die zweite Säule, auf der das Potato-System basiert, ist die agentenorientierte Prozessinfrastruktur Pia. Diese Prozessinfrastruktur für Agenten wird in verschiedenen Ausbaustufen beschrieben in [Reese et al., 2005, Reese et al., 2006a, Reese, 2010, Wagner, 2009b].

In diesem Kapitel wird nun zunächst ein Überblick über das Pia-System gegeben und die verschiedenen Bestandteile erläutert. Anschließend wird diskutiert, welche Anpassungen erforderlich sind, um HeRA und Pia zu einem gemeinsamen System zu vereinigen, dem Potato-System. Schließlich wird noch ein Vorgehen beschrieben, das Modifikationen an laufenden Workflowprozessen ermöglicht.

7.1 Überblick

Wie in Kapitel 5 beschrieben, bildet die Prozessinfrastruktur einen wichtigen Bestandteil des Potato-Systems. Entsprechend der dort vorgenommenen Kategorisierung wird für das Potato-System eine Integration von Helferagenten und Prozessinfrastruktur auf Stufe IV durchgeführt. Das bedeutet, dass das System nach außen hin als Workflow Management System auftritt, dabei aber die Helfer- und Ressourcenagenten aus HeRA verwendet.

Im Rahmen dieser Arbeit wurde auch an der Prozessinfrastruktur mit entwickelt. Das Hauptaugenmerk liegt aber hier in der Verwendung von Pia als einem Bestandteil des Potato-Systems. In Abschnitt 7.2 wird daher zunächst die Prozessinfrastruktur beschrieben, wie sie unter anderem in [Reese, 2010] dargestellt wird. Dabei wird auf die zugrundeliegenden Konzepte und Techniken eingegangen und die beteiligten Agenten und Interaktionen werden erläutert. Die Prozessinfrastruktur selbst ist in dieser Arbeit nur am Rande Thema,

der Schwerpunkt liegt hier auf der Verwendung und Integration in Verbindung mit dem HeRA-System.

Ausgehend davon wird dann in Abschnitt 7.3 beschrieben, wie durch die Verwendung von Helferagenten in Pia eine Architektur für Multiagentenanwendungen geschaffen werden kann, die Struktur- und Prozessaspekte vereint. Konkret ist es auch in Potato möglich, Helfer- und Ressourcenagenten unabhängig vom WfMS zu definieren und zu verwenden. Die hier vorgestellten Erweiterungen zielen aber darauf ab, das WfMS auf die Verwendung von Helfer- und Ressourcenagenten einzustellen, um so die Prozessinfrastruktur um zusätzliche Funktionalität zu erweitern. Für Anwendungsentwickler wird das System vornehmlich als WfMS auftreten und die entsprechenden Schnittstellen anbieten.

Es wird beschrieben, welche Anpassungen erforderlich sind, um Helfer- und Ressourcenagenten in Pia nutzen zu können. Hierfür sind einerseits Anpassungen an einigen Interaktionen und Agenten der Prozessinfrastruktur erforderlich, andererseits müssen spezielle Helferagenten implementiert werden, die die Verbindung realisieren. Insbesondere wird der Aktivitätsagent vorgestellt, ein spezieller Typ von Helferagent, der dazu dient, dem Benutzer die Bearbeitung der im Workflow anfallenden Aktivitäten zu ermöglichen.

7.2 Agentenbasierte Prozessinfrastruktur

In diesem Abschnitt wird die Prozessinfrastruktur vorgestellt, mit deren Hilfe Abläufe in Potato koordiniert werden. Diese Infrastruktur basiert auf dem petrinetzbasierten Workflow Management System von Jacob ([Jacob, 2002a]). Dieses System wurde als Multiagentensystem neu implementiert, was es unter anderem ermöglicht, das WfMS auf mehrere Agentenplattformen aufzuteilen.

Die ersten Arbeiten zur Implementierung eines WfMS als Multiagentensystem in Mulan/Capa wurden als Lehreprojekt durchgeführt [Reese et al., 2005]. In späteren Erweiterungen wurde eine weitere Integration mit offenen Agentensystemen [Reese et al., 2006a] und Funktionen für die Verteilung von Workflows durch die Fragmentierung von Workflowinstanzen und die Verteilung auf verschiedene WfMS untersucht [Reese et al., 2006a]. Dieses System wird als Prozessinfrastruktur für Agentenanwendungen (Pia) in [Reese, 2010] vorgestellt und wird beständig weiterentwickelt [Wagner, 2009a, Wagner, 2009b, Wagner, 2009c].

7.2.1 Überblick

Zur Ausführung und Verwaltung von Workflows wird ein Multiagentensystem verwendet. Die verschiedenen Teile eines Workflowmanagementsystems (vgl. Abschnitt 3.6) sind darin als interagierende Agenten vorhanden. Hierfür wurde zunächst eine Ontologie entworfen, die die erforderlichen Konzepte für

ein WfMS enthält und die Kommunikation der Agenten über diese Konzepte ermöglicht. Desweiteren wurden die verschiedenen Rollen und Interaktionen identifiziert und implementiert, die für die Funktion eines WfMS erforderlich sind.

Im Folgenden wird zunöchst die Task-Transition beschrieben, die in der Referenznetz-Version von Workflownetzen zur Modellierung von Aktivitäten eingesetzt wird. Anschließend werden die Bezüge zum WfMS-Referenzmodell der WfMC [WfMC, 1995] aufgezeigt, aus denen sich dann der Aufbau der Prozessinfrastruktur ergibt. Diese wird durch die PAOSE-typischen Bestandteile Ontologie, Agentenrollen und Interaktionen beschrieben.

7.2.2 Task-Transition

In Abschnitt 3.1.2 wurde die Netzklasse der Workflownetze für die Modellierung von Geschäftsprozessen beschrieben. Diese abstrahieren von einer ganzen Reihe von Eigenschaften eines Workflows, die für die konkrete Ausführung in einem WfMS erforderlich sind. Daher werden die Workflows in PIA durch eine Workflow-Version von Referenznetzen modelliert. Dies erlaubt es unter anderem, Parameter und Ressourcen im Workflownetz zu verwalten, aber auch direkt die auszuführenden Aufgaben zu beschreiben.

Für die Bearbeitung einer Workflowaktivität wurde in [JACOB, 2002a] ein neues Netzelement in Renew eingeführt, die Workflowtransition oder Task-Transition. Im Gegensatz zu einer normalen Transition erlaubt sie es, einen begonnenen Schaltvorgang wieder abzubrechen und die beteiligten Marken wieder auf die Eingangsstellen zurückzulegen. Dadurch ist es möglich, dass ein Akteur ein Workitem annimmt, dass dann zu einer Aktivität wird (Vgl. Abschnitt 3.6). Diese kann dann bearbeitet und abgeschlossen werden, aber auch vorzeitig abgebrochen und wieder für andere Bearbeiter freigegeben werden. Intern wird dies realisiert durch eine Übersetzung der Workflowtransition in mehrere separate Netzelemente (siehe Abb. 7.1).

Wird die Bearbeitung durch das Schalten der Request-Transition begonnen, werden die Marken aus dem Vorbereich der Transition abgezogen und eine Marke in die interne Stelle (active) gelegt. Die Aufgabe ist nun für einen Bearbeiter reserviert. Durch die Confirm-Transition wird die Aufgabe abgeschlossen und die Marken auf die Ausgangsstellen verteilt, die Cancel-Transition hingegen stellt den Zustand vor dem Beginn der Aktivität wieder her.

Das Schalten der Request-Transition stellt eine Zuweisung der Aufgabe an einen Workflow-Teilnehmer dar. Bevor das passieren kann, muss die Aufgabe jedoch allen in Frage kommenden Benutzern in ihrer Worklist vorgelegt werden. Dies wird durch einen Beobachtermechanismus ermöglicht, der eine Aktion im System auslöst, wenn eine Task-Transition aktiviert ist, also noch bevor sie tatsächlich schaltet. Der Aktivierungszustand der Task-Transition wird dann durch das WfMS umgesetzt in eine Aktualisierung der Aufgabenlisten der angeschlossenen Teilnehmer. Erst die Annahme einer Aufgabe zur

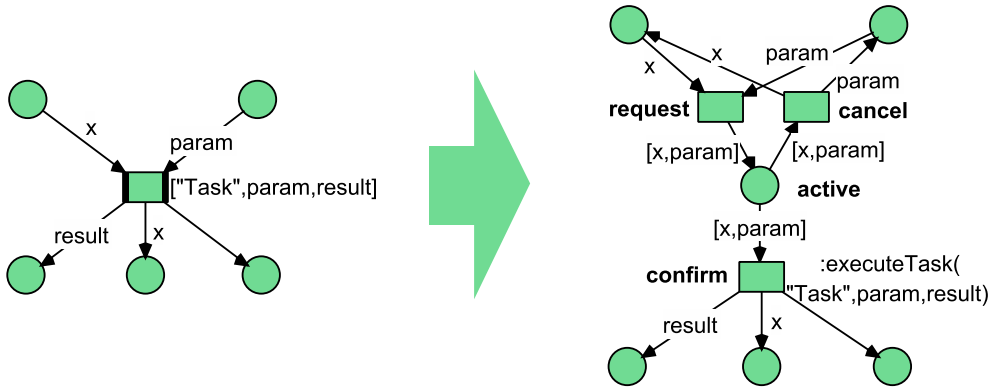

Abbildung 7.1: Task-Transition

Bearbeitung bewirkt ein tatsächliches Schalten der Transition.

Eine Task-Transition wird mit einer dreiteiligen Anschrift versehen, die die auszuführende Aufgabe beschreibt und die vom WfMS ausgewertet wird. Der erste Bestandteil dieser Anschrift ist eine Bezeichnung der Aufgabe, über die weitere Informationen zur Aufgabe aus einer Datenbank abgerufen werden können, etwa welche Art von Anwendung für die Bearbeitung verwendet werden soll. Der zweite Teil beschreibt die Eingabeparameter der Aufgabe. Hier können beispielsweise Informationen aus früheren Aufgaben übergeben werden, die die aktuelle Aufgabe beeinflussen. Der letzte Parameter schließlich beschreibt das Ergebnis der Aufgabe und wird während der Ausführung der Aufgabe ermittelt.

7.2.3 Bezug zum WfMC Referenzmodell

In Abschnitt 3.6.3.2 wurden die Bestandteile eines WfMS nach dem WfMC Referenzmodell beschrieben. An diesem Modell orientiert sich auch die Prozessinfrastruktur. Die Workflow Engine und der Workflow Enactment Service werden als interagierende Agenten implementiert. Ebenso werden Agenten implementiert, die die verschiedenen Schnittstellen des Systems nach außen darstellen.

Die Schnittstelle für Prozessdefinitionen wird realisiert durch den Workflow Definition Database Agenten, der Workflowdefinitionen verwaltet und verantwortlich ist für die Erstellung, Bearbeitung und den Zugriff auf Workflowdefinitionen für die Ausführung. Die Workflow Client Application Schnittstelle wird repräsentiert durch den WfMS-Agenten, der die Schnittstelle des WfMS nach außen hin darstellt und den Workitem Dispatcher, der die Worklisten der Benutzer verwaltet.

Zusätzlich werden eine Reihe von Agenten benötigt, um Verwaltungsaufgaben des Systems auszuführen, wie die Benutzerverwaltung und die Verwaltung der Prozess- und Aufgabendefinitionen. Gemeinsam bilden diese Agenten eine

Einheit, die nach außen hin als ein Workflow Management System auftreten.

7.2.4 Ontologie

Für die Implementierung der Prozessinfrastruktur sind zunächst die verwendeten Konzepte in einer Ontologie festgehalten worden. Sie orientiert sich im Wesentlichen an der Terminologie des WfMC-Referenzmodells [Kreplin, 1998, WfMC, 1999]. Die entsprechenden Begriffe sind bereits in Abschnitt 3.6.3.1 erläutert worden.

- **Workflow-Definition** Die Beschreibung eines Workflows umfasst zunächst einen Namen, eine textuelle Beschreibung und eine Versionsnummer, um den Workflow referenzieren und auffinden zu können. Dazu kommt dann die eigentliche, programmatische Beschreibung als Workflow-Petrinetz. Für eine informelle Beschreibung eines Workflows gibt es darüber hinaus ein Objekt „Workflow-Beschreibung“.

- **Workflow-Instanz** Zur Bearbeitung eines konkreten Falls wird eine Instanz eines Workflows erzeugt. Workflowinstanzen können über einen eindeutigen Bezeichner identifiziert werden.

- **Aufgabe (Task)** Eine Workflow-Definition enthält eine Reihe von Aufgaben, die im Verlauf des Workflows bearbeitet werden müssen. Aufgaben besitzen neben einem Bezeichner eine Aktivierungsregel, die beschreibt, wie die Workflow-Teilnehmer ermittelt werden, denen die Aufgabe angeboten wird, eine Priorität zur Sortierung innerhalb der Aufgabenliste eines Benutzers, sowie eine Definition der konkret auszuführenden Tätigkeit.

  In Pia gibt es eine Reihe vordefinierter Aufgabentypen, wie etwa den Formulartask, die über weitere Parameter an den Kontext angepasst werden können. Im Potato-System werden Aufgaben durch spezielle Aktivitätsagenten ausgeführt, wodurch eine dynamischere Auswahl möglich wird.

- **Workitem** Bei der Instanziierung eines Workflows werden die in der Workflow-Definition beschriebenen Aufgaben zu konkreten Workitems. Über die Aktivierungsregel der Aufgabe wird ermittelt, welchen Agenten ein Workitem über die Aufgabenliste zur Bearbeitung angeboten wird.

- **Aktivität (Activity)** Hat ein Agent ein Workitem zur Bearbeitung akzeptiert, wird daraus eine Aktivität für diesen Agenten. In Pia muss der Client für jeden Typ von Aufgabe wissen, wie sie zu bearbeiten ist. In Potato wird ein Activity-Agent erzeugt, der für die Ausführung der Aktivität verantwortlich ist.

- **Formular** Ein Beispiel für Aufgaben in PIA ist die Formularaufgabe. Ein Formular besteht aus verschiedenen Formularfeldern (Texteingabefelder, Comboboxen, Checkboxen etc.), aus denen der Client dann die Formulardarstellung generiert. Über `FormToFormRules` können Daten, die in einem Formular eingegeben worden sind, in einem anderen Formular wieder verwendet werden.

- **Bearbeiter (Executor)** Teilnehmer des WfMS, die die Rolle des Bearbeiters einnehmen, können Workitems zugewiesen bekommen und Aktivitäten ausführen.

- **Prädikate** Zur Formulierung bestimmter Aussagen bezüglich des WfMS werden Prädikate definiert, die die o.g. Konzepte in einen Zusammenhang setzen. Beispiele dafür sind etwa `current-workitems-of`, für die aktuelle Worklist eines Bearbeiters, oder `current-activities-of` für die derzeit ausgeführten Aktivitäten. Diese Prädikate werden in den Nachrichten verwendet, die zwischen den Agenten des WfMS und mit den Anwendern ausgetauscht werden.

- **Agentenaktionen** Ebenfalls für die Kommunikation der verschiedenen Agenten untereinander werden eine Reihe von Agentenaktionen definiert, die über `request`-Nachrichten angefordert werden. Dies reicht von `login/logout` bis `assign-workitem/set-activity-done/-canceled`. Auch für den Zugriff auf die verschiedenen Datenbankagenten gibt es verschiedene Aktionen (`create-/retrieve-/update-db-entry`)

7.2.5 Agentenrollen

Die Prozessinfrastruktur besteht aus einer Reihe interagierender Agenten, deren Aufgaben in diesem Abschnitt beschrieben werden. Die Aufteilung der Funktionalität orientiert sich am WFMC-Referenzmodell ([WfMC, 1995], Vgl. Abschnitt 3.6) mit einigen Erweiterungen.

Abb. 7.2 zeigt den Aufbau der Prozessinfrastruktur und die verschiedenen Agenten, die darin eine Rolle spielen. Das WfMS ist dabei als Plattform dargestellt, die die anderen Teilagenten des Systems enthält, ebenso wie der Workflow Enactment Service seinerseits als Plattform für Workflow Engine Agenten dargestellt wird. In der aktuellen Version von PIA ist dies als eine logische Zuordnung zu verstehen. Ansätze zur Schachtelung von Agenten und Agentenplattformen werden in [SCHLEINZER, 2007] ausgeführt, diese sind aber bislang weder in PIA noch in POTATO implementiert.

Durch die Aufteilung auf verschiedene Rollen ist es möglich, die Funktionalität auf unterschiedliche Agentenplattformen zu verteilen, indem die zugehörigen Agenten verteilt werden. Um den Nachrichtenoverhead zwischen Agenten zu reduzieren, können aber auch mehrere Rollen in einem Agenten zusammengefasst werden. Folgende Agentenrollen sind vorhanden:

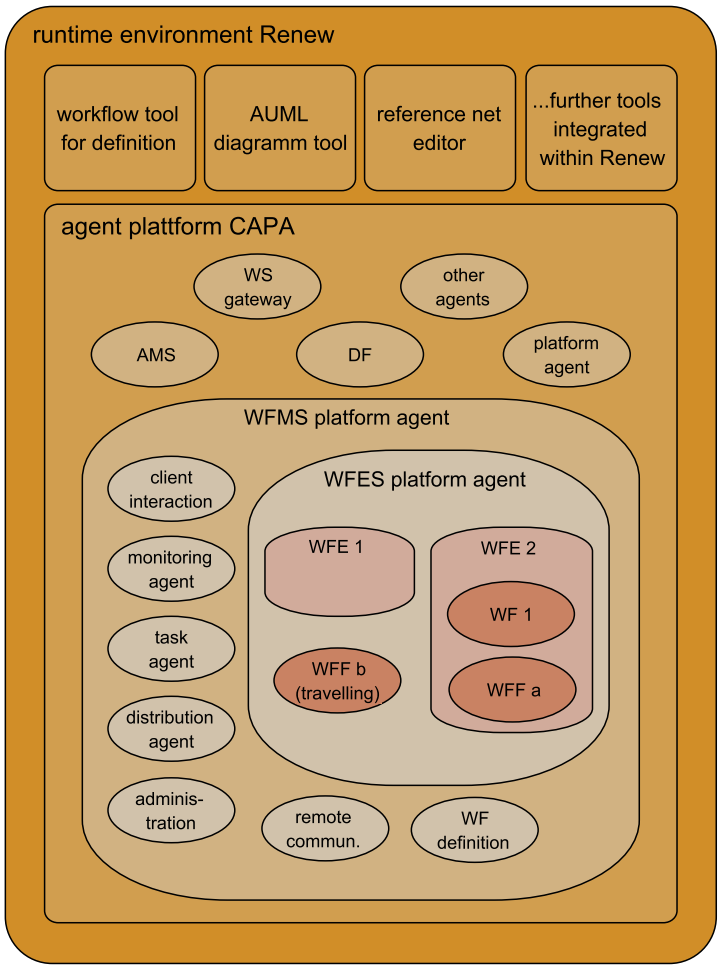

Abbildung 7.2: Aufbau der Prozessinfrastruktur (Aus [Reese et al., 2006a])

- WfMS Agent
- Workflow Enactment Service (WFES) Agent
- Workflow Engine (WFE) Agent
- Workitem Dispatcher
- Account Manager
- Workflow Definition Database (WFDDB) Agent
- Task Database (TaskDB) Agent
- Roles/Rules/Rights Database (RRRDB) Agent

Zusätzlich ist die Rolle des Workflow-Users zu erwähnen. Benutzer des WfMS sind selbst nicht Teil des Systems, nehmen aber an vielen der hier beschriebenen Interaktionen teil.

7.2.5.1 WfMS-Agent

Der WfMS Agent repräsentiert das Workflowmanagementsystem nach außen hin. Beim Aufsetzen des Systems startet er die verschiedenen Subkomponenten und dient für Benutzer als erster zentraler Anlaufpunkt. Eine Möglichkeit besteht darin, alle Kommunikation zwischen Benutzer und WfMS über diesen Agenten laufen zu lassen. Dadurch würde aber der Kommunikationsaufwand unnötig erhöht und die Gefahr entstehen, dass der WfMS Agent zum Engpass wird. Stattdessen teilt er einem Benutzer bei der Anmeldung am System die Adressen der für ihn zuständigen Agenten mit, mit denen er dann direkt kommuniziert.

7.2.5.2 Workflow Enactment Service (WFES) Agent

Der Workflow Enactment Service Agent ist zuständig für die Verwaltung der verschiedenen Workflow Engines innerhalb des WfMS. Er hat die Aufgabe, neue Workflows zu starten und einen Überblick über die derzeit laufenden Workflowinstanzen zu behalten. Wenn ein neuer Workflow gestartet werden soll, sorgt er dafür, dass die Definition abgerufen und ein neuer Workflow bei einer Workflow Engine gestartet und beim Workitem Dispatcher registriert wird. Erweiterungen zu Pia erlauben es, Workflows nicht direkt in der WFEngine zu starten, sondern in einem Workflow Agenten oder einem Strukturagenten (Vgl. [Wagner, 2009c]. Dies erlaubt eine höhere Flexibilisierung der Workflows und ermöglicht die Modellierung und Ausführung verteilter, interorganisationaler Workflows. Diese Art der Workflowausführung wird hier aber nicht weiter betrachtet.

Die einzelnen WFEngine Agenten melden jeweils ihre aktivierten Workitems an den WFES, der in Zusammenarbeit mit dem Workitem Dispatcher die Zuordnungen zwischen Workflows und Benutzern regelt.

7.2.5.3 Workflow Engine (WFE) Agent

Ein Workflow Engine Agent hat die Aufgabe, Workflowinstanzen auszuführen. Eine Engine kann mehrere Workflowinstanzen ausführen. Ebenso kann ein WfMS mehrere WFEngines enthalten, beispielsweise um die Arbeit auf mehrere Plattformen zu verteilen.

Wenn der Benutzer eine neue Workflowinstanz startet, fordert der WFES die Engine auf, einen neuen Workflow zu starten und übergibt ihm eine Beschreibung des entsprechenden Workflownetzes. Dieses Netz wird instanziiert und die WFEngine registriert sich über eine Decision Component als `TaskOccurrenceListener`. Wenn sich der Aktivierungsstatus einer Task-Transition

ändert (vgl. Abschnitt 7.2.2) oder eine ihrer Teiltransitionen schaltet, wird die DC darüber informiert und startet die notwendigen Protokolle.

Die Aufgaben der WFEngine umfassen die Auswertung der aktivierten Transitionen und der für die Behandlung der Aufgaben zuständigen Agenten, die Weiterschaltung der Workflownetze, wenn Aufgaben übernommen und ausgeführt werden und die Ermittlung des Workflowendes. Wenn ein Benutzer ein Workitem annimmt oder eine Aktivität abschließt bzw. abbricht, wird das durch den WFEngine Agenten im zugehörigen Workflownetz festgehalten.

7.2.5.4 Workitem Dispatcher

Der Workitem Dispatcher ist dafür zuständig, die jeweils aktivierten Workitems den angemeldeten Benutzern zuzuordnen und sie über ihre jeweils gültige Worklist zu informieren. Die Worklist eines Benutzers ergibt sich aus allen derzeit aktivierten Aufgaben in allen laufenden Workflowinstanzen, deren Ausführungsregel der Benutzer erfüllt. Dafür bekommt er von den Workflow Engine Agenten jedesmal ein Statusupdate, wenn sich der Aktivierungszustand einer Task-Transition ändert, also wenn eine Transition aktiviert wird, startet zu schalten oder das Schalten beendet oder abbricht.

Diese Informationen werden zusammengetragen und in den Worklists der verschiedenen angemeldeten Workflowteilnehmer gesammelt. Eine Änderung an einer Task-Transition kann Änderungen in einer ganzen Reihe von Worklists hervorrufen, z.B. wenn eine Aufgabe von einem Teilnehmer angenommen wird und dann allen anderen nicht mehr zur Verfügung steht.

Der Workitem Dispatcher errechnet dann die Worklists neu und startet die Interaktion zur Benachrichtigung über eine Worklist-Änderung für alle Agenten, deren Liste sich geändert hat. Diese Interaktion wird auch gestartet, wenn ein Benutzer sich neu am System anmeldet, um ihm initial seine Worklist mitzuteilen. Andersherum bekommt der Workitem Dispatcher dann von den Benutzern Anfragen, die angebotenen Workitems zu bearbeiten. Diese Requests leitet er an die Workflow Engine weiter.

Der Workitem Dispatcher führt ebenfalls eine Liste der offenen Aktivitäten je Workflow Instanz, sowie eine Liste der angemeldeten Workflow-Teilnehmer und welche Aktivitäten ihnen jeweils zugeordnet sind. Dies sind die Aufgaben, die konkret einem Benutzer zur Bearbeitung zugewiesen worden sind. Bei jedem Update einer Workflow Engine werden die Listen aktualisiert und wenn sich die Liste eines Teilnehmers geändert hat, wird ihm ein aktualisierter Stand gesendet.

7.2.5.5 Account Manager

Der Account Manager besitzt Kenntnis über die verschiedenen Benutzer, die sich am WfMS anmelden können und ihre Berechtigungen. Er wird von den

anderen Infrastrukturagenten befragt, ob ein Benutzer eine bestimmte Aktion ausführen, bzw. sich am System anmelden darf.

7.2.5.6 Workflow Definition Database (WFDDB) Agent

Der WFDDB Agent verwaltet die Workflowdefinitionen, die im WfMS instanziiert werden können. Auf Anfrage kann eine Liste der bekannten Workflows produziert werden, aus denen dann einer zur Instanziierung ausgewählt werden kann. Bei der Erzeugung einer neuen Workflowinstanz wird ein gespeicherter Workflow an den Workflow Enactment Service übergeben, der ihn an eine Workflow Engine übergibt, die damit eine neue Instanz erzeugt.

Der WFDDB-Agent ist darüber hinaus verantwortlich für die Erzeugung und Bearbeitung der Workflowdefinitionen. Diese Funktionalität ist in der aktuellen Version von Pia nicht implementiert.

Für die Umsetzung der Funktionalität sind Interaktionen erforderlich für das Erzeugen, Bearbeiten und Löschen von Workflowdefinitionen. Der WFDDB-Agent nimmt dann nach Prüfung die entsprechenden Änderungen an der Datenbasis vor.

Hierfür gibt es Algorithmen zur Überprüfung der Soundness einer Workflowdefinition, wie in Abschnitt 7.4 beschrieben. Wird die Definition eines Workflows zu dessen Laufzeit geändert, muss zudem eine Übertragungsvorschrift spezifiziert werden. Diese teilt der WFDDB Agent bei einer Änderung allen betroffenen Workflow Engines mit, die dann die Änderungen vornehmen.

7.2.5.7 Task Database (TaskDB) Agent

Der TaskDB Agent verwaltet eine Liste der verschiedenen Aufgabentypen, die in dem jeweiligen WfMS ausgeführt werden können. Er wird von der Workflow Engine konsultiert, wenn eine neue Aufgabe ausgeführt werden soll und liefert dann die nötigen Informationen zur Bearbeitung. Handelt es sich um eine Aufgabe, die über einen Activity-Agenten behandelt wird, weiß der TaskDB Agent, welche Art von Helferagent für die Aufgabe instanziiert werden muss, und wie er initialisiert wird.

7.2.5.8 Roles/Rules/Rights Database (RRRDB) Agent

Dieser Agent verwaltet die verschiedenen Rollen, Regeln und Berechtigungen, die im WfMS definiert sind. Im Gegensatz zum Account Manager sind dies die Rollen, die in Bezug auf die jeweiligen Workflows definiert sind, also die Anwendungsrollen.

Anwendungsrollen können bei der Definition eines neuen Workflows angelegt und bearbeitet werden. Der RRRDB-Agent wird dann darüber informiert, welche Rollen es gibt und wird auch darüber informiert, welche Benutzer über welche Rollen verfügen. Bei der Ausführung eines Workflows erfolgen Aufgabenzuweisungen oft über Rollenzugehörigkeiten. Bei einer Aufgabe wird dann

nicht spezifiziert, welcher Benutzer sie ausführt, sondern lediglich, welche Rollen hierfür in Frage kommen. Der RRRDB Agent kann dann eine Übersetzung vornehmen, um die Benutzer zu ermitteln, die die richtige Rolle innehaben.

Der Workitem Dispatcher verwendet die Interaktion `getSatisfiedRules` um festzustellen, welche Aufgaben welchem Benutzer aufgrund der definierten Regeln zugewiesen werden können.

7.2.5.9 Workflow User Agent

Der Benutzer des WfMS kann eine Reihe von Rollen einnehmen. Die wichtigste ist die des Executors, der Workitems und Activities vorgelegt bekommt und diese bearbeitet. Als Initiator startet er neue Prozesse, als Administrator kann er neue Prozesse definieren oder Berechtigungen bearbeiten. All diese Rollen setzen voraus, dass er sich mit gültigen Zugangsdaten am System anmelden kann.

Diese Agentenrolle sollte nicht mit dem Benutzeragenten des HeRA-Systems verwechselt werden. Als Workflow User Agent kann jeder Agent auftreten, der die entsprechenden Interaktionen zur Kommunikation mit dem WfMS beherrscht. Im POTATO-System wird dies durch einen Helferagenten realisiert (siehe 7.3.2.1). In Multiagentensystemen, die nur die Prozessinfrastruktur nutzen, kann diese Rolle aber auch durch einen Agenten ausgefüllt werden, der direkt unter der Kontrolle eines Benutzers steht.

Benutzer des WfMS können eine ganze Reihe von Rollen einnehmen. Sie alle haben gemeinsam, dass sie nicht Teil des WfMS im engeren Sinne sind. Oft kann ein Agent mehrere dieser Rollen einnehmen, zur Abgrenzung und um Rollen individuell zuweisen zu können, werden sie hier aber einzeln aufgeführt. Die Benutzerfunktionalität umfasst verschiedene Bereiche:

- **Anmeldung am System und Worklistverwaltung** Ein Benutzer mit der Rolle „Workflow Bearbeiter“ hat Zugriff auf die aktuelle Aufgabenliste, sowie die Auswahl von Aufgaben zur Bearbeitung.

- **Workflowdefinitionen erstellen und bearbeiten** Die Bearbeitung der Prozessstruktur erfolgt mit Hilfe des RENEW-Werkzeugs, zusätzliche Parameter, Aufgabendefinitionen, Formulare und anderes werden in einer Datenbank gespeichert. Zurzeit wird diese Rolle noch nicht durch Interaktionen im System unterstützt.

- **Workflow starten** Wenn der Benutzer die entsprechende Berechtigung hat, kann er vorhandene Workflowdefinitionen suchen und neue Prozesse im System starten.

- **Workflows untersuchen und modifizieren** Die Administration von Workflows umfasst das Untersuchen und die Bearbeitung von laufenden

Workflows und ist nur für Benutzer mit speziellen Berechtigungen verfügbar. Auch diese Funktionen sind derzeit nur durch externen Zugriff, zum Beispiel über den MulanViewer möglich.

7.2.6 Interaktionen

In diesem Abschnitt werden die Interaktionen zwischen den Agenten der Prozessinfrastruktur beschrieben. Im letzten Abschnitt wurde bereits angedeutet, welche Arten von Interaktionen zwischen den Agenten vorkommen, in diesem Abschnitt werden diese Interaktionen nun genauer erörtert.

Tabelle 7.1 zeigt die Matrix der verschiedenen Interaktionen im PIA-System und welche Agentenrollen jeweils daran beteiligt sind. Die Tabelle enthält nicht alle Agentenrollen, z.B. sind der Accountmanager und der RRR-Agent nicht aufgeführt, weil bei fast jeder Interaktion Berechtigungsüberprüfungen vorgenommen werden müssen.

Durch die Aufteilung der Systemkomponenten auf eine relativ große Anzahl von Rollen werden die Vorgänge hervorgehoben, die hier erforderlich sind. Es ist allerdings möglich, die nötige Kommunikation zu reduzieren, indem mehrere Rollen in einem Agenten zusammengefasst werden. Die Interaktionen lassen sich unterteilen in:

- Anmeldeinteraktionen
- Verteilung der Aufgabenlisten
- Annehmen und Bearbeiten von Aufgaben
- Start und Abschluss von Workflows
- Bearbeitung von Stammdaten

7.2.6.1 Anmeldeinteraktionen

Bevor ein Workflow-Teilnehmer irgendeine Aktion im System vornehmen kann, muss er sich zunächst anmelden. Dadurch wird er als derzeit aktiv gekennzeichnet und ihm werden beispielsweise Nachrichten gesendet, wenn sich Änderungen an seiner Worklist ergeben. Dafür wendet er sich an den WfMS Agenten und sendet ihm einen Login-Request mit seinen Zugangsdaten. Diese werden beim Accountmanager abgeglichen und dem Benutzer entsprechend Rückmeldung gegeben.

Zusätzlich könnten an dieser Stelle Sicherheitschecks vorgenommen werden. In der Regel authentifiziert sich ein Benutzer über Passwort, Zertifikat oder ähnliches gegenüber dem System. Auf die Implementierung einer kompletten Sicherheitsinfrastruktur wurde zunächst verzichtet. Der RRR-Agent prüft lediglich, ob ein Benutzer für bestimmte Aktionen die nötige Berechtigung hat,

	Benutzer (alle Rollen)	Workflow Enactment Service	Workflow Engine	Workitem Dispatcher	Activity-Agent	WFDDB-Agent	WfMS-Agent	Sonstige Helfer
An-/Abmelden beim WfMS	X						X	
Worklist aktualisieren	X	X	X	X				
Aufgabe anfordern/beenden	X	X	X	X	X			
Aufgabe bearbeiten	X				X			
Neue Workflowdefinition	X	X				X		WFEdit
Workflowdefinition bearbeiten	X	X	X	X		X		WFEdit
Workflows auflisten	X					X		
Neuen Workflow starten	X	X	X	X		X		
Workflowstatus abfragen	X	X	X					
Workflowinstanz modifizieren	X	X	X	X				

Tabelle 7.1: Rollen und Interaktionen der Prozessinfrastruktur

komplexere Systeme wären aber denkbar und für produktive Systeme auch wichtig.

Anschließend erfragt der Benutzer beim WfMS Agenten die Adressen der anderen Agenten, mit denen er interagiert und meldet sich beim Workitem Dispatcher an, um seine Workitem- und Activitylisten zu erhalten. Nach Abschluss der Anmeldung wird dann durch den Workitem Dispatcher die aktuelle Worklist des Benutzers ermittelt. Das führt dazu, dass die Interaktion zur Aktualisierung der Worklist gestartet wird, um dem Benutzer diese Worklist mitzuteilen. Bis sich der Agent wieder abmeldet, erhält er nun immer automatisch Nachrichten, wenn sich seine Aufgabenlisten ändern.

7.2.6.2 Verteilung der Aufgabenlisten

Der Workitem Dispatcher führt die Worklists der verschiedenen am System beteiligten Benutzer. Änderungen ergeben sich, wenn der Workflow fortschreitet und nun neue Aufgaben vorliegen, wenn ein neuer Workflow gestartet wird oder wenn sich ein Benutzer neu anmeldet. Für alle Benutzer, für die sich eine Änderung ergeben hat, versendet der Workitem Dispatcher die geänderten Listen.

Die WFEngine Agenten verwalten die instanziierten Workflows. Wenn sich darin eine Änderung an der Aktivierung von Workflowtransitionen ergibt, er-

fährt der WFEngine Agent dies durch Beobachtung der Task-Transitionen. Er aktualisiert die Liste der aktivierten Workitems in seiner Wissensbasis und startet die Interaktion `updateWorkitemList` zum Aktualisieren der Worklisten.

Die WFEngine sendet eine Liste aller aktivierten Workitems an den WFES Agenten. Dieser konsolidiert die Listen aller im WfMS vorhandenen WFEngines und sendet diese Liste weiter an den Workitem Dispatcher. Der Workitem Dispatcher wiederum überprüft jetzt anhand der Zugriffsregeln, die er vom RRR Agenten abgefragt hat, und anhand seiner Liste der angemeldeten Benutzer, welcher Benutzer welche Aufgaben in seiner Worklist vorgelegt bekommt. Diese Listen sendet er dann, sofern Änderungen vorliegen, an die jeweiligen Benutzer.

Dies geschieht analog für Aktivitäten. Wenn ein Benutzer ein Workitem zur Bearbeitung angenommen hat, wird daraus eine Aktivität, die einem konkreten Benutzer zugeordnet ist. Auch dies erfährt die WFEngine durch Beobachtung der Task-Transitionen der instanziierten Workflows. Auch die Aktivitäten werden zunächst beim WFES Agenten konsolidiert und dann durch den Workitem Dispatcher an die zuständigen Benutzer verteilt.

7.2.6.3 Anfordern und Bearbeiten einer Aktivität

Ein Benutzer, der eine Liste von möglichen Workitems zur Bearbeitung vorliegen hat, kann sich entscheiden, eines oder mehrere davon zu bearbeiten. Wenn sich ein Benutzer aus der Liste der angebotenen Aktivitäten seiner Worklist eine Aufgabe zur Bearbeitung ausgewählt hat, startet er die Interaktion `requestWorkitem`.

Diese sendet eine Nachricht mit der betreffenden Aufgabe an den Workitem Dispatcher. Dieser prüft zunächst, ob der Benutzer die Berechtigung hat, diese Aufgabe zu bearbeiten und ob er sie in seiner Worklist vorliegen hat. Sind diese Prüfungen erfolgreich, sendet er die Anforderung weiter an den WFEnactment Service Agenten. Dieser ermittelt nun die zuständige WFEngine zur Aufgabe und leitet den Request weiter.

Die WFEngine erzeugt dann aus dem Workitem eine Aktivität, indem sie es dem angeforderten Bearbeiter zuweist. Dann schaltet sie die Task-Transition, die dem Workitem zugeordnet ist. Daraufhin wird das Workitem von allen Worklists entfernt, da sie nicht mehr als offen gekennzeichnet ist. Wird es in der Zwischenzeit erneut zur Bearbeitung angefordert, wird eine Fehlermeldung zurückgegeben.

Der Bearbeiter erhält eine Nachricht, dass er die Aufgabe erhalten hat und die nötigen Informationen zur Bearbeitung, also den Typ der Aufgabe und die Parameter, die im Workflow zugeordnet worden sind. Im Pia-System muss der Bearbeiter selbst wissen, wie jede Art von Aufgabe zu behandeln ist. Hier setzt später die Integration des HeRA-Systems an, indem spezielle Helferagenten für die Bearbeitung von Aufgaben eingesetzt werden.

Ist die Aufgabe in der Zwischenzeit bereits einem anderen Benutzer zugeordnet worden oder schlägt der Berechtigungstest fehl, wird eine entsprechende Fehlermeldung erzeugt. Der Benutzer kann dann eine aktualisierte Worklist anfordern, sofern er die Aufgabe noch immer auf seiner Worklist vorliegen hat.

7.2.6.4 Abschließen von Aktivitäten

Ein Benutzer kann eine Aktivität auf zwei Arten abschließen. Entweder die Bearbeitung war erfolgreich und die Aufgabe wird als erledigt markiert, oder sie war nicht erfolgreich und die Aufgabe wird wieder als offen gekennzeichnet.

Der Benutzer startet die Interaktion `confirmActivity` bzw. `cancelActivity` und sendet ein `setActivityDone/-Canceled`-Objekt an den Workitem Dispatcher. Im Falle eines erfolgreichen Abschlusses wird hierbei auch ein Ergebnisobjekt übergeben, das abhängig von der ausgeführten Aktivität ist. Im Falle eines Formtasks beispielsweise wäre dies eine Repräsentation des ausgefüllten Formulars. Der Workitem Dispatcher prüft die Berechtigung des Benutzers und ob diesem die Aktivität zugewiesen worden ist. Anschließend leitet er den Request weiter an den WFES Agenten, der ihn wiederum an die zuständige WFEngine weitergibt.

Die WFEngine ermittelt die Task-Transition, die für die Aktivität zuständig ist und veranlasst eine Schaltung der `confirm-` oder `cancel`-Transition. Das Ergebnisobjekt eines `setActivityDone`-Objektes wird hierbei mit übergeben und wird so in das Workflownetz überführt, von wo aus es in weiteren Aktivitäten verwendet werden kann.

7.2.6.5 Workflows auflisten

Bevor ein Initiator einen neuen Workflow starten kann, muss er zunächst eine Liste der im WfMS bekannten Workflow-Definitionen abfragen. Dazu sendet er eine Anfrage an den WFDDB-Agenten. Diese Anfrage kann Filterkriterien enthalten. Als Ergebnis wird eine Liste mit den Bezeichnungen und Beschreibungen der passenden Workflows zurückgesendet. Die Workflowdefinitionen selbst werden nicht gesendet, sondern lediglich eine Beschreibung der Funktionalität sowie eventuell erforderlicher Startparameter.

7.2.6.6 Start von Workflows

Benutzer, die die Rolle Initiator innehaben, können neue Prozesse starten. Dafür wählen sie einen Prozess aus der Liste der bekannten Definitionen aus. Um einen neuen Workflow zu starten, verwendet der Benutzer die Interaktion `startWorkflow`. Er sendet eine Nachricht mit dem Namen des gewünschten Workflows und eventuellen Parametern an den Workflow Enactment Service Agenten. Dieser prüft die Berechtigung des Benutzers und fragt dann beim Workflow Definition Database Agenten die Workflowdefinition ab.

Anschließend wählt er eine WFEngine aus und beauftragt diese mit der Ausführung des Workflows. Die Workflowdefinition ist eine Serialisierung eines Workflow-Petrinetzes, die nun von der WFEngine wieder in ein Netz umgesetzt und instanziiert wird. Die WFEngine registriert ihre DecisionComponent als `TaskOccurrenceListener` bei dem neu erzeugten Netz, um über Aktivierungen und Schaltvorgänge informiert zu werden. Abschließend wird eine Bestätigungsnachricht mit einer Identifizierung der Workflowinstanz zurückgegeben. Diese kann zum Beispiel für die Administration des Workflows verwendet werden, oder dient bei der Benachrichtigung über das Workflowende zur Identifikation.

7.2.6.7 Abschluss eines Workflows

Wenn ein Prozess beendet ist, wird dem Benutzer, der ihn initial gestartet hat, eine Rückmeldung mit der Identifikation der Workflow-Instanz gegeben. Am Ende eines Workflownetzes befindet sich immer eine Transition mit dem Kanal `:stopwf()`. Diese wird mit dem `instantiateWorkflow()`-Netz der WFEngine synchronisiert und signalisiert diesem, dass der Workflow beendet ist.

Die WFEngine entfernt sich daraufhin als Beobachter des entsprechenden Netzes, damit es zur Garbage Collection freigegeben wird. Anschließend sendet sie eine Nachricht über die erfolgreiche Beendigung an den WFES. Dieser informiert den Agenten, der den Workflow ursprünglich gestartet hat, über die Beendigung. Der Benutzer kann dann beispielsweise über eine Messagebox darüber informiert werden, dass der Prozess fertig ist. Ebenfalls ist es möglich, dass ein Workflow als Teil eines größeren Gesamtprozesses gestartet wurde, so dass die Beendigung eines Teilprozesses die Beendigung einer Aufgabe in einem anderen Workflow signalisiert.

7.2.6.8 Bearbeitung von Stammdaten

Die Bearbeitung von Stammdaten darf nur durch Benutzer mit speziellen Administrations-Privilegien vorgenommen werden. Es können die Workflowdefinitionen, die vorhandenen Tasks, Rollen, Regeln und Rechte, sowie die in den Workflows verwendeten Formulare bearbeitet werden.

Zu diesem Zweck kann der bearbeitende Agent die bisherige Version des zu bearbeitenden Elementes anfordern. Diese kann dann durch entsprechende Werkzeuge bearbeitet werden und die neue Version wieder an den Administrationsagenten zurückgesendet werden. Dieser muss dann prüfen, ob die Bearbeitung legal ist und führt die entsprechenden Änderungen durch. Unter Umständen müssen andere Agenten, die auf diese Daten angewiesen sind, informiert werden. Im aktuellen Prototypen sind diese Funktionen nicht implementiert.

7.3 Helferagenten in der Prozessinfrastruktur

Im letzten Abschnitt wurde die Prozessinfrastruktur vorgestellt, die die Ausführung agentenbasierter Workflows ermöglicht. In diesem Abschnitt wird nun beschrieben, wie dies mit den Helfer- und Ressourcenagenten aus Kapitel 6 verbunden werden kann, um weitere Flexibilität und Funktionalität zu erreichen.

7.3.1 Überblick

Die Prozessinfrastruktur hat den Zweck, die Entwicklung von Anwendungen zu ermöglichen, die im Kern die verteilte Ausführung von Prozessen umfassen. Ein solcher Prozess, oder vielmehr eine Sammlung verschiedener Prozesse ist auch die gemeinschaftliche Entwicklung von Software, wie sie durch die verteilte Softwareentwicklungsumgebung unterstützt werden soll.

Wie in Abschnitt 5.4 beschrieben, wird für die Integration von HeRA und Pia ein asymmetrischer Ansatz gewählt, bei dem die Prozessinfrastruktur auf den Helfer- und Ressourcenagenten aus HeRA aufsetzt und sie an verschiedenen Stellen im System verwendet. Dafür müssen eine Reihe von Helfern entwickelt werden, die die verschiedenen Nutzerrollen innerhalb des WfMS darstellen (Ausführen von Aufgaben, Prozessdefinition, Nutzer- und Rechteverwaltung), sowie spezialisierte Helfer für die verschiedenen Aufgaben innerhalb des Prozesses. Das lässt sich darstellen anhand der verschiedenen Schnittstellen eines WfMS (vgl. Abb. 3.17 auf Seite 78).

7.3.1.1 Definition von Prozessen

Die erste Schnittstelle eines WfMS dient der Definition von Prozessen. Zum Teil können Prozesse als Teil einer Entwicklungsumgebung vordefiniert werden, viele Prozesse können jedoch nicht einmal und endgültig definiert werden, sondern müssen ad-hoc auf die jeweilige Situation angepasst werden. Daher ist es wichtig, dem Benutzer die Möglichkeit zu geben, eigene Prozessdefinitionen zu erstellen und im System ausführen zu lassen.

Benutzer, die die entsprechende Berechtigung besitzen, können daher über einen Helferagenten zur Workflow-Definition neue Prozessbeschreibungen erzeugen und ins System laden. Die Prozessbeschreibungen werden als Ressource erzeugt und in der WFDDB gespeichert. Von dort können sie dann abgerufen und instanziiert werden.

7.3.1.2 Helfer für Benutzeraufgaben

Die Aufgaben, die im Laufe eines Workflows bearbeitet werden müssen, werden über zwei verschiedene Schnittstellen abgehandelt: Die Workflow Client Application Schnittstelle dient dazu, Workflowteilnehmern Aufgaben zuzuweisen

und sie mit den Informationen zu versorgen, die sie zur Bearbeitung der jeweiligen Aufgabe benötigen. Die Invoked Applications Schnittstelle hingegen ist verantwortlich für Aufgaben, die nicht von einem Benutzer ausgeführt werden, sondern von einem Systemprozess.

Wenn ein Teilnehmer eine Aufgabe zur Bearbeitung annimmt, so bekommt er über die Workflow Client Schnittstelle die Informationen zur Bearbeitung. Darüber hinaus kann die Prozessinfrastruktur den Bearbeiter aber auch mit einem kompletten Helferagenten versehen, der zur Bearbeitung verwendet werden kann. Wenn dies in der Workflowdefinition vorgesehen ist, wird zunächst ein Helferagent erzeugt und mit den Daten zur Aufgabe initialisiert. Dieser sogenannte Aktivitätsagent wird dann mit dem Benutzeragenten des Bearbeiters verbunden und erlaubt diesem dann, die Aufgabe zu erledigen. Mit den Ergebnissen der Bearbeitung kehrt der Aktivitätsagent dann wieder zum WfMS zurück.

7.3.1.3 Aufgerufene Anwendungen

Die Invoked Applications Schnittstelle ähnelt der Workflow Client Schnittstelle insofern, dass beide für die Bearbeitung von Aufgaben im Workflow zuständig sind. Aufgerufene Anwendungen benötigen allerdings keine Benutzerinteraktion, sondern sind lediglich Service-Aufrufe. Daher muss hier kein Helferagent konstruiert werden, sondern lediglich der Aufruf mit den richtigen Parametern an einen passenden Diensteanbieter übergeben werden. Dieser Dienstanbieter kann die Form eines Automatenagenten (vgl. Abschnitt 3.5) annehmen.

Als Spezialfall kann der aufgerufene Dienst wiederum ein Workflow sein. Auf diese Weise können Subflows realisiert werden, unter Umständen auch transparent für den aufrufenden Workflow.

7.3.1.4 Entfernte WfMS

Über diese Schnittstelle können entfernte WfMS angesprochen werden, um einen Prozess auszuführen, der sich über mehrere WfMSe erstreckt. Dabei geht es nicht einfach um untergeordnete Teilprozesse, diese können wie oben ausgeführt über die Invoked Applications Schnittstelle behandelt werden. Vielmehr geht es bei dieser Schnittstelle um die Fragmentierung und verteilte Ausführung kompletter Workflows.

Zur Behandlung dieser Thematik können beispielsweise Workflow Fragmente [Carl, 2004], Workflowagenten [Reese et al., 2005] oder Workflow Strukturagenten [Wagner, 2009c] verwendet werden.

7.3.1.5 Administration und Monitoring

Das Starten und Überwachen von Workflowinstanzen wird über diese Schnittstelle behandelt. Hierfür werden verschiedene Helfer und Interaktionen be-

nötigt. Das Starten eines neuen Workflows ist eine Aktion, die verschiedene Helfer ausführen können, je nachdem, welche Art von Prozess sie starten. Beispielsweise kann ein Helferagent zum Testen einer Software automatisch einen neuen Fehlerbehebungsworkflow starten, wenn sich im Test ein Fehler ergibt. Diese Helfer können dann entweder auch die von ihnen gestarteten Prozesse überwachen, oder man verwendet einen speziellen Prozessüberwachungshelfer.

Auch das Auditing von Prozessen, beispielsweise zur Verifikation der Einhaltung von Business-Regeln, fällt in diese Kategorie (vgl. hierzu auch [VAN HEE et al., 2010]), ebenso wie die Steuerung von Workflows. Zum außerplanmäßigen Abbrechen, Anhalten oder Verändern von Workflows benötigt man einen spezialisierten Helfer. Es ist natürlich wichtig, die entsprechenden Berechtigungen zu berücksichtigen, da nicht jeder Benutzer jeden Prozess überwachen oder gar steuern kann, selbst wenn er ihn selbst gestartet hat.

7.3.2 Agenten

Für das POTATO-System sind eine Reihe von spezialisierten Helferagenten erforderlich, die in die Struktur von PIA integriert werden. Im ursprünglichen PIA-System wurde ein Benutzeragent implementiert, der alle Interaktionen des Benutzers mit dem WfMS behandelt. Wie angesprochen führt das zu Problemen mit Workflows, die neue Funktionen benötigen, da diese jeweils individuell dem Benutzeragenten hinzugefügt werden müssten.

Stattdessen werden in POTATO alle Funktionen, die ein Benutzer im WfMS ausführt, durch zu diesem Zweck entwickelte Helferagenten ausgeführt. Für den Zugriff eines Workflow-Teilnehmers auf seine Aufgabenliste wird ein Workflow-Client Helferagent verwendet. Aufgaben im Workflow werden je nach Aufgabentyp durch spezialisierte Aktivitätsagenten bearbeitet. Administrationsaufgaben erfordern ebenfalls spezielle Helferagenten.

7.3.2.1 Workflow-Client Helfer

Der Workflow-Client Agent ist ein Helferagent, der auf das agentenorientierte WfMS zugreift. In den Interaktionen der Prozessinfrastruktur nimmt dieser Agent die Rolle des Benutzers ein, speziell die Rollen des Workflow Initiators und des Executors. Gegenüber dem Benutzeragenten verhält er sich wie ein Helferagent.

Zu diesem Zweck erhält der Agent alle Protokolle, die die Benutzerrolle im WfMS besitzt, zusätzlich die Helferagentenprotokolle. Anstelle der Benutzer-GUI aus PIA wird eine Helferagenten-GUI verwendet, die sich in einen HERA-Benutzeragenten integrieren lässt.

Wird der Helfer gestartet, verwendet er das `Login`-Protokoll, um sich am WfMS anzumelden und registriert sich beim Workitem Dispatcher für Worklist-Updates. Die GUI erlaubt eine Übersicht über die Aufgabenliste, aus der Aufgaben zur Bearbeitung ausgewählt werden können.

Für die Ausführung dieser Aufgaben registriert der Workflow-Client die verwendeten Activity-Agenten bei seinem eigenen Benutzeragenten. Sind die Aufgaben fertig bearbeitet oder werden sie abgebrochen, leitet der Workflow-Client die entsprechenden Nachrichten an das WfMS weiter.

7.3.2.2 Activity-Agent

Aktivitätsagenten oder Activity-Agenten sind eine spezielle Klasse von Helferagenten, die in der Prozessinfrastruktur verwendet werden. Sie können innerhalb von Workflows für die Bearbeitung einer Aufgabe verwendet werden. Im Gegensatz zu normalen Helferagenten muss ein Aktivitätsagent nicht explizit durch einen Benutzeragenten angefordert werden, sondern er wird erzeugt, wenn ein Benutzer eine Aufgabe zur Bearbeitung anfordert [Markwardt et al., 2009b].

Durch die Verwendung von Aktivitätsagenten können die Aufgaben, die in einem Workflow verwendet werden, flexibel gestaltet sein. Für einen neuen Typ von Aufgabe muss nicht der Benutzeragent angepasst werden, sondern es muss lediglich ein neuer Typ von Helferagent in die Helferfabrik eingefügt werden, der dann von Workflows referenziert werden kann.

Um zur Bearbeitung einer Aufgabe verwendet werden zu können, ist der Activity-Agent vorkonfiguriert mit Kontextdaten aus dem Workflow, die zur Bearbeitung der aktuellen Aufgabe benötigt werden. Er wird in den Arbeitsplatz des Benutzers eingepasst, um die Aufgabe zu bearbeiten und verlässt ihn wieder, wenn die Bearbeitung abgeschlossen ist. Er besitzt einige weitere Protokolle zusätzlich zum normalen Helferagenten, die wiederum gemeinsam sind für alle Arten von Activity-Agenten.

Wenn bei der Workflow Engine eine Aufgabe im Workflow angefordert wird, die einen Activity-Agenten erfordert, kontaktiert sie die Helferfabrik, um den neuen Agenten zu erzeugen. Die Adresse des Activity-Agenten wird dem Workflow-Client in der Aufgabenbeschreibung mitgegeben. Dieser veranlasst dann, dass sich der Activity-Agent beim Benutzeragenten als Helferagent registriert. Beim Beenden des Activity-Agenten wird die Aufgabe über den Workflow-Client als erledigt oder abgebrochen gekennzeichnet.

7.3.2.3 Workflowbearbeitungs-Helfer

Für die Bearbeitung von Workflow-Definitionen wird ein Helferagent benötigt, der es einem Benutzer erlaubt, alle Elemente zu spezifizieren, die für die Instanziierung und Ausführung eines Workflows erforderlich sind.

Dies ist zum einen die Definition des Workflow-Petrinetzes, das den Ablauf und die Abhängigkeiten der Aufgaben beschreibt. Dies lässt sich mit einem Renew-Editor bearbeiten. Dabei kann auch die Soundness des Netzes überprüft werden. Handelt es sich um die Bearbeitung eines vorhandenen Workflows, und sollen existierende Instanzen auf die geänderte Version umgestellt

werden, so muss zusätzlich eine Übertragungsvorschrift erstellt werden (vgl. Abschnitt 7.4).

Zum anderen müssen die verwendeten Aufgaben spezifiziert und über Regeln den gewünschten Rollen zur Bearbeitung zugewiesen werden. Für die Aufgaben muss festgelegt werden, wie diese bearbeitet werden sollen. Dazu können die im System vorhandenen Helfer- und Ressourcenagenten herangezogen werden.

Der Workflowbearbeitungs-Helfer speichert dann die neuen Workflowdefinitionen beim Workflow Definition Database Agenten ab. Der sorgt dafür, dass für neu angelegte Workflows die neuen Prozessdefinitionen verwendet werden.

7.3.2.4 Workflowadmin-Helfer

Dieser Helfer erlaubt es, verschiedene administrative Funktionen innerhalb des WfMS vorzunehmen. Für den reibungslosen Betrieb einer WfMS-basierten Anwendung ist es unerlässlich überprüfen zu können, welche Workflowinstanzen gerade aktiv sind und in welchem Status sie sich befinden. Diese Informationen können vom WFES-Agenten abgefragt werden und liefern dem Nutzer hilfreiche Informationen über den Zustand des Systems.

Noch kritischer als das Monitoring der Prozesse ist der direkte Eingriff in den Ablauf eines Workflows. Aus verschiedenen Gründen kann es während der Ausführung eines Prozesses erforderlich sein, Änderungen an der Markierung eines Workflownetzes vorzunehmen. Dies kann erforderlich sein, weil eine Aufgabe versehentlich vorschnell abgeschlossen wurde oder im System nicht abgeschlossen werden kann, weil eine Verzweigung im Workflow fehlerhaft ausgewählt wurde etc. In all diesen Fällen kann es erforderlich sein, manuell in den Workflow einzugreifen und Marken innerhalb des Netzes zu verschieben.

An dieser Stelle werden die potentiellen Konsequenzen bzgl. der Soundness eines Workflows, die sich durch direkte Eingriffe ergeben können, nicht weiter betrachtet. Da es sich immer um Einzelfälle handelt, muss jeweils speziell dafür Sorge getragen werden, dass die Prozesse nicht fehlerhaft geändert werden. Bei Bedarf kann durch geeignete Verfahren überprüft werden, ob eine manuell eingestellte Markierung im Netz erreichbar ist, bzw. ob sie zu Verklemmungen führen kann.

Üblicherweise sind nur wenige Benutzer berechtigt, direkte Eingriffe in den Workflow vorzunehmen. Die Änderung wird durch den Workflowadmin-Helfer angestoßen, der die zuständige WFEngine damit beauftragt, die entsprechenden Änderungen vorzunehmen. Auswirkungen, die sich daraus bezüglich aktivierter Aufgaben etc. ergeben, werden durch die `TaskOccurrenceListener` erfasst und durch die üblichen Protokolle behandelt.

7.3.3 Interaktionen

Für die Verwendung von Helfer- und Ressourcenagenten in PIA sind eine Reihe von Interaktionen erforderlich, die im Folgenden beschrieben werden. Nicht alle diese Interaktionen sind in dem Prototypen realisiert, der in Kapitel 8 beschrieben wird. Sie werden dennoch hier angegeben, da sie Teil eines kompletten POTATO-Systems sind und verdeutlichen, wie Helfer- und Ressourcenagenten die Prozessinfrastruktur ergänzen.

7.3.3.1 Anmelden am WfMS

Bei POTATO meldet sich der Benutzer nicht direkt am WfMS an, sondern der Workflow-Client Helfer übernimmt das für ihn. Der Benutzeragent fordert den Workflow-Client als neuen Helferagenten an und registriert ihn, daraufhin führt der Helferagent die Benutzerinteraktionen zur Anmeldung am WfMS aus. In der aktuellen Version erfordert lediglich das WfMS eine Benutzeranmeldung, eine Gesamtanmeldung auch für den Zugriff auf die Helferagenten ist aber für Produktivanwendungen ebenfalls erforderlich.

7.3.3.2 Workflowaufgabe bearbeiten

Die wichtigste Änderung, die sich durch die Verwendung von Helferagenten in der Prozessinfrastruktur ergibt, ist die Verwendung von Activity-Agenten bei der Bearbeitung von Aufgaben. Den Ablauf hierbei zeigt Abb. 7.3.

Der Benutzer fordert eine Aufgabe zur Bearbeitung an, indem er den entsprechenden Menupunkt im Workflow-Client Helferagenten auswählt. Dieser sendet, wie üblich, eine `request(assignWorkitem)`-Nachricht an den Workitem Dispatcher, der sie über den WFES an die WFEngine weiterleitet. Der Übersichtlichkeit halber wurden der Workitem Dispatcher und der WFES Agent in der Abbildung weggelassen, da dieser Teil bereits behandelt wurde und sie die Nachricht lediglich weiterleiten.

Die WFEngine erstellt die neue Aktivität und überprüft, ob ein Acitivityagent erzeugt werden muss. Ist das der Fall, wird das Protokoll zum Erzeugen eines Activity-Agenten gestartet, das eine Nachricht mit den Details des benötigten Agenten an die Helferfabrik sendet. Diese erzeugt wie üblich den neuen Agenten, indem die Wissensbasis konstruiert und an den AMS-Agenten der Plattform gesendet wird. In diesem Fall werden der Executor (der Workflow-Client Agent) und die Aktivität als Parameter übergeben und direkt bei der Erzeugung in die Wissensbasis des Agenten geschrieben, so dass dieser bereits beim Start weiß, für welche Aufgabe er erzeugt worden ist.

Die Helferfabrik sendet die Adresse des neuen Helferagenten an die WFEngine. Diese fügt die Adresse in die Beschreibung der Aktivität ein und schaltet die Request-Transition der Task-Transition, die der Aufgabe zugeordnet ist. Dadurch wird der Activity-Agent in das Workflownetz eingebracht. Anschließend bestätigt die WFEngine die Anforderung der Aufgabe.

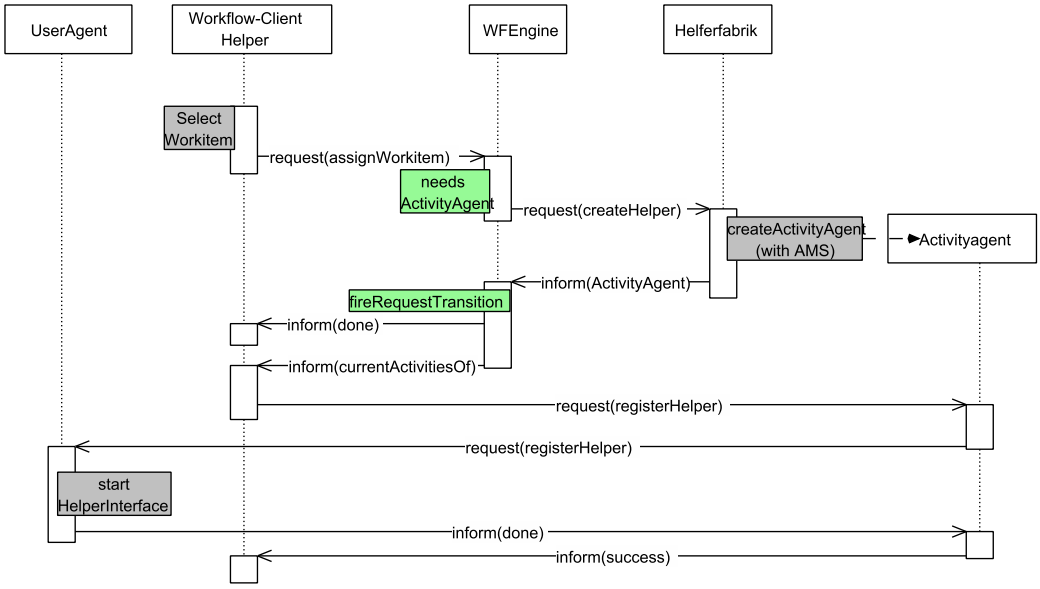

Abbildung 7.3: Interaktion: Anforderung eines Activity-Agenten

Das Schalten der Task-Transition verursacht ein Update der Aktivitäten des Workflows, so dass durch WFES und Workitem Dispatcher eine neue Aktivitätenliste an den Workflow-Client Agenten gesendet wird. Dieser erhält die neue Aktivität und stellt fest, dass sie einen Activity-Agenten enthält. Der Workflow-Client weist nun den Activity-Agenten an, sich beim Benutzeragenten als Helfer zu registrieren. Der Benutzeragent integriert den neuen Helfer in seine Oberfläche, so dass der Anwender mit der Bearbeitung der Aufgabe beginnen kann.

7.3.3.3 Aktivität abschließen

Ein Benutzer kann die Bearbeitung einer Aufgabe abschließen oder abbrechen. Wie dies im Detail in der Benutzungsoberfläche realisiert ist, ist Sache des Activity-Agenten. In den implementierten Beispielen öffnet der Activity-Agent einen Editor, bei dessen Schließen der Benutzer gefragt wird, ob die Aufgabe abgeschlossen oder abgebrochen werden soll.

Der Agent erzeugt ein `SetActivityDone`- bzw. `SetActivityCanceled`-Objekt, in dem er die Aktivität und den Executor aus seiner Wissensbasis einträgt. Wenn die Aktivität erfolgreich beendet wird, kann er auch noch ein Ergebnis hinzufügen, beispielsweise in Form eines Ressourcenagenten, das dann im Workflow weiter verwendet werden kann.

Die Nachricht sendet der Activity-Agent zunächst an den Workflow-Client Agenten, den er als Executor der Aktivität in seiner Wissensbasis stehen hat, da dieser für die Kommunikation mit dem WfMS verantwortlich ist. Das weitere Abschließen der Aktivität wird dann wie im normalen PIA-System gehand-

habt. Wenn der Workflow-Client die Bestätigung des Abschlusses erhalten hat, teilt er dies dem Activity-Agenten mit, der sich dann beim Benutzeragenten deregistriert.

7.3.3.4 Neue Workflowdefinition erstellen

Um eine neue Workflowdefinition zu erstellen, verwendet der Workflow-Designer einen Workflowbearbeitungs-Helferagenten. Er erzeugt zunächst eine neue Workflowdefinitions-Ressource und unterstützt den Benutzer dann bei deren Bearbeitung. Das Workflownetz kann mit Hilfe von Renew bearbeitet werden, zusätzlich können Workflowparameter konfiguriert und Ressourcen hinzugefügt werden.

Die fertige Definition wird dann an den Workflowdefinitions-Agenten gesendet. Dieser prüft den Workflow auf formale Korrektheitskriterien (vgl. Abschnitt 3.1.2.1) und speichert ihn in seiner Datenbank ab.

7.3.3.5 Workflowdefinition bearbeiten

Auch die Bearbeitung einer Workflowdefinition erfolgt über den Workflowbearbeitungs-Helferagenten. Er fordert zunächst vom Workflowdefinitions-Agenten die Ressource mit der bisherigen Definition an, die bearbeitet werden soll.

Bei der Bearbeitung gibt es zwei Möglichkeiten: Bereits existierende Instanzen der bearbeiteten Workflowdefinition können unverändert weiterlaufen, und nur neu erstellte Instanzen folgen der neuen Definition, oder bereits laufende Prozesse werden auf die veränderte Definition umgestellt.

Der erste Fall ist trivial, die Bearbeitung unterscheidet sich in diesem Fall nicht von der oben beschriebenen Erstellung einer neuen Workflowdefinition. Sollen vorhandene Instanzen aber angepasst werden, so muss bei jeder Veränderung, die am Workflow vorgenommen wird, festgelegt werden, wie sich diese auf vorhandene Instanzen auswirkt.

Sollen bereits vorhandene Instanzen des bearbeiteten Workflows auf die neue Version umgestellt werden, muss eine Übertragungsvorschrift aufgenommen werden. Insbesondere Änderungen am Workflownetz müssen genau dokumentiert werden, damit nicht durch die Umstellung des Prozesses ein ungültiger Workflow entsteht. In [Markwardt et al., 2008a] wird eine Methode diskutiert, mit der die Soundness auch bei einer Workflowänderung zur Laufzeit sichergestellt werden kann (siehe auch Abschnitt 7.4.

Ist die Bearbeitung abgeschlossen, wird die neue Workflowdefinition zusammen mit den Migrationsvorschriften an den Workflowdefinitions-Agenten geschickt. Dieser prüft nun seinerseits die Soundness der neuen Definition und der Übertragungsvorschrift. Wenn hierbei ein Fehler gefunden wird, wird der Bearbeitungsagent darüber informiert und kann entsprechende Korrekturen vornehmen.

Ist die Bearbeitung fehlerfrei, wird die alte Workflowversion archiviert und die neue Version aktiviert. Wenn laufende Instanzen der alten Definition auf die neue Definition umgestellt werden müssen, fragt der Workflowdefinitions-Agent beim Workflow Enactment Service an, welche Instanzen dies betrifft und informiert die betroffenen Workflow Engines über die Änderungen. Diese tauschen dann die Netze aus und führen die Übertragungsvorschrift durch.

7.4 Workflowmodifikation

Für die Untersuchung von Workflowprozessen existiert mit der Klasse der Workflownetze (vgl. Abschnitt 3.1.2) ein gut untersuchter Formalismus. Anhand dieses Modells kann die Soundness einer Prozessdefinition überprüft werden [Aalst, 1997]. Daher wird dieser Test durch den Workflow Definition Database Agenten durchgeführt, bevor eine neue Workflowdefinition zur Ausführung akzeptiert wird.

Dieser Test ist aber nur für die Prüfung von Prozessdefinitionen vor der Ausführung geeignet. Das Potato-System soll es aber erlauben, Workflowdefinitionen auch zur Laufzeit anzupassen, um so möglichst flexible Prozesse zu erzielen. Das Verfahren hierfür wird beschrieben in [Markwardt et al., 2008a].

Prinzipiell ist dieses Verfahren auch anwendbar auf allgemeine Agentenprotokolle. Über die Kombination von Protokollbausteinen zu neuen Protokollen können in Mulan Agentenprotokolle zur Laufzeit verändert werden [Rölke, 2004, S. 142ff]. Die Untersuchung von Agenteninteraktionen über Workflow-Soundness wird in [Lehmann und Moldt, 2004] beschrieben und kann in Verbindung mit dem hier beschriebenen Verfahren auch auf Veränderungen von Protokollen zur Laufzeit angewandt werden.

Ebenfalls verwandt mit dem Themenbereich der Workflowprozesse und Agentenprotokolle ist die Untersuchung von Diensten in serviceorientierten Architekturen (siehe Abschnitt 4.2.2). Verschiedene Ansätze dazu finden sich beispielsweise in [Ryu et al., 2008, Liske et al., 2009].

Workflows adaptiv zu gestalten und zur Laufzeit an veränderte Anforderungen anpassen zu können, ist auch ein Ziel des ADEPT$_{flex}$-Systems [Reichert und Dadam, 1998, Reichert et al., 2003]. Darin lassen sich die Struktur, Attribute oder der Status von Workflowinstanzen modifizieren und beispielsweise Tasks hinzugefügt oder entfernt werden. Das zugrundeliegende Framework $ADEPT_{base}$ verwendet ein graphbasiertes Prozessmodell [Reichert und Dadam, 1997].

7.4.1 Idee

Mit Hilfe dieses Verfahrens ist es möglich, eine gültige Übertragung von einem korrekten Workflownetz in ein anderes zu erzielen, so dass jeder laufende Work-

flowprozess, der von einer Definition auf die andere umgestellt wird, ebenfalls korrekt abläuft. Voraussetzung hierfür ist neben der Angabe des Ausgangsnetzes und der geänderten Version eine Übertragungsvorschrift, die den Übergang von einer Version zur anderen regelt.

7.4.2 Verfahren

Ziel des Verfahrens ist es, für eine Workflowmodifikation zu prüfen, ob sie für alle möglichen laufenden Workflowinstanzen weiterhin sound ist. Das bedeutet (vgl. Abschnitt 3.1.2.1), dass von jeder erreichbaren Markierung, unabhängig davon, zu welchem Zeitpunkt die Workflowdefinition auf die geänderte Version umgestellt wurde, der Endzustand erreicht werden kann und dass die Markierung in der genau eine Marke in der Endstelle liegt, die einzige Markierung ist, bei der diese Stelle belegt ist und die einzige Endmarkierung des Netzes.

Die Eingabe besteht daher aus:

- Ausgangsworkflownetz $WN_A = (P_A, T_A, F_A, i_A, o_A)$
- Modifiziertes Workflownetz $WN_B = (P_B, T_B, F_B, i_B, o_B)$
- Übertragungsnetz $WN_{Trans} = (P_{Trans}, T_{Trans}, F_{Trans}, i_{Trans}, o_{Trans})$

Die Workflownetze WN_A und WN_B müssen sound sein. Wäre WN_A nicht sound, hätte es überhaupt nicht erst verwendet werden können, analog für WN_B. Dies allein genügt allerdings nicht, da es denkbar ist, dass ein Workflow, der zunächst nach WN_A abläuft, um dann auf WN_B zu wechseln, dabei in einen inkonsistenten Zustand läuft, der eine ordnungsgemäße Termination verhindert. Das Übertragungsnetz PN_{Trans} beschreibt daher den Übergang zwischen den beiden Workflownetzen:

Definition 3 (Workflow-Übertragungsnetz)
Ein Workflow-Übertragungsnetz zu zwei Workflownetzen WN_A und WN_B ist ein Workflownetz

$WN_{Trans} = (P_{Trans}, T_{Trans}, F_{Trans}, i_{Trans}, o_{Trans})$
$P_{Trans} = P_A \uplus P_B \uplus P_t$
$T_{Trans} = T_A \uplus T_B \uplus T_t \cup t_{iA} \cup t_{oA} \cup t_{iB} \cup t_{oB}$

Wobei P_t und T_t die Stellen und Transitionen eines Verbindungsnetzes darstellen für die Überführung von WN_A nach WN_B. Die Kantenrelation ergibt sich als

$F_{Trans} = F_A \uplus F_B \uplus F_{trans}$

F_{trans} ist eine Menge von Transformationskanten und ergänzenden Kanten, um ein vollständiges Workflownetz zu formen

$F_{trans} \subset ((P_A \cup P_t) \times T_t) \cup (T_t \times (P_t \cup P_B))$
$\uplus\{(i_{trans}, t_{iA}), (i_{trans}, t_{iB}), (t_{iA}, i_A), (t_{iB}, i_B),$
$(o_A, t_{oA}), (o_B, t_{oB}), (t_{oA}, o_{trans}), (t_{oB}, o_{trans})\})$

mit $\forall p_a \in P_A : \exists (p_a, t_t), (t_t, p_t) \in F_{trans}$ mit $\cdot t_t = \{p_a\}, \cdot p_t = \{t_t\}$ (Von jeder Stelle in WN_A geht eine Übertragungskante zu einer zugehörigen Stelle in P_t aus, so dass die Übertragung zu jedem Zeitpunkt durchgeführt werden kann.)

Mit Hilfe dieses Übertragungsnetzes kann nun die Soundness einer Workflow-Übertragung auf die Soundness des Übertragungsnetzes zurückgeführt werden.

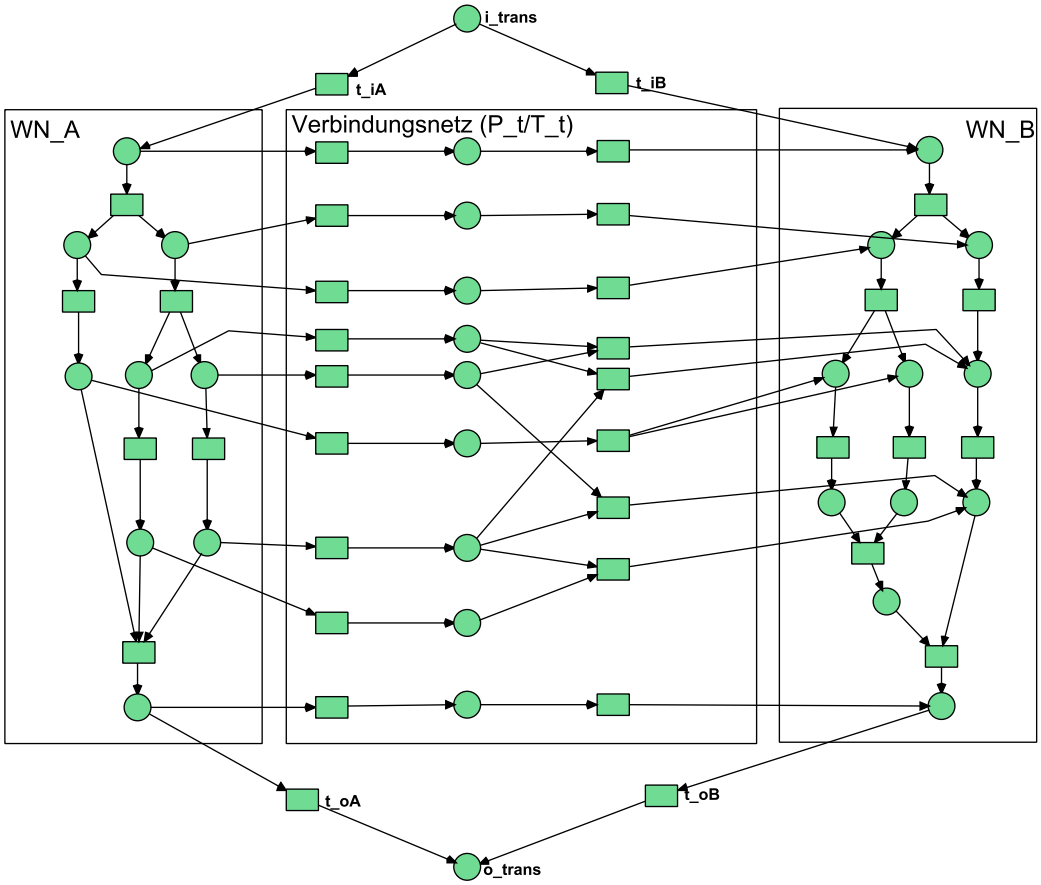

Abbildung 7.4: Workflow Übertragungsnetz

Abb. 7.4 veranschaulicht die Konstruktion des Übertragungsnetzes. In diesem Beispiel ist die Übertragung recht einfach, so dass $P_t = \emptyset$, für komplexere Übertragungen könnte der mittlere Teil, der der Übertragung dient, aber auch ausführlicher ausfallen. Zu beachten ist, dass Kanten nur aus WN_A zum Übertragungsnetz und von diesem zu WN_B verlaufen, sowie innerhalb der jeweiligen Teilnetze.

Bei der Übertragung eines Workflows von einer Definition auf die andere wird der Workflow quasi „eingefroren“, indem alle Kanten aus WN_A entfernt werden. Durch die Kanten in WN_{trans} erfolgt dann die Umstellung auf die

geänderte Workflowdefinition, wo der Prozess dann weiterläuft.

Definition 4 (Soundness einer Workflow-Übertragung)
Eine Workflow-Übertragung WN_{Trans} (nach obiger Definition) ist sound, wenn für jeden Workflow, der nach WN_A gestartet wurde, zu einem beliebigen Zeitpunkt mittels der Kanten in F_{trans} auf die Workflowdefinition nach WN_B umgestellt und dort weitergeführt wurde, die Soundness-Bedingungen erfüllt sind:

- Von jeder erreichbaren Markierung ist eine Endmarkierung $m_o \geq \{o_{trans}\}$ erreichbar (Workflow terminiert).
- Die einzige erreichbare Markierung $m_o \geq \{o_{trans}\}$ ist die Markierung $m_o = \{o_{trans}\}$ (Endmarkierung ist eindeutig; keine Marken bleiben liegen).

Korollar 1
Eine Workflow-Übertragung ist sound, wenn das Workflow-Übertragungsnetz, sowie das Ausgangs- und Endnetz sound sind.

Korrektheitsnachweis 1
Schritt 1: Von jeder erreichbaren Markierung ist eine Endmarkierung erreichbar. Eine Schaltfolge eines transformierten Workflows besteht stets aus drei Teilfolgen: Zuerst schaltet der Workflow nach der Ausgangsdefinition, dann erfolgt irgendwann die Übertragung, schließlich schaltet der Workflow nach der neuen Definition. Diese Schaltvorgänge lassen sich parallel in WN_{Trans} vollziehen, da dieses Netz die gleichen Stellen, Transformationen und Kanten enthält. Durch die Phaseneinteilung und zeitweise Deaktivierung von Kanten sind bei einer Workflowumstellung nicht alle Schaltvorgänge möglich, die in WN_{Trans} möglich sind, umgekehrt können aber alle Schaltungen nachvollzogen werden.

In Phase 1 kann es keine Verklemmungen geben, da WN_A sound ist. Die Schaltvorgänge, die in den Übertragungsteil führen, können jederzeit ausgeführt werden, da die entsprechenden Transitionen keine weiteren Nebenbedingungen haben. Folglich sind diese Markierungen in WN_{Trans} erreichbar. Daraus folgt aufgrund der Soundness von WN_{Trans}, dass von hier wiederum eine Endmarkierung erreichbar ist.

Schritt 2: Die Endmarkierung ist eindeutig. Jede Markierung, die bei der Übertragung eines Workflows erreicht werden kann, ist auch in WN_{Trans} erreichbar. Da WN_{Trans} sound ist, ist in der Erreichbarkeitsmenge keine weitere Endmarkierung $m'_o > \{o_{trans}\}$ enthalten.

7.4.3 Anwendung

Das beschriebene Verfahren erlaubt die Modifikation von Workflowdefinitionen innerhalb eines Workflow Management Systems zur Laufzeit, da die Überprüfung der Soundness-Eigenschaft automatisiert durchgeführt werden kann.

Voraussetzung hierfür ist aber, dass das verwendete Werkzeug die benötigten Informationen erhebt. Es kann nicht einfach eine beliebige Workflowdefinition auf eine andere umgestellt werden und mit Hilfe des Algorithmus geprüft werden, ob das Ergebnis sound ist. Vielmehr muss explizit ein Übertragungsnetz konstruiert werden, das angibt, auf welche Weise laufende Workflowinstanzen umgestellt werden. Nur mit diesem kann die Gültigkeit überprüft werden.

Wird das bei der Konstruktion eines Werkzeuges beachtet, ist das Problem aber verhältnismäßig gering. In den seltensten Fällen wird ein Workflow komplett verändert, meist sind es nur kleinere, inkrementelle Änderungen, die sich durch neue Anforderungen ergeben, etwa eine hinzugefügte oder entfallende Aktivität, eine Veränderung der Reihenfolge, Parallelisierung oder Sequentialisierung etc. Der Großteil des Workflows, der unverändert bleibt, kann dann „trivial“ umgestellt werden, indem jede Stelle auf die ihr entsprechende, unveränderte Stelle im neuen Workflow überführt wird.

Lediglich die Stellen, an denen tatsächlich eine Änderung vorkommt, müssen dann explizit behandelt werden. Für häufig vorkommende Änderungsmuster könnte das Werkzeug hier sogar Übertragungsmuster vorschlagen. Die Überprüfung der Gültigkeit kann dann effektiv automatisiert vorgenommen werden.

7.5 Zusammenfassung

In diesem Kapitel wurde diskutiert, wie Multiagentensysteme für die Durchführung und Steuerung von Geschäftsprozessen verwendet werden können. Dafür wurde zunächst die agentenorientierte Prozessinfrastruktur Pia beschrieben, die im Kontext dieser Arbeit, aber ursprünglich unabhängig von HeRA entwickelt worden ist. Anschließend wurde ihre Integration mit den Helfer- und Ressourcenagenten aus dem HeRA-System beschrieben.

Pia ist eine agentenorientierte Umsetzung des WfMC Referenzmodells für Workflow Management Systeme. Die verschiedenen Komponenten dieses Modells sind als Agenten implementiert, die durch ihre Interaktion den Dienst eines WfMS erbringen. Das System verwendet Workflow-Referenznetze für die Modellierung und Ausführung der Prozesse.

Durch die Verwendung des HeRA-Systems in Pia wird das System anschließend erweitert. Helfer- und Ressourcenagenten als Bausteine in Pia erlauben eine höhere Flexibilität der Prozessinfrastruktur. Einheiten, die vorher fester Bestandteil des WfMS waren, können nun als eigenständige Agenten dargestellt werden, die autonom, flexibel und adaptiv handeln können. Verschiedene Anwendungsbereiche dieser Flexibilisierung wurden dargestellt.

Ein Beispiel für die Flexibilisierung ist die Entkopplung der Benutzer vom WfMS. Im ursprünglichen Pia-System musste jeder Benutzer, der als Teilnehmer das System nutzen wollte, über die hierfür erforderliche Benutzungsschnittstelle und die richtigen Protokolle verfügen. Stattdessen wird in Potato

ein Workflow-Client Helferagent verwendet, der gegenüber der Prozessinfrastruktur die Rolle des Teilnehmers einnimmt und den Benutzeragenten um die benötigte Funktionalität ergänzt.

Der Workflow-Client Helfer muss aber nicht alle Aktivitäten kennen, die in den verschiedenen Workflowdefinitionen vorkommen können, wie das in PIA noch gefordert war. Stattdessen können die Aktivitäten ihrerseits durch spezialisierte Helferagenten, Activity-Agenten genannt, ausgeführt werden. Diese Helferagenten, sowie eventuell im Workflow benötigte Ressourcen, werden dem Workflow-Client durch die Prozessinfrastruktur zur Verfügung gestellt und im Benutzeragenten ausgeführt. Dies erlaubt zusätzliche Flexibilität bei der Bearbeitung der Aufgaben und der Definition neuer Workflows, da neue Aufgaben- und Ressourcentypen dem System hinzugefügt und in Prozessdefinitionen verwendet werden können, ohne dass dafür der Benutzeragent oder der Workflow-Client Helfer angepasst werden müssten.

Eine höhere Flexibilität bei der Ausführung von Workflows erlaubt ein Verfahren zur Modifikation von Workflowdefinitionen zur Laufzeit. Dadurch können die vorgegebenen Prozesse von entsprechend berechtigten Benutzern bei Bedarf an die tatsächlichen Erfordernisse angepasst werden. Damit wird verhindert, dass zu starre Prozessvorgaben die Benutzern bei der Arbeit behindern. Prozesse können durch stetige Korrekturen optimal an die Erfordernisse angepasst werden.

Kapitel 8

Prototypen

In diesem Kapitel wird die prototypische Implementierung des POTATO-Systems beschrieben. Dabei wurde der Schwerpunkt nicht darauf gelegt, ein funktional vollständiges Framework oder eine komplette Anwendung zu entwickeln. Vielmehr werden die in der Arbeit aufgezeigten Konzepte und die Architektur anhand von praktischen Beispielen veranschaulicht. Dabei werden die Designentscheidungen aufgezeigt, die an verschiedenen Stellen getroffen wurden, um das in den vorhergehenden Kapiteln entwickelte Konzept umzusetzen. Daher wurden nicht alle Agenten entwickelt, die man für ein vollständiges System benötigen würde. Manche Aspekte wurden auch bewusst komplett weggelassen.

Zu diesem Zweck wurden eine Reihe von Prototypen entwickelt, die jeweils bestimmte Aspekte des Systems beleuchten und die teilweise aufeinander aufbauen. Dies ermöglicht es, die verschiedenen Bestandteile von POTATO einzeln vorzustellen und aufzuzeigen, wie die einzelnen Teile für sich gesehen bereits eine praktische Anwendbarkeit besitzen. Dies mündet schließlich in einem Prototypen, der die verschiedenen Aspekte des POTATO-Systems in sich vereint und die Ausführung eines einfachen Workflows aus dem Softwareentwicklungsumfeld mit unter Verwendung verschiedener Helferagenten ermöglicht.

Ein wichtiger Aspekt verteilter Anwendungen etwa, der in dieser Arbeit vollständig ausgeklammert wird, ist der Aspekt der Sicherheit. Eine Benutzerverwaltung wird da verwendet, wo sie für die Funktionalität erforderlich ist, etwa für die Zuweisung von Aufgaben an die richtigen Benutzer. Die Prüfung von Berechtigungen wird aber vorerst komplett ausgeklammert. Natürlich müssen in einem produktiven System an allen Stellen die Berechtigungen geprüft werden, ob ein Benutzer beispielsweise einen Helfer verwenden, eine Ressource verändern oder einen Workflow administrieren darf. Aus Übersichtlichkeitsgründen und um die Implementation zu vereinfachen, wird darauf aber hier verzichtet.

8.1 Überblick

Die Prototypen, die in diesem Kapitel beschrieben werden, stellen dar, wie das POTATO-System und seine Bestandteile aufgebaut sind. Daher bauen die Prototypen aufeinander auf und besitzen zunehmende Komplexität.

Die ersten Prototypen stellen das HERA-System vor. Als erstes wird ein einfacher Helferagent gezeigt, der ohne weitere Ressourcen auskommt. Dabei wird zugleich auch die Infrastruktur gezeigt, die zum Auffinden und Erstellen eines Helferagenten und zum Einbinden in den Benutzeragenten benötigt wird. Als Beispiel wird eine einfache Chatanwendung implementiert.

Anschließend wird das System um einen Ressourcenagenten erweitert. Der zugehörige Helferagent besitzt nun weitere Protokolle für die Erzeugung und Manipulation von Ressourcen. Das Whiteboardsystem in diesem Prototypen zeigt das Zusammenspiel bei der Behandlung von Ressourcen durch Helferagenten.

Anschließend wird das PIA-System gezeigt. Als erster Prototyp dient hierbei ein einfacher Workflow, der komplett ohne die Hilfsmittel aus dem HERA-System abgewickelt wird. Das Workflowbeispiel für einen einfachen Change Management-Prozess stellt die Prozessinfrastruktur dar.

Im nächsten Schritt werden die Helferagenten gezeigt, mit denen auf das Workflow Management System zugegriffen werden kann. Der Helferagent für die Worklistenverwaltung zeigt eine Art der Integration von PIA und HERA.

Die andere Seite der Integration zeigt dann der Activity-Agent, der die Bearbeitung von Aktivitäten mit Hilfe von Helferagenten erlaubt. In diesem Beispiel wird eine zu implementierende Funktionalität zunächst textuell beschrieben, dieses Design dann durch einen anderen Benutzer geprüft und schließlich ein Petrinetz erstellt, das die gewünschte Funktionalität erbringt.

8.2 Prototyp Helferagent - Der Chat-Agent

Der Chat-Agent ist ein Beispiel für einen Helferagenten, der ohne zusätzliche Ressourcen eine nützliche Dienstleistung erbringt [WILLMOTT et al., 2005]. Er gibt einem Benutzer die Möglichkeit, mit anderen Anwendern der Entwicklungsumgebung per Textnachrichten zu kommunizieren. In der hier vorgestellten Implementierung wird der Directory Facilitator der Plattform verwendet, um die Verbindung zwischen den Agenten herzustellen, die Kommunikation erfolgt dann Peer-to-Peer.

8.2.1 Ontologie

Für den Chat-Agenten wird zunächst natürlich die Basis-Ontologie für Helferagenten benötigt, die in Abschnitt 6.3 beschrieben wurde. Da in diesem Beispiel

noch keine Ressourcenagenten verwendet werden, sind diese Konzepte natürlich noch nicht erforderlich. Dazu kommt eine spezialisierte Anwendungsontologie, die den Begriff der `ChatMessage` definiert, einer Nachricht, die zwischen Chat-Anwendern ausgetauscht wird.

8.2.2 Agenten

Für die grundlegende Helferumgebung HeRA müssen eine Reihe von Agenten implementiert werden, die später für die verteilte Softwareentwicklungsumgebung erweitert und ergänzt werden können. Der Benutzeragent als Schnittstelle zwischen Agentensystem und Benutzer implementiert die Funktionalität zum Anfordern und Verwenden von Helferagenten, sowie den Rahmen einer Benutzungsoberfläche, der durch die Helferagenten ausgefüllt wird.

Die Helferagentenfabrik ist zuständig für die Erzeugung von neuen Helferagenten. Dazu stellt sie einen Katalog von Helfern bereit, aus dem die gewünschten Agenten abgerufen werden können. Die Fabrik veranlasst dann die Erzeugung dieser Agenten und ihre Verbindung mit dem Benutzeragenten.

Helfer- und Ressourcenagenten stellen die Funktionen und Objekte im System dar und müssen daher je nach gewünschter Anwendung individuell erstellt werden. Zunächst muss daher nur ein Rahmen hierfür geschaffen werden, der später ausgefüllt wird. Die Funktionalität wird durch Helferagenten bereitgestellt. Diese bestehen aus einem Agententeil, der innerhalb der Capa Plattform ausgeführt wird und einem GUI-Anteil, der in den Benutzeragenten integriert wird und dort Elemente für den Zugriff auf die Helferfunktionalität anbietet.

Schließlich wurde für das Chatbeispiel ein Chat-Helferagent entwickelt. Dieser besitzt eine einfache GUI, die verschiedene Konversationen anzeigt und eine einfache Interaktion, mit der Nachrichten zwischen Agenten ausgetauscht werden können.

8.2.2.1 Der Benutzeragent

Der Benutzeragent stellt die Schnittstelle zwischen dem Agentensystem und dem Benutzer dar. Dementsprechend ist seine Implementierung zweigeteilt. Auf der Agentenseite gibt es einen Mulan-Agenten, der den Benutzer vertritt. Dieser Agent ist verantwortlich für das Senden und Empfangen von Nachrichten und die Ausführung von Agentenprotokollen. Auf der anderen Seite gibt es eine grafische Benutzungsoberfläche (GUI), die es einem Benutzer erlaubt, mit dem System zu interagieren. Die Ereignisse, die der Agent entgegennimmt werden darin dem Benutzer angezeigt und er erhält die Möglichkeit, seinerseits Aktionen im Agentensystem auszulösen.

Der Agent Der Benutzeragent verfügt selbst nicht über viel Funktionalität. Er sucht eine Helferfabrik im System und fragt die Liste der verfügbaren Helfer ab, um sie über die GUI darzustellen. Auf Anfrage aus der GUI startet er die

Interaktion zum Erstellten eines neuen Helferagenten. In seiner Wissensbasis verwaltet er eine Liste der verbundenen Helfer- und Ressourcenagenten.

Die Benutzungsoberfläche Die Benutzungsoberfläche des Agenten wurde mit Hilfe der Rich Client Platform (RCP) [RCP, 2011] implementiert. Dabei handelt es sich um ein von Eclipse entwickeltes Applikationsframework. Eine Basisapplikation kann durch verschiedene Plugins flexibel erweitert und an die Bedürfnisse des Benutzers angepasst werden. Die Eclipse-IDE ist selbst aufgebaut auf der Rich Client Platform. Auf diese Weise lässt sich POTATO unmittelbar als neues Plugin in eine vorhandene Eclipse-Entwicklungsumgebung integrieren. Die vorhandenen Möglichkeiten der Softwareentwicklung mit Eclipse können weiter verwendet und durch Helferagenten flexibel angesteuert und erweitert werden.

Die Oberfläche des Benutzeragenten ist implementiert als Produkt in RCP. Ein Produkt ist eine bestimmte Konfiguration, die aus bestimmten Plugins besteht, die vorhanden sein müssen, sowie einer Anwendungsklasse, die den Lebenszyklus des Produkts verwaltet. In diesem Produkt wird das Benutzeragentenplugin geladen und über den `RCPGuiConnector` eine Verbindung zwischen Agent und GUI aufgebaut.

Das Plugin selbst benötigt für diesen Prototypen lediglich ein Anwendungsfenster und einen Extension Point für Helferagenten. Dadurch, dass das Plugin diese Andockstelle für weitere Plugins definiert, können die GUIs der angeschlossenen Helferagenten in die Benutzungsoberfläche eingebaut werden.

Integration Die Zusammenarbeit des Agententeils und der Benutzungsoberfläche erfolgt über eine spezielle Entscheidungskomponente im Agenten, die sogenannte RemoteDC. Diese Entscheidungskomponente dient dazu, Entscheidungen des Agenten durch einen externen Benutzer vornehmen zu lassen, der sich über RMI[1] mit dem Agenten verbindet.

RemoteDC Abb. 8.1 zeigt einen Überblick über die Komponenten, die für die Verbindung der RemoteDC erforderlich sind. Um die Integration zu ermöglichen, wird eine Entscheidungskomponente im Agenten gestartet, die als Netstub implementiert ist [KUMMER et al., 2009, S. 60f]. Das ist ein Referenznetz, welches über eine speziell generierte Java-Klasse angesprochen werden kann. Wenn der Benutzeragent gestartet wird, wird eine Instanz der Klasse `DCJavaNetInteraction` erzeugt und in der Stelle für Entscheidungskomponenten abgelegt.

Beim Start erzeugt das Netz einen neuen `DCModelManager` und registriert sich bei diesem als zugehöriges Netz. Der `DCModelManager` bietet verschiedene Methoden an, um über Java auf den Agenten zuzugreifen, indem er Methoden

[1] Remote Method Invocation, vgl. [RMI, 2011]

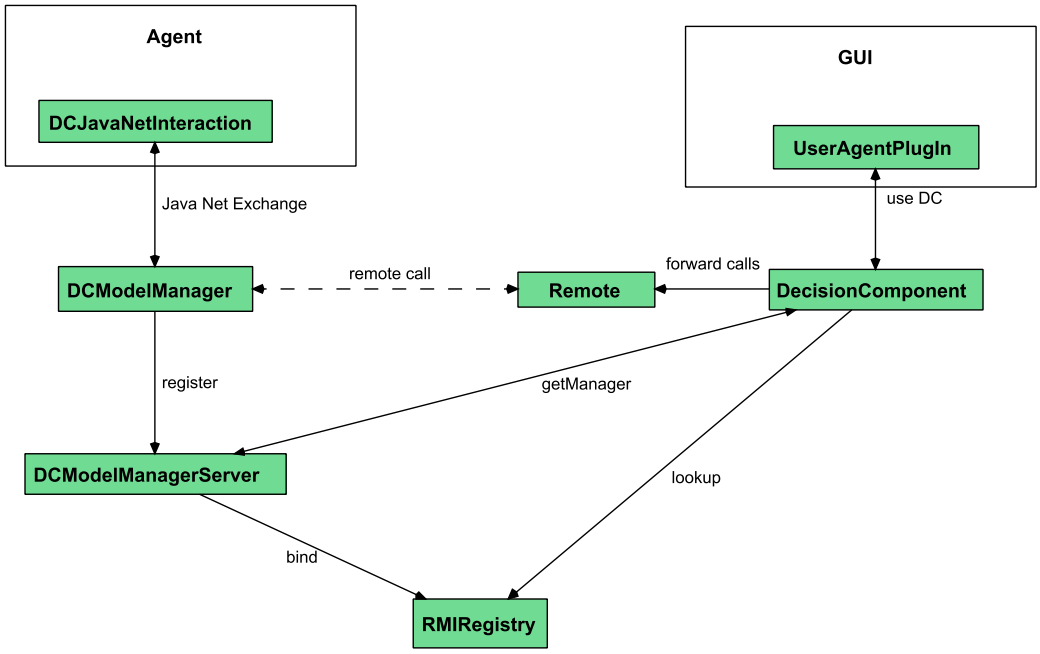

Abbildung 8.1: Registrierung und Aufruf der RemoteDC

wie `readWB, exchange, newPlan` bereitstellt und in Netzkanalaufrufe übersetzt.

Damit der `DCModelManager` von außerhalb der Agentenplattform gefunden werden kann, registriert er sich beim lokalen `ModelManagerServer`. Dieses Brokerobjekt ermöglicht den Zugriff auf die registrierten `ModelManager`, wenn sie mit dem korrekten Agentennamen und Passwort abgefragt werden. Dadurch können verschiedene Agenten auf einer Plattform über RemoteDC von außerhalb zugreifbar gemacht werden. Der `ModelManagerServer` selbst ist über eine RMI-Adresse zu erreichen. Die erforderlichen Agentennamen und Passwörter für Helferagenten sind Teil des `HelperInterface`, das ein Helferagent bei der Registrierung an den Benutzeragenten übermittelt.

Auf Clientseite wird ein Objekt der Klasse `DecisionComponent` erzeugt und mit Server, Agentenname und Passwort initialisiert. Es ermittelt dann über RMI den `ModelManagerServer` und bekommt über Agentenname und Passwort den RMI-Proxy für den gesuchten `DCModelManager`. Die `DecisionComponent` leitet dann alle Aufrufe an den Proxy weiter, so dass dadurch der entfernte Agent transparent gesteuert werden kann.

Abhängigkeiten Auf Benutzerseite muss die Benutzungsoberfläche gestartet und mit der RemoteDC verbunden werden. Hierfür sind verschiedene Strategien denkbar: Der Agent kann die GUI starten, die GUI kann den Agenten bzw. die lokale Plattform starten oder beide Teile werden separat gestartet und finden sich dann über einen Brokerdienst, in diesem Fall die RMIRegistry.

An dieser Stelle wird es offen gelassen, welche dieser Möglichkeiten die „richtige“ Lösung ist, das hängt letztendlich davon ab, wie der Fokus der Anwendung gelegt wird, welcher Teil „führend“ ist.

Ist die GUI nur eine Benutzungsschnittstelle für das Agentensystem, dann ist es sicherlich sinnvoll, sie vom Agentensystem starten zu lassen. Wenn aber auch die anderen Funktionen der Eclipse-Umgebung regelmäßig verwendet werden sollen, scheint es unintuitiv, dies nur über das Agentensystem zu starten. Da es an dieser Stelle noch nicht klar ist, wie die konkreten Benutzungsszenarien aussehen werden, wird eine Lösung gewählt, bei der das Agentensystem die GUI startet, die beiden Komponenten aber hinreichend unabhängig sind, so dass eine spätere Entkopplung ohne Probleme möglich ist.

GUI-Start Wenn der Agent die Entscheidungskomponente erstellt und die RemoteDC registriert hat, wird ein `GuiConnector` gestartet, um die GUI zu öffnen.[2] Dieser lädt über einen neuen `ClassLoader` die erforderlichen Klassen und startet die Rich Client Anwendung. Zusätzlich werden die RemoteDC-Verbindungsdaten als Properties gesetzt, so dass die Anwendung darauf zugreifen kann, um die Verbindung herzustellen. Wenn das Benutzeragenten-Plugin in der Rich Client Anwendung startet, liest es diese Daten aus und verbindet sich über RMI mit dem Agenten. Da Agentenplattform und Benutzungsoberfläche in derselben Java Virtual Machine laufen, wäre der Umweg über RMI nicht nötig, diese Vorgehensweise erleichtert aber eine spätere Trennung der beiden Komponenten. Auch ist es einfach möglich, die Agentenplattform auf einem anderen Rechner laufen zu lassen als die GUI und sich mit dieser über das Netzwerk zu verbinden.

Agenten-GUI-Interaktion Über die RemoteDC kann die GUI nun den Benutzeragenten steuern und Informationen erhalten. Dazu bietet die RemoteDC drei Funktionsgruppen an:

- Zugriff auf die Wissensbasis zum Lesen und Schreiben (bei Bedarf auch Bearbeiten der Protokolle und Entscheidungskomponenten),
- Starten neuer Protokolle über den `newPlan()`-Kanal,
- Registrieren von Callbacks, über die der Agent Ereignisse an die DC pushen kann.

Für den Zugriff auf die Wissensbasis bietet die RemoteDC einen `KnowledgeBaseProxy` an, der die gleichen Methoden besitzt, die auch das Wissensbasis-Netz als Kanäle zur Verfügung stellt. Auch das Starten neuer Protokolle orientiert sich an den Möglichkeiten, die eine normale Entscheidungskomponente besitzt. Ein Protokoll kann über den Namen ausgewählt und gestartet werden.

[2] In diesem Fall ein `RCPGuiConnector`

Dabei kann optional ein Ontologieobjekt als Parameter übergeben werden, das als Startnachricht im Protokoll verwendet wird.

Damit auch Ereignisse vom Agenten an die GUI übermittelt werden können, muss ein `CallHandler` registriert werden. Dieses Objekt registriert sich bei der Entscheidungskomponente und gibt an, auf welchen Kanälen es Anfragen entgegennimmt. Diese Kanäle können dann von Protokollen genutzt werden, um Entscheidungen treffen zu lassen, aber auch, um relevante Informationen weiterzugeben, die dann in der GUI dargestellt werden können.

Ein Beispiel hierfür ist die Interaktion zum Abfragen verfügbarer Helferagenten bei der Helferfabrik. Die Benutzeragenten-GUI stellt einen Button oder Menupunkt zur Verfügung „Helferagenten auflisten“. In dessen Eventhandler wird an der Entscheidungskomponente (also dem Proxy in der GUI) die Methode `newPlan` aufgerufen, um die Interaktion zu starten. Über RMI wird der Stub im Agentensystem angestoßen, der wiederum einen Schaltvorgang im DC-Netz auslöst. Damit wird die Interaktion gestartet, an deren Ende die Liste der verfügbaren Helfer bereitgestellt wird.

Um diese wiederum in der GUI darzustellen, wird im Protokoll der `:exchange`-Kanal aufgerufen, der über den Agenten mit dem DC-Netz synchronisiert wird. Hierbei werden dann die Liste der Helfer und eine Kanalidentifizierung in eine Stelle gelegt. Die verfügbaren Callhandler wurden vorher bereits mit ihrem Kanal registriert, daher kann der zuständige Handler ausgewählt und aufgerufen werden. Das führt zu einem Methodenaufruf in der GUI, wo dann in einem View die entsprechenden Daten in einer Liste angezeigt werden.

8.2.2.2 Die Helferfabrik

Aufgabe der Helferfabrik oder Helperfactory ist es, neue Helferagenten zu instanziieren und dafür zu sorgen, dass diese mit dem anfordernden Benutzeragenten verbunden werden. Dafür verfügt sie über ein Repository von bekannten Helferagenten, die von Benutzern angefordert werden können.

Eine spezielle Entscheidungskomponente erhält die Anfrage des Benutzeragenten und erzeugt eine Anfrage an das Agent Management System (AMS) der Plattform, um den neuen Helferagenten zu erzeugen. Hierbei können Parameter übergeben werden, die zusätzlich in die Wissensbasis des neu zu erzeugenden Agenten eingefügt werden sollen. Ein Beispiel hierfür ist die Erzeugung von Aktivitätsagenten im Workflow, die als Parameter unter anderem die zu bearbeitende Aktivität übergeben bekommen. Auf diese Weise können Helferagenten bei der Erzeugung individualisiert werden.

8.2.2.3 Der Helferagent

Helferagenten implementieren die Funktionalität im HeRA-System. Daher unterscheiden sie sich stark untereinander, je nach Anwendungsgebiet. Gemeinsam haben sie die Basisprotokolle für die Registrierung beim Benutzeragenten

und für die Erstellung und Verwendung von Ressourcenagenten.

Basis-Agent Die verschiedenen Helferagenten in einem HeRA-System bauen alle auf einem grundlegenden Basis-Helferagenten auf. Dieser hat die Protokolle zum Registrieren mit einem Benutzeragenten, für die GUI-Interaktion etc. Spezialisierte Helferagenten, wie der unten beschriebene Chat-Agent erweitern dann dieses Gerüst um eigene Funktionalität. Neben speziellen, anwendungsspezifischen Informationen hat jeder Helferagent in seiner Wissensbasis eine Referenz zu dem Benutzeragenten, dem er zugeordnet ist.

GUI-Integration Die Helferagenten besitzen einen GUI-Anteil, der als Plugin in eine RCP-Umgebung eingefügt werden kann. Zu diesem Zweck definiert der Benutzeragent einen Extension Point, also eine Schnittstelle zur Erweiterung durch Plugins. Helferagenten implementieren dann diesen Extension Point, indem sie die Klasse `HelperAgent` erweitern. `HelperAgent` erledigt die Registrierung der Entscheidungskomponente des Helferagenten und ermöglicht abgeleiteten Klassen so Zugriff auf den Netzagenten.´

Bei der Registrierung eines Helferagenten beim Benutzeragenten erhält dieser eine `RegisterHelper`-Nachricht, die die Zugriffsdaten für die Entscheidungskomponente, sowie URL und Pluginklasse für die GUI enthält. Von der URL kann dann über OSGI [OSGi, 2011] das erforderliche Plugin geladen und anschließend eine Instanz des Helfers erstellt werden. Wird ein Helfer mehrfach instanziiert, muss das Plugin natürlich nicht erneut geladen werden, sondern lediglich eine neue Instanz der Helferagentenklasse erzeugt werden. Diese Agentenklasse kann dann notwendige Initialisierungen in der GUI vornehmen, wie etwa das Öffnen von Views und Editoren.

8.2.2.4 Chat-Agent

Der Chat-Agent verwendet die vorhandenen Klassen des Helferagenten und stellt eine eigene GUI-Erweiterung zur Verfügung. Beim Start öffnet diese ein Editorfenster, in dem eine Liste der möglichen Chatpartner angezeigt wird. Da der Chat-Agent vom Directory Facilitator nur die Adressen der anderen Chat-Agenten erhält, muss er die `GetUserAgentMapping`-Interaktion verwenden, um die zugehörigen Benutzeragenten zu ermitteln, mit denen der Chat geführt wird.

Aus der Liste können Chatpartner ausgewählt werden, für die sich dann ein neuer Reiter mit der jeweiligen Konversation öffnet (siehe Abb. 8.2). Erhält der Chat-Agent eine Nachricht von einem bisher unbekannten Chatpartner, öffnet er für diesen ebenfalls einen neuen Reiter.

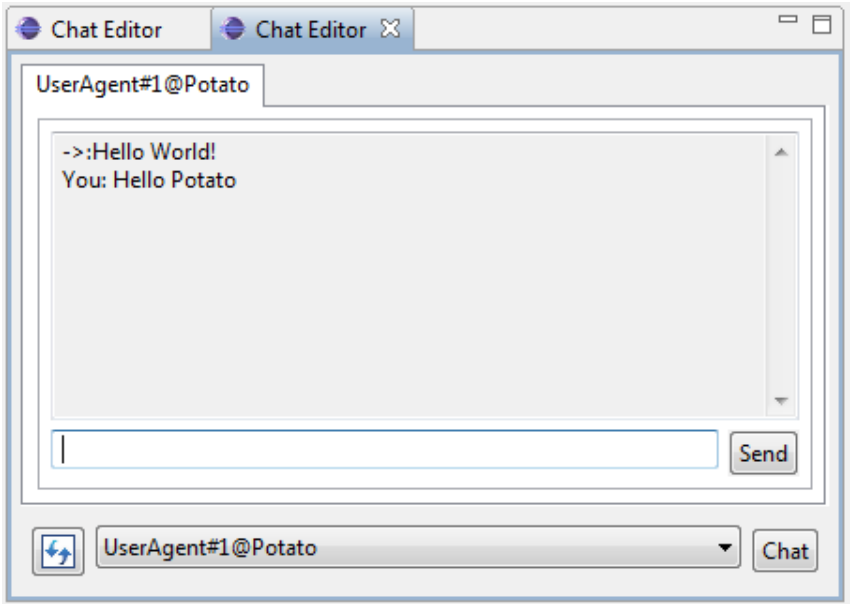

Abbildung 8.2: Benutzungsoberfläche des Chat-Agenten

8.2.3 Interaktionen

In diesem Abschnitt wird die Implementierung der verschiedenen Interaktionen beschrieben, die für die Helferagentenverwaltung erforderlich sind, sowie die Interaktion für die Chat-Anwendung.

8.2.3.1 Helferliste anfordern

Bevor der Benutzeragent einen neuen Helferagenten anfordern kann, muss er zunächst wissen, welche Arten von Helfern eine Helferfabrik anbietet. Die im Multiagentensystem verfügbaren Helferfabriken kann er über den Directory Facilitator (DF) Agenten finden, den jede Agentenplattform besitzt. Anschließend kontaktiert er eine oder mehrere Helferfabriken mit einer Anfrage nach verfügbaren Helfern. Dabei können einschränkende Parameter übergeben werden, wenn ein bestimmter Typ von Helferagent gewünscht wird. Im aktuellen Prototypen ist diese Funktionalität allerdings nicht implementiert, hier werden immer alle verfügbaren Helfer aufgelistet.

8.2.3.2 Helferagenten anfordern

Im Zuge dieser Interaktion wird vom Benutzeragenten bei der Helferfabrik ein neuer Helferagent angefordert und mit dem Benutzeragenten verbunden. Die Helferfabrik kennt die Details zur Erzeugung des Agenten und sorgt dafür, dass er erzeugt wird. Dafür kontaktiert sie den AMS-Agenten der Plattform. Anschließend sorgt sie dafür, dass der neue Agent mit dem Benutzeragenten

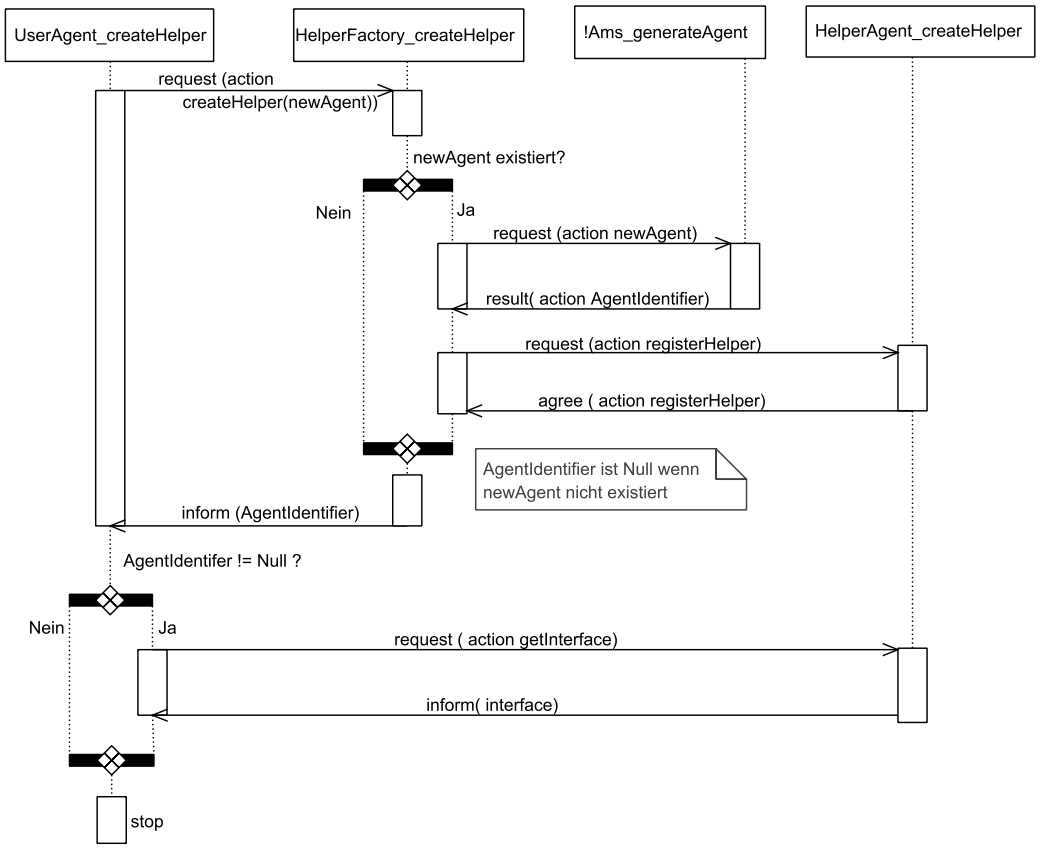

Abbildung 8.3: Interaktion zum Erzeugen eines neuen Helferagenten

bekannt gemacht wird, indem er beim Helferagenten die Interaktion zum Registrieren beim Benutzeragenten startet.

Die Helferfabrik muss hierfür die Wissensbasis des neuen Agenten konstruieren. Diese wird zunächst aus einer Datei in eine Hashmap eingelesen. Anschließend werden eventuelle übergebene Parameter für den Helferagenten ebenfalls in die Hashmap eingefügt. Diese Hashmap wird dann als codierter String in die Nachricht an den AMS-Agenten gesendet.

Die Interaktion zur Erzeugung und Registrierung eines neuen Helferagenten zeigt Abb. 8.3. Im Gegensatz zu der Abbildung im Entwurf sind hier auch die verschiedenen Fehlerfälle aufgeführt, da dieses Agenteninteraktionsprotokoll direkt zur Generierung einer Ausgangsversion der Agentenprotokolle verwendet werden kann.

8.2.3.3 Helferagenten beim Benutzeragenten registrieren

Nach der Erzeugung muss sich der neue Helferagenten beim Benutzeragenten registrieren, bevor er verwendet werden kann. Diese Interaktion wird durch die Helferfabrik gestartet, die dem neuen Helferagenten mitteilt, bei welchem

Benutzeragenten er sich registrieren soll.

Der Helferagent sendet dann dem Benutzeragenten die Anfrage zur Registrierung, zusammen mit dem eigenen `HelperInterface`. Die Daten im `HelperInterface` erlauben es dem Benutzeragenten, die GUI des Helferagenten zu laden und eine Verbindung mit dem Agenten herzustellen. Auch hierfür wird die RemoteDC verwendet, die über die im `HelperInterface` übergebenen Zugriffsparameter angesteuert wird. Ebenso merkt sich der Helferagent in seiner Wissensbasis, bei welchem Benutzeragenten er registriert ist. Anschließend wird eine Bestätigung an den Helferagenten und die Helferfabrik gesendet.

8.2.3.4 Benutzeragenten abfragen

Es kann vorkommen, dass ein Agent nur einen Helferagenten kennt, er aber wissen möchte, zu welchem Benutzeragenten dieser gehört. Beim Chat-Agenten etwa ist ein Anwender daran interessiert, mit welchem Benutzer er chattet, nicht wie der Helferagent heißt. Daher bieten Helferagenten eine Interaktion `GetUserAgentMapping` an, um ihr `UserAgentMapping` abzufragen.

8.2.3.5 Listener registrieren

Mit dieser Interaktion kann sich ein Helferagent bei einer Ressource als Beobachter registrieren. Er wird dann über Änderungen am Zustand der Ressource informiert. Dies ist insbesondere für Ressourcen interessant, auf die von verschiedenen Helfern gleichzeitig zugegriffen werden kann (lesend oder schreibend).

8.2.3.6 Chat-Interaktion

Die Chat-Anwendung greift zunächst auf die vorhandenen allgemeinen Helferagenten-Interaktionen zu. Ein Benutzeragent kann über die Interaktion zum Anfordern und Registrieren eines Helferagenten einen neuen Chat-Agenten erzeugen und registrieren.

Der Chat-Agent registriert sich dabei beim Verzeichnisdienst der Agentenplattform und gibt an, dass er einen Chat-Dienst anbietet. Anschließend fragt er seinerseits dort an, welche anderen Agenten diesen Dienst anbieten und damit mögliche Chatpartner darstellen. Dies liefert zunächst eine Liste der Chat-Helferagenten. Von diesen wird dann über die Interaktion `GetUserAgentMapping` erfragt, zu welchem Benutzeragenten sie gehören, um dessen Namen in der Liste verfügbarer Chatpartner anzuzeigen.

Um zu chatten, wird lediglich eine Nachricht mit einer `ChatMessage` erzeugt und an den entsprechenden Chat-Agenten gesendet. Dieser kann dann ein neues Fenster für die Kommunikation öffnen und die Nachricht anzeigen. Abb. 8.2 zeigt die Benutzungsoberfläche des Chat-Agenten.

Die Implementierung wurde bewusst so simpel wie möglich gehalten, da hier nur das Konzept eines Helferagenten demonstriert wird. Für die tatsächliche Anwendung in größeren Umgebungen könnten eigene Chatserver implementiert werden, über die geeignete Chatpartner gesucht werden können, Chaträume, Listen von bekannten Chatpartnern etc. Das liegt aber außerhalb des Fokus dieses Prototypen.

8.3 Prototyp Ressourcenagent - Der Whiteboardagent

Im nächsten Prototypen wird der zweite Baustein von HeRA implementiert, der Ressourcenagent. Zur Verdeutlichung wird eine einfache Whiteboardanwendung implementiert, bei der verschiedene Benutzer über ihre Helferagenten gemeinsam eine verteilt zugreifbare Ressource verwenden können. Die Änderungen, die ein Benutzer an der Ressource vornimmt, werden unmittelbar für die anderen Benutzer sichtbar. Dieses Beispiel wurde bereits in Kapitel 6 diskutiert.

8.3.1 Überblick

Das Whiteboard-System ist ein Beispiel für die Interaktion zwischen Helfer- und Ressourcenagenten. Aufgabe des Systems ist es, mehreren Benutzern zu ermöglichen, auf einer gemeinsamen, verteilt zugreifbaren Ressource (dem Whiteboard), Bearbeitungen vorzunehmen, die von allen Benutzern beobachtet werden können. Auf diese Weise können Ideen und Konzepte gemeinschaftlich entwickelt werden.

Ausgehend vom Chat-Prototypen sind für die Whiteboardanwendung eine Reihe weiterer Funktionen erforderlich. Als erstes muss der neue Agententyp des Ressourcenagenten definiert werden. Für die Erzeugung dieses Agenten sind die Helferagenten zuständig, die daher um Protokolle zur Erzeugung von Ressourcenagenten erweitert werden müssen. Für den Zugriff auf die Ressourcenagenten können im Einzelfall jeweils spezialisierte Protokolle verwendet werden, die sich an Benutzungsmustern der Ressource orientieren, hier wird mit dem Beispiel der Whiteboardressource eine einfache Variante gezeigt, die nur geringe Autonomie des Ressourcenagenten erfordert.

Für dieses System wurden zwei verschiedene Arten von Agenten implementiert:

- Die Whiteboard-Ressource, die den aktuell bearbeiteten Inhalt darstellt und Änderungen daran überwacht.

- Der Whiteboard-Helfer, der dem Benutzer den Zugriff auf ein Whiteboard ermöglicht, indem es den aktuellen Inhalt darstellt und Bearbei-

tungsfunktionen bereitstellt. Dazu gehört auch ein GUI-Plugin für die Verwendung im Benutzeragenten.

8.3.2 Ontologie

Zentrale Objekte in der Whiteboard-Ontologie sind der Whiteboardinhalt (zunächst nur als Text, Grafiken und andere Objekte wären aber natürlich auch denkbar), sowie Whiteboardänderungen. Änderungen können als neue Versionen oder als Differenzbeschreibungen formuliert werden. Sie werden verwendet für die Anforderung einer Änderung durch den Helfer wie auch für Bekanntmachung einer erfolgten Änderung an die registrierten Listener.

In der gewählten Implementierung wird aus Vereinfachungsgründen eine Änderung nur als neue Version mit einer Referenz auf die der Bearbeitung zugrundeliegende Basisversion angegeben. Eine Zusammenführung von nebenläufigen Änderungen oder die Bildung von Differenzen zur Reduktion des Datenverkehrs finden nicht statt.

8.3.3 Ressourcenagent

Ressourcenagenten stellen Arbeitsmaterialien und andere Ressourcen dar, die durch Helfer erzeugt und bearbeitet werden. Innerer Aufbau und Zugriffsmuster sind daher sehr unterschiedlich, je nachdem, um welche Art von Ressource es sich jeweils handelt. Die Muster zum Zugriff auf eine Ressource muss daher nur den Helferagenten bekannt sein, die diese Ressource bearbeiten. Gemeinsam haben verschiedene Ressourcenagenten lediglich die grundlegenden Protokolle zum Erzeugen und Verschieben zwischen Helferagenten, andere Protokolle können bei Bedarf verwendet werden.

Im Whiteboard-Prototypen wird ein Whiteboard-Ressourcenagent verwendet. Dieser kapselt lediglich einen internen Zustand und erlaubt verschiedenen Helferagenten, diesen zu verändern. Die Helferagenten können sich als Listener registrieren und erhalten dann automatisch eine Nachricht, wenn sich der Zustand der Ressource ändert. Die hierfür verwendete `notifyResourceChange`-Interaktion kann für verschiedenste Ressourcen verwendet werden, deren Zustand durch einen Helferagenten beobachtet wird.

Es sind verschiedene Möglichkeiten denkbar, wo sich ein Ressourcenagent befinden kann. Für die Bearbeitung kann sich die Ressource (logisch) innerhalb des Benutzeragenten befinden, der gerade Zugriff darauf hat, oder sogar innerhalb des bearbeitenden Helferagenten. Im Falle des Whiteboards handelt es sich aber um eine verteilt genutzte Ressource, die sich an einer beliebigen Stelle im Multiagentensystem befinden kann und auf die von allen Helferagenten frei zugegriffen werden kann. Andere Ressourcenagenten können hier restriktivere Zugriffsbeschränkungen (z.B. nur innerhalb der gleichen Plattform oder nur durch einen festgelegten Agenten zur Zeit) durchsetzen.

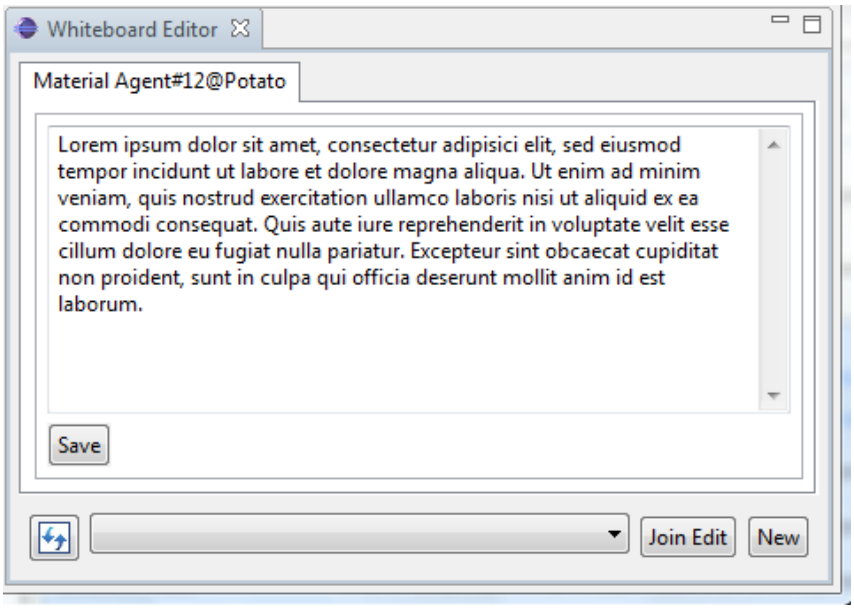

Abbildung 8.4: Benutzungsoberfläche des Whiteboard-Agenten

8.3.4 Whiteboard-Helfer

Der Whiteboard-Helferagent ist eine einfache Adaption des Chat-Agenten. Statt im Peer-to-Peer-Verfahren Nachrichten auszutauschen, wird der Zustand einer Ressource bearbeitet. Die dafür verwendete Oberfläche ist aber der des Chat-Agenten sehr ähnlich (siehe Abb. 8.4). Die Oberfläche des Whiteboard-Helfers, wird auf die gleiche Weise in die GUI des Benutzeragenten eingebunden wird, wie das mit dem Chat-Agenten gemacht wurde.

Ein Benutzer, der einen neuen Whiteboard-Helfer anfordert, bekommt als Oberfläche einen Editor. Dieser besitzt ein Hauptfenster für die Bearbeitung verschiedener Whiteboards, sowie eine Auswahlliste vorhandener Whiteboard-Ressourcen in der Umgebung. Der Benutzer kann nun eine Verbindung zu einer existierenden Whiteboard-Ressource aufbauen oder eine neue erstellen. Die verbundenen Whiteboards werden dann in einer Tab-Ansicht dargestellt, so dass mehrere Whiteboards gleichzeitig bearbeitet werden können.

In der prototypischen Implementierung besteht die Eingabefläche des Whiteboards lediglich aus einem Textfeld, in dem Änderungen vorgenommen werden können. Diese Änderungen können dann gespeichert werden, woraufhin bei allen verbundenen Helferagenten ein Update mit dem neuen Inhalt vorgenommen wird.

So wie der Chat-Agent neue Chatpartner über den Verzeichnisdienst (Directory Facilitator) der Plattform sucht, sucht der Whiteboard-Helfer nach verfügbaren Whiteboard-Ressourcen. Diese registrieren sich bei der Erzeugung als `WhiteboardResource` beim Verzeichnisdienst und können dann von allen geeigneten Helfern beobachtet und bearbeitet werden.

Wird eine Ressource zur Bearbeitung ausgewählt oder eine neue erstellt, registriert sich der Agent als Beobachter bei dem Ressourcenagenten, um stets über den aktuellen Inhalt auf dem Laufenden gehalten zu werden. Bei einer Änderung wird eine Nachricht mit dem neuen Inhalt gesendet, die dazu führt, dass die Anzeige im Helferagenten aktualisiert wird. Nimmt der Benutzer selbst eine Änderung vor und speichert diese, wird eine Nachricht mit den Details der Änderung an den Ressourcenagenten gesendet, so dass auch andere Beobachter auf den neuen Stand gebracht werden.

8.3.5 Interaktionen

Der Benutzeragent verwendet die normalen Protokolle, um einen Whiteboard-Helferagenten zu erstellen. Dieser wird beim Benutzeragenten registriert und startet sein GUI-Plugin.

Anschließend bietet er die Möglichkeit, eine neue Ressource zu erzeugen. Hierfür sendet der Helferagent eine Nachricht mit den erforderlichen Daten an den AMS-Agenten der Plattform und bekommt von diesem die Adresse des neuen Ressourcenagenten genannt. Er meldet sich bei diesem als Beobachter an, um über Änderungen auf dem Laufenden gehalten zu werden. In der GUI wird ein neuer Tab geöffnet, um den Inhalt des Whiteboards anzuzeigen und zu bearbeiten.

Damit das Whiteboard interaktiv von verschiedenen Benutzern verwendet werden kann, ist es möglich, auf eine Whiteboardressource zuzugreifen, die von einem anderen Helferagenten erzeugt worden ist. Dafür muss ein Mitbearbeiter zunächst den Whiteboard-Ressourcenagenten finden. Analog zur Implementierung beim Chat-Agenten werden auch hier potentielle zu bearbeitende Ressourcen über die angebotenen Dienste gesucht. Bei diesen kann sich der Helferagent als Listener registrieren und Änderungsanfragen an diese senden.

Hierfür wären natürlich auch andere Lösungen denkbar, die mehr Flexibilität und größere Kontrolle über die Whiteboards erlauben, etwa über einen zentralen Whiteboardserver, sowie verschiedene Berechtigungen. Diese Aspekte werden aber zunächst zurückgestellt, um den Anwendungsfall einer verteilten Ressource darzustellen.

Die Bearbeitung des Whiteboardinhalts erfolgt über das `changeWhiteboardContent`-Protokoll. Wenn der Benutzer eine Änderung am Whiteboardinhalt festschreiben will, wird eine `request(WhiteboardChange)`-Nachricht an den Ressourcenagenten gesendet. Dieser versucht dann, diese Änderungen in die interne Repräsentation einzufügen und benachrichtigt den Agenten über Erfolg oder Misserfolg. Eine Bearbeitung kann beispielsweise fehlschlagen, wenn die im `WhiteboardChange` angegebene Basisversion nicht die aktuelle Version des Inhalts ist, also versucht wird, eine veraltete Version zu bearbeiten.

Anschließend geht der Agent die Liste der registrierten Listener durch und startet für jeden das `notifyResourceChange`-Protokoll mit der durchgeführ-

ten Änderung. Die Helferagenten passen daraufhin ihre jeweilige Darstellung des Whiteboardsinhalts an. In der aktuellen Version werden an dieser Stelle noch nicht gespeicherte Änderungen überschrieben, stattdessen könnten aber natürlich auch Merge-Algorithmen verwendet werden, wie sie beispielsweise bei Sourcecode Verwaltungssystemen zum Einsatz kommen.

8.3.5.1 Ressourcenerzeugung

Die Erzeugung neuer Ressourcen ist Aufgabe der Helferagenten, diese müssen wissen, wie die Ressource erzeugt werden kann. Konkret bedeutet das, dass der Helferagent eine Nachricht mit den nötigen Informationen für den neuen Agenten an die Agentenplattform sendet, insbesondere dem Inhalt seiner Wissensbasis. Im Falle des Whiteboard-Helfers werden alle Ressourcen identisch erzeugt, mit einem initial leeren Inhalt. Der Helfer besitzt eine komplette Kopie dieser initialen Wissensbasis in seiner eigenen Wissensbasis und verwendet diese dann, um den neuen Agenten zu konstruieren. Analog zur Helferfabrik sind natürlich auch hier komplexere Erzeugungsmuster möglich.

8.3.5.2 Ressourcenbearbeitung

Die genaue Art, wie eine Ressource bearbeitet wird, ist abhängig von der jeweiligen Ressource. Im Beispiel wurde daher eine eigene Interaktion implementiert für die Bearbeitung von Whiteboard-Inhalten.

Damit ein Helferagent ein Whiteboard bearbeiten kann, muss er als Listener registriert sein. Dadurch besitzt er eine aktuelle Version des Whiteboardinhaltes. Um eine Änderung vorzunehmen, wird der neue Inhalt unter Angabe der Basisversion, auf die sich diese Änderung bezieht, an den Ressourcenagenten gesendet.

Dieser prüft, ob die geänderte Basisversion noch die aktuelle Version ist, oder ob in der Zwischenzeit von anderer Seite Änderungen vorgenommen worden sind. Wurde der Inhalt in der Zwischenzeit geändert, wird die Änderung abgelehnt. Die korrekte Version bekommt der Helferagent als Listener automatisch zugesendet. Ist die Änderung erfolgreich, wird eine Bestätigung gesendet und anschließend alle Listener über den neuen Zustand unterrichtet.

Andere Möglichkeiten der Konfliktbehandlung wären natürlich denkbar gewesen (etwa der Versuch einer automatischen Zusammenführung, wie das bei Versionskontrollsystemen gemacht wird), für den hier vorgestellten einfachen Prototypen ist dieses Verhalten aber ausreichend.

8.3.6 Diskussion

In den hier für das HeRA-System vorgestellten Prototypen sind nicht sämtliche Funktionen umgesetzt, die für ein vollwertiges Helferagenten-Framework

erforderlich sind. Ziel war es hier vielmehr, die Grundfunktionalität vorzustellen und das Zusammenspiel von Helfer- und Ressourcenagenten darzustellen.

Ein Aspekt, der für die praktische Nutzbarkeit des Systems unabdingbar ist, der hier aber komplett ausgeklammert wurde, ist eine Nutzerverwaltung. Für eine sinnvolle Arbeit mit dem System muss es möglich sein, Benutzer am System zu authentifizieren, um beispielsweise ihren aktuellen Arbeitsplatz direkt beim Systemstart wiederherstellen zu können. Hand in Hand damit geht die Funktionalität, die Konfiguration eines Benutzeragenten, also die Menge der verbundenen Helfer- und Ressourcenagenten zwischenzuspeichern, um sie dann später wieder abrufen zu können.

Zusätzlich können über verschiedene Benutzerrollen auch die Funktionen der Helferagenten angepasst werden. Verfügt ein Benutzer nicht über die Berechtigung, bestimmte Ressourcen zu bearbeiten, kann ihm ein Helferagent zur Verfügugn gestellt werden, der überhaupt nicht über die entsprechenden Protokolle zur Bearbeitung verfügt.

8.4 Prototyp Prozessinfrastruktur - Change Management

Nachdem das HeRA-System mit Prototypen vorgestellt worden ist, wird jetzt die Prozessinfrastruktur näher beleuchtet. Für die Prozessinfrastruktur wurde zunächst ein Prototyp implementiert, der die Funktionalität des HeRA-Systems überhaupt nicht verwendet. In der ursprünglichen Implementierung von Pia ist diese Integration nicht vorgesehen, daher funktioniert das System auch ohne Helfer- und Ressourcenagenten.

Die Prozessinfrastruktur stellt ein komplettes Workflow Management System dar, das mit Agenten implementiert wurde. Die verschiedenen Bestandteile eines klassischen WfMS wurden als einzelne Agenten realisiert. Die Funktionsweise und Implementation des Pia-Systems wurde bereits in Kapitel 7 beschrieben. Hier wird daher nur kurz der Beispielworkflow beschrieben.

8.4.1 Change Management-Beispielworkflow

Der in diesem Beispiel verwendete Workflow ist ein vereinfachter Ablauf für die Behandlung von Fehlern oder Änderungen in einem Softwaresystem (Vgl. [Markwardt et al., 2006]). Abb. 8.5 zeigt das verwendete Workflownetz, das für die Ausführung des Workflows verwendet wird.

Wenn ein Benutzer einen Fehler in der Anwendung findet oder eine Änderung wünscht, erfasst er einen neuen Change Request (CR). Dafür verwendet er ein Formular, das in der Datenbank definiert ist. Der CR wird dann durch einen dazu berechtigten Benutzer (Change-Koordinator) kategorisiert. Dieser entscheidet über die Priorität des CR und wählt aus, ob er beauftragt werden

soll. Entscheidet er sich gegen eine Beauftragung, endet der Workflow an dieser Stelle.

Andernfalls wird der CR durch einen Bearbeiter angenommen. Damit beginnt die Bearbeitung. Sie endet mit dem Abschließen des CRs, woraus dann beispielsweise eine Bearbeitungszeit berechnet werden kann, die später in eine Kostenberechnung eingeht. Das Ergebnis des CRs wird dann einem Tester vorgelegt, der den CR prüft und entscheidet, ob die gewünschten Änderungen umgesetzt wurden, bzw. ob der Fehler behoben wurde. War die Bearbeitung erfolgreich, wird der Workflow abgeschlossen, andernfalls wird der CR erneut zur Bearbeitung vorgelegt.

8.4.2 Workflow-Aufgaben

Der einzige Aufgabentyp, der im Pia-System implementiert ist, ist der FormTask, der es erlaubt, Formulare zu definieren, die dann durch Benutzer im Workflow ausgefüllt werden. Für den Change Management-Workflow wurde eine Reihe von Formularen definiert, mit denen ein CR erfasst und kategorisiert werden kann. Die Entscheidung über die Beauftragung des CRs bzw. den erfolgreichen Test wird durch eine Checkbox realisiert, die dann direkt im Workflownetz ausgewertet wird.

Die verschiedenen Aufgaben erfordern unterschiedliche Formulare, die teilweise Daten aus den vorangegangenen Formularen weiterverwenden. Beispielsweise wird im Formular „CR erfassen“ eine Bezeichnung und Beschreibung des CR eingegeben, die in allen folgenden Formularen wieder aufgegriffen werden. Über FormToFormRules wird festgelegt, wie diese Übertragung erfolgt.

Die Benutzer-GUI interpretiert die Formularbeschreibungen und setzt sie in Java-Formulare um, die dann durch den Benutzer ausgefüllt werden. Abb. 8.6 zeigt eine Formularbeschreibung und das zugehörige, durch das System aufbereitete Formular. Die Beschreibung verwendet eine in Pia definierte Ontologie für Formulare. Ein Formular besteht aus verschiedenen Formularelementen unterschiedlicher Typen (Label, Checkbox, Text etc.). Diese besitzen verschiedene Attribute, die die Darstellung beeinflussen. Eingabefelder können darüber hinaus einen Namen und einen Wert besitzen, über den ein Vorgabewert eingetragen und Werte für weitere Verwendung im Workflow gespeichert werden können.

Diese Beschreibungen werden dann in konkrete GUI-Elemente umgesetzt, in diesem Fall in eine Swing-GUI. Der Benutzeragent sorgt dafür, dass alle Felder richtig angezeigt und ausgefüllt werden und die eingetragenen Werte an das WfMS übermittelt werden.

8.4.3 Rollen und Regeln

Regeln im Workflow legen fest, welche Aufgaben welchen Benutzern zugeordnet werden können. Zurzeit ist dies lediglich eine Auflistung der Rollen, die

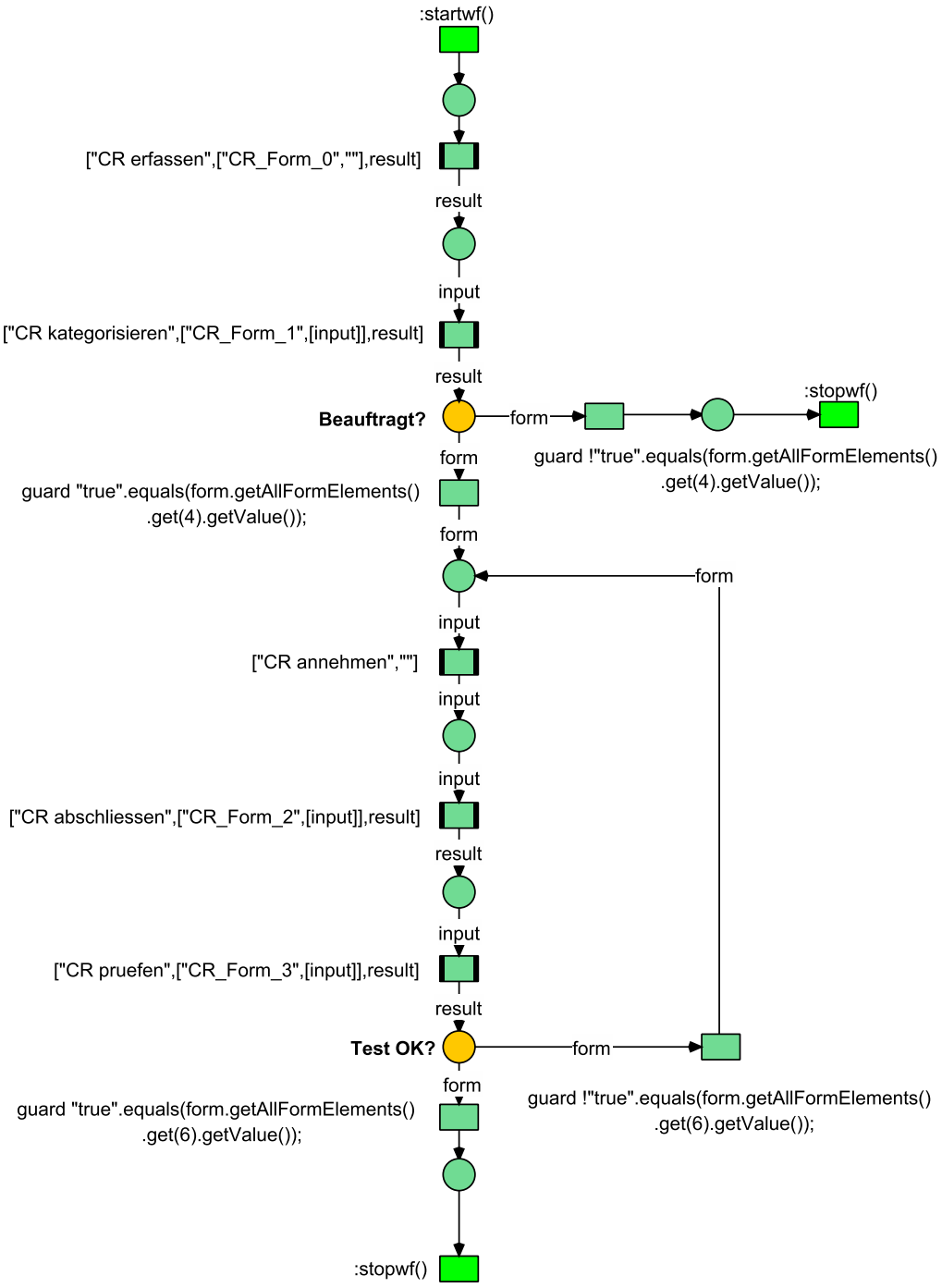

Abbildung 8.5: Change Management-Workflow

```
(form :name CR\_Form\_1 :form-elements (sequence
 (label-element :title "CR Bezeichnung" :name title :value "Lorem ipsum")
 (text-element :title Beschreibung :name body :value
   "dolor sit amet, consectetur adipisici elit, sed eiusmod temp")
 (combo-box-element :title Typ :name type :value Fehler :options (sequence
   Fehler Erweiterung))
 (combo-box-element :title Prio :name priority :value "" :options (sequence
   "1 - Hoch" "2 - Mittel" "3 - Niedrig"))
 (check-box-element :title "Umsetzung beauftragen" :name assign :value "")
 )
)
```

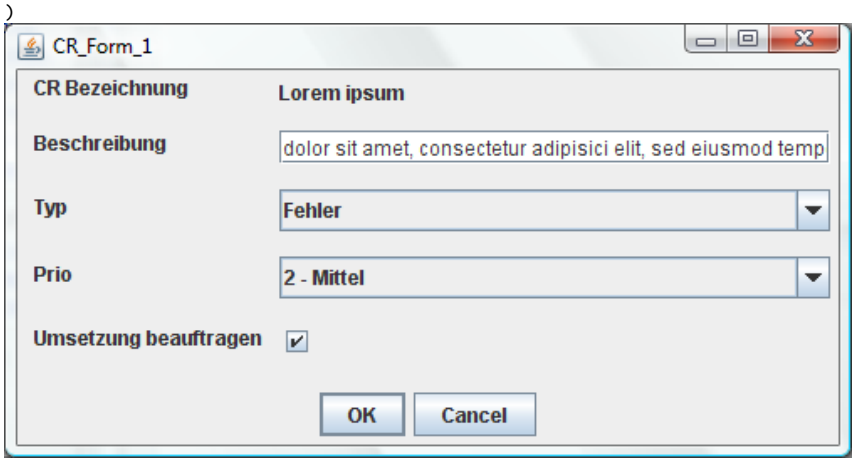

Abbildung 8.6: Formularbeschreibung und -darstellung in Pia

diese Regel erfüllen: Jeder Benutzer, der eine der aufgelisteten Rollen innehat, ist damit qualifiziert für die Aufgabe. Komplexere Zuweisungen wären denkbar, etwa eine Zuweisung in Abhängigkeit davon, welcher Benutzer eine vorherige Aufgabe ausgeführt hat oder aufgrund von Daten im Workflow. Für das Beispiel genügt aber eine einfache Rolleneinteilung in CR-Erfasser, Change Koordinator, Entwickler und Tester.

8.5 Prototyp Integration - Netzentwicklung

Als nächstes wird nun die Verwendung der HeRA-Agenten in der Prozessinfrastruktur gezeigt. Zur Veranschaulichung, wie die Verwendung von Benutzer-, Helfer- und Ressourcenagenten das Pia-System erweitern kann, wird ein Beispielworkflow vorgestellt, der den Workflow-Client Helfer und zwei verschiedene Activity-Agenten verwendet.

Aufgaben im Workflow werden charakterisiert durch eine Aufgabendefinition, die bestimmt, auf welche Art diese Aufgabe ausgeführt werden kann. Im klassischen Pia-System müssen alle Aufgabentypen im Voraus im WfMS und im Workflow-Client bekannt sein. Die Change Management-Anwendung verwendete beispielsweise ausschließlich die Formularaufgabe. Diese lässt sich zwar in gewissem Maße über flexibel definierbare Formulare konfigurieren, aber komplett neue Aufgabentypen erfordern individuelle Programmierung im Workflow-Client.

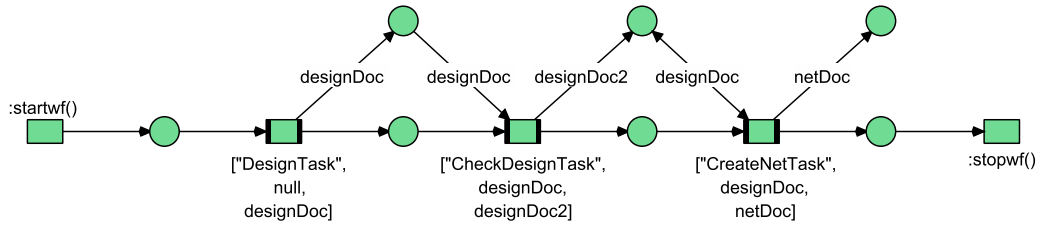

Abbildung 8.7: Netzentwicklungs-Workflow

Aus diesem Grund wird in POTATO das PIA-System um die Agententypen aus HERA erweitert. Aufgaben werden durch spezialisierte Helferagenten, Activity-Agenten genannt, bearbeitet. Dadurch kann ein Workflow beliebige Aufgabentypen hinzufügen, ohne dass dafür der Benutzeragent angepasst werden muss. Lediglich die Helferfabrik muss um die neuen Helferagententypen erweitert werden. Zusätzlich wird ein Helferagent verwendet, der dem Benutzer den Zugriff auf das WfMS erschließt, der Workflow-Client Helfer.

In diesem Beispiel wird ein einfacher Workflow für die Entwicklung eines neuen Petrinetzes verwendet, zum Beispiel als Teil eines größeren Softwareprojektes. Der Workflow besteht aus drei Schritten: Im ersten Schritt wird eine textuelle Spezifikation für ein Netz aufgestellt, im zweiten Schritt wird diese Spezifikation durch einen anderen Benutzer überarbeitet und schließlich wird sie in ein Petrinetz umgesetzt. Das verwendete Workflownetz zeigt Abb. 8.7.

8.5.1 Agenten

Um Workflows in POTATO mit Hilfe von Helferagenten bearbeiten zu können, sind einige Anpassungen an PIA vorgenommen, sowie eine Reihe neuer Agenten und Interaktionen implementiert worden (vgl. Abschnitt 7.3). Wie in Kapitel 7 ausgeführt, wird ein Workflow-Client Helferagent verwendet, um Benutzern den Zugriff auf das WfMS zu ermöglichen. Zusätzlich werden Activity-Agenten verwendet, um Aufgaben innerhalb des Workflows zu bearbeiten und Ressourcenagenten, die die im Workflow verwendeten Dokumente repräsentieren.

8.5.1.1 Der Workflow-Client Helfer

Der Zugriff des Benutzers als Workflow-Bearbeiter erfolgt über einen Helferagenten, der gegenüber dem WfMS die Rolle des Bearbeiters einnimmt. Er wird parametrisiert mit den WfMS-Zugangsdaten des Anwenders und stellt diesem eine Benutzungsoberfläche zur Verfügung, über die er seine Aufgaben bearbeiten kann. Beim Start des Agenten wird die `login`-Interaktion des WfMS gestartet. Damit nimmt der Workflow-Client die Rolle des Benutzers im WfMS ein. Alle Interaktionen mit dem WfMS laufen über ihn.

Der Workflow-Client Helfer ist ein Helferagent analog zum Chat- oder

Whiteboard-Agenten. Wenn der Agent mit dem Benutzeragenten verbunden ist, meldet er sich mit dessen Zugangsdaten am WfMS an und erhält vom Workitem Dispatcher die Worklist gemeldet, um sie dem Anwender in einem View in der GUI darzustellen. Wählt dieser eine Aufgabe zur Bearbeitung aus, führt der Workflowclient-Agent die entsprechenden Protokolle aus, und legt dem Benutzer die Aktivität vor.

In diesem Prototypen enthält der Workflow-Client die Möglichkeit, neue Workflowinstanzen zu starten. Dafür fragt der Agent eine Liste der bekannten Workflowdefinitionen ab und startet auf Benutzerwunsch hin das Protokoll zum Starten eines neuen Workflows. Die Funktion zum Starten neuer Workflows würde normalerweise nicht jedem Workflow-Teilnehmer zur Verfügung stehen und könnte beispielsweise je nach Berechtigung des Benutzers direkt in der Helferfabrik ein- oder ausgeblendet werden.

Der Benutzer hat zudem die Möglichkeit, eine Aufgabe aus der Worklist auszuwählen. Erfordert die Aufgabe die Verwendung eines Activity-Agenten, veranlasst der Workflow-Client den Activity-Agenten, sich beim Benutzeragenten zu registrieren. Wird eine Aktivität abgeschlossen, kontaktiert der Activity-Agent zunächst den Workflow-Client, der die Aktivität dann an das WfMS zurücksendet.

8.5.1.2 Aktivitätsagenten

Statt die Bearbeitung von Aktivitäten fest im Workflow-Client zu implementieren, werden in POTATO Activity-Agenten verwendet, die die Bearbeitung erlauben. Dabei handelt es sich um spezialisierte Helferagenten, die im WfMS erzeugt und an den Workflow-Client übermittelt werden. Dieser sorgt dafür, dass der Activity-Agent als Helferagent beim zuständigen Benutzeragenten eingebunden wird und dessen Benutzungsoberfläche um die Elemente erweitert, die für die Bearbeitung der jeweiligen Aufgabe erforderlich sind.

Für das Beispiel wurden zwei Typen von Activity-Agenten implementiert, ein TextEditorHelper zum Bearbeiten von Textdokumenten und ein NetEditorHelper für die Erstellung von Petrinetzen. Beide verwenden für die Bearbeitung ein Eclipse-Plugin, das im Benutzeragenten einen Editor öffnet, mit dem die entsprechende Ressource bearbeitet werden kann.

TextEditorHelper Dieser Helferagent für die Bearbeitung von Textdokumenten veranschaulicht, wie die Funktionalität der Eclipse-Umgebung innerhalb von POTATO verwendet werden kann. Die Bearbeitung von Textdokumenten ist eine Standardfunktionalität, die die Eclipse-IDE bereitstellt. Für die Verwendung als Activity-Agent wurden ein Netzagent und ein Agentenplugin implementiert.

Der Agent ist unter anderem dafür verantwortlich den Editor zu öffnen, wenn die Aufgabe gestartet wird und die benötigten Agentenaktionen zu starten, wenn das bearbeitete Dokument gespeichert oder der Editor geschlossen

wird.

Der Standard-Texteditor verwendet Dateien im Workspace des Benutzers, in denen die Texte gespeichert werden. Dies wurde umgestellt auf die Verwendung eines Ressourcenagenten, der über Agentennachrichten angesprochen wird und der die Texte verwaltet. Hierfür wurde ein alternativer `EditorInput` definiert und die `doSave()`-Methode des Editor überschrieben.

NetEditorHelper Der Helferagent zum Bearbeiten von Petrinetzen basiert auf [Mitreiter, 2008]. Darin wurde eine Integration des Renew-Werkzeugs in Eclipse vorgenommen. Der in der Studie erstellte Prototyp erlaubt die Erstellung von Referenznetzen in einem Eclipse-Editor, Zoom in Netzen und die Verwendung des OSGi-Standards für Renew. Da es sich um einen Prototypen handelt, sind allerdings nicht alle Funktionen des Renew-Programms umgesetzt.

Für den Netzeditor-Helferagenten wurde der in der genannten Arbeit erstellte Netzeditor verwendet und erweitert. Auch hier wurden ein Netzagent und ein Agentenplugin erstellt, die gemeinsam die Funktionen des Activity-Agenten implementieren.

Abbildung 8.8 zeigt den Prototypen in Aktion. Auf der linken Seite ist die Liste der Helferagenten zu sehen, die in den Benutzeragenten geladen werden können, rechts wird durch den Workflow-Client Helfer die Worklist angezeigt. Der mittlere Bereich enthält die Editoren, die zum Bearbeiten von Aktivitäten im Workflow verwendet werden, hier den Netzeditor.

8.5.1.3 Ressourcenagenten

Die hier verwendeten Ressourcenagenten für Texte und Petrinetze sind Abwandlungen der Whiteboardressource. Sie dienen dazu, die durch die Helfer bearbeiteten Daten zu speichern. Sie werden durch die Helferagenten erzeugt und im Workflownetz zwischen den Aktivitäten ausgetauscht.

8.5.2 Interaktionen

Für die Integration von Helferagenten in die Prozessinfrastruktur wurden einige Anpassungen an vorhandenen Interaktionen vorgenommen und zum Teil neue Akteure aufgenommen. Anstatt eine Aktivitätsbeschreibung direkt an den ausführenden Agenten zu senden, wird jetzt gegebenenfalls zuerst ein Activity-Agent erzeugt.

8.5.2.1 Aktivität bearbeiten

Die Hauptänderung, die am Pia-System vorgenommen werden musste, um Helferagenten zu integrieren, war eine Anpassung der WFEngine. Sie wurde um eine Decision Component (DC) für die Verwaltung von Activity-Agenten

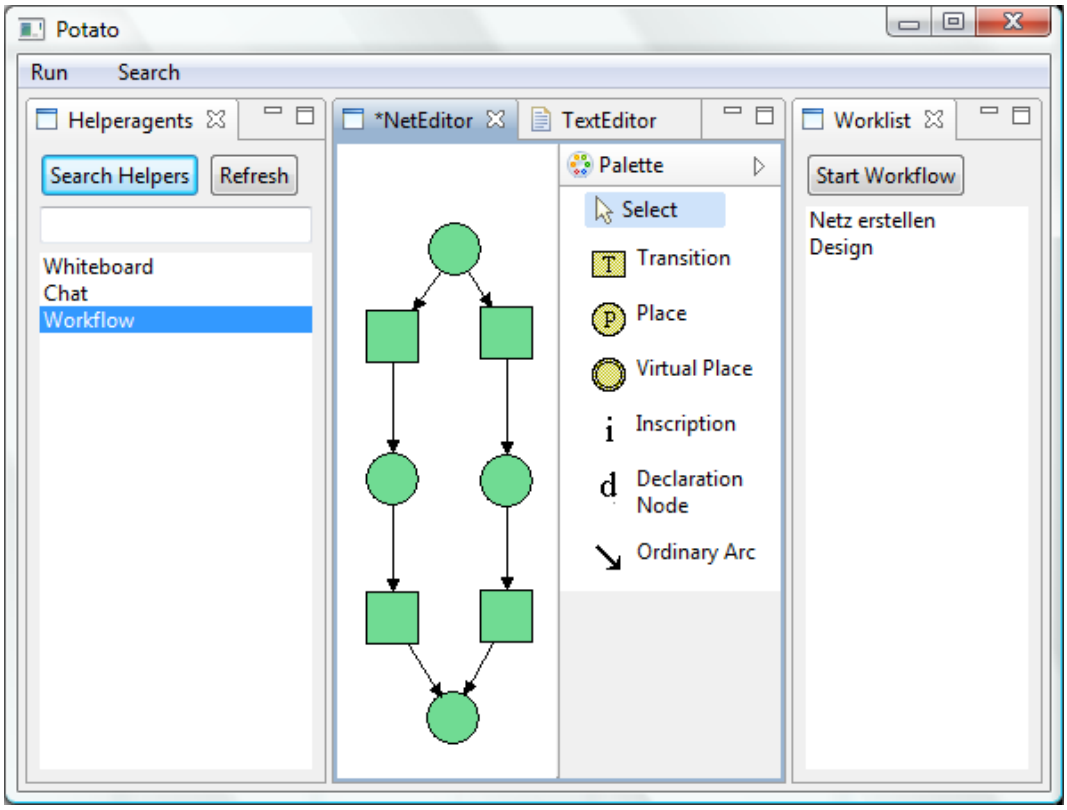

Abbildung 8.8: Aktivität zum Bearbeiten eines Netzes in POTATO

erweitert. Nimmt ein Benutzer ein Workitem zur Bearbeitung an, sendet der Workflow-Client über Workitem Dispatcher und WFES einen Request an die WFEngine. Anstatt einfach die Aktivität aus dem Workitem zu erzeugen, wird sie an die DC überstellt.

Die DC prüft, ob ein Activity-Agent erzeugt werden muss. Ist die Workitem-Definition vom Typ `HelperAgentTask`, wird das Protokoll `startActivityAgent` gestartet. Der Executor und die Activity werden als Parameter übergeben, damit sie in die Wissensbasis des neuen Helferagenten geschrieben werden. Dieser kann beispielsweise darauf Bezug nehmen, um die Aufgabe abzuschließen oder um zusätzliche Informationen abzurufen, welche Aufgabe genau er zu bearbeiten hat. Die DC wartet so lange, bis ein neuer Activity-Agent erzeugt wurde. Dies wird über die Wissensbasis synchronisiert.

Das `startActivityAgent`-Protokoll sendet einen `createHelper`-Request an die Helferfabrik, die dann den neuen Helferagenten konstruiert. Die Adresse des Helferagenten wird in der Activity vermerkt und dann die Schaltung der Task-Transition für die Aufgabe ausgelöst.

Hierdurch werden die Workitem- und Activitylisten aktualisiert, wodurch der Workflow-Client über die Aktivität informiert wird. Da hierin ein Activity-Agent eingetragen ist, startet dieser das Protokoll `registerActivityAgent`, eine Erweiterung des `registerHelper`-Protokolls. Hiermit teilt er

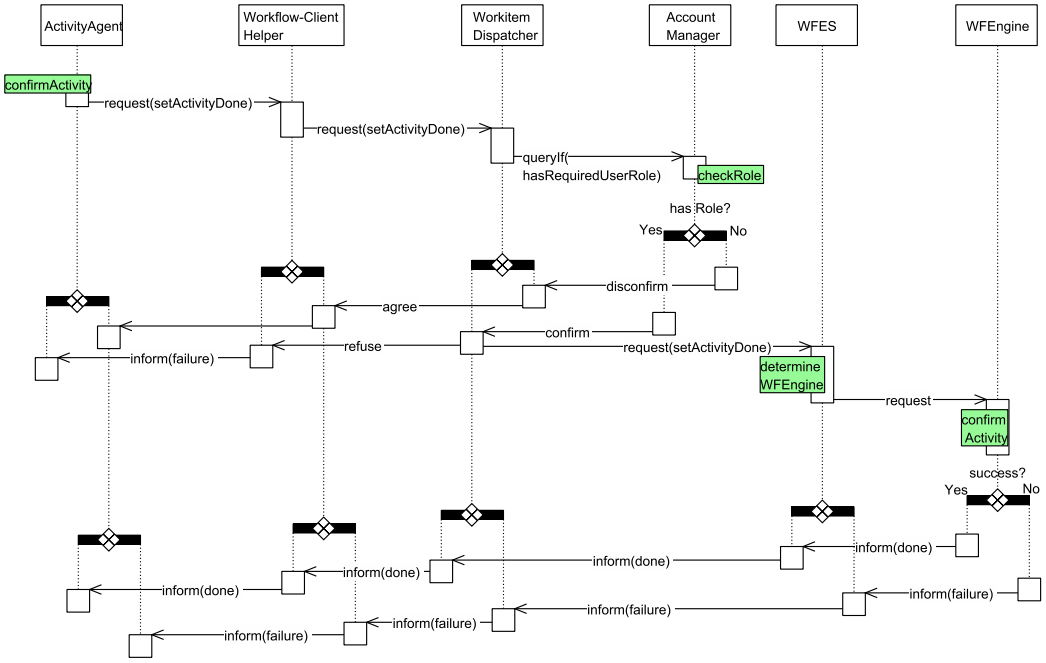

Abbildung 8.9: Interaktion zum Abschließen einer Aktivität

dem Activity-Agenten mit, dass er sich bei dem Benutzeragenten des Workflow-Clients als Helferagent registrieren soll. Dadurch wird der neue Helferagent in der GUI des Benutzers angezeigt und dieser kann damit beginnen, die Aktivität zu bearbeiten. Der Helfer ist durch die Parameterübergabe bereits für die jeweilige Aufgabe vorkonfiguriert und kann beispielsweise direkt eine übergebene Ressource anzeigen.

8.5.2.2 Aktivität abschließen

Wenn eine Aktivität fertig bearbeitet ist, startet der Activity-Agent die Interaktion zum Bestätigen oder Abbrechen der Aktivität, je nachdem, wie die Bearbeitung durch den Benutzer beendet wurde. Im Falle der Beispielaktivitäten wird der Benutzer beim Schließen des Editorfensters gefragt, ob die Aktivität abgeschlossen werden soll. Wird dies verneint, wird sie abgebrochen. Gegebenenfalls wird vorher angefragt, ob der Inhalt des Editors gespeichert werden soll.

Entsprechend wird dann ein `SetActivityDone`- bzw. `SetActivityCanceled`-Objekt konstruiert. Der Executor und die Aktivität, die hierfür erforderlich sind, wurden dem Activity-Agenten durch die Helferfabrik bei der Erzeugung direkt in die Wissensbasis geschrieben, von wo er sie abrufen kann. Da bei der Anforderung des Activity-Agenten seine Adresse noch nicht bekannt war, muss diese als Parameter in der Aktivität nachgetragen werden, da die Aktivitätsbeschreibung sonst nicht mit der beim WfMS bekannten Beschreibung

übereinstimmt.

Abbildung 8.9 zeigt den Ablauf der Interaktion. Der Activity-Agent sendet die `SetActivityDone/Canceled`-Nachricht an den Workflow-Client Agenten, der sie an den Workitem Dispatcher weiterleitet. Dieser prüft, ob der Benutzer die Berechtigung hat, die Aktivität abzuschließen. Ist dies der Fall, wird eine Bestätigung gesendet, andernfalls eine Ablehnung und die Interaktion beendet.

Der Workitem Dispatcher sendet die Anfrage weiter an den Workflow Enactment Service Agenten, der die zuständige Workflow Engine ermittelt und dieser die Anfrage wiederum weiterleitet. Die Workflow Engine schließlich verarbeitet den Request, indem die für die Aktivität zuständige Task-Transition weiter- bzw. zurückgeschaltet wird. Anschließend gibt sie eine Erfolgs- oder Fehlermeldung zurück, die über die gleichen Stationen wieder an den Activity-Agenten zurückgesendet wird.

Der Activity-Agent deregistriert sich daraufhin beim Benutzeragenten. Es wäre möglich, Activity-Agenten innerhalb des Workflows weiterzuverwenden, insbesondere wenn eine Aktivität abgebrochen wurde, dies ist aber zurzeit nicht implementiert. Nicht mehr benötigte Activity-Agenten werden stattdessen aus dem Multiagentensystem gelöscht.

8.6 Zusammenfassung

Dieses Kapitel beschreibt das Vorgehen bei der Implementierung des Prototypen für die verteilte Entwicklungsumgebung. Durch verschiedene iterative Prototypen wird die Anwendbarkeit der in den vorhergehenden Kapiteln aufgestellten Konzepte für die konkrete Softwareentwicklung gezeigt. Zusätzlich zur Anwendung der beschriebenen Strukturierungskonzepte werden auch Teile der Implementation des POTATO-Systems beschrieben.

Der erste vorgestellte Prototyp stellt eine Chatanwendung dar. Dafür werden die Grundbestandteile des HERA-Systems implementiert. Ein Benutzeragent mit einer Benutzungsoberfläche in Eclipse RCP ermöglicht es Benutzern, auf das System zuzugreifen und darin neue Helferagenten zu erzeugen. Dafür wird ein Helferfabrikagent verwendet, der über die Funktionen zur Verwaltung und Erzeugung von Helferagenten verfügt.

Die Chatanwendung verwendet einen Typ von Helferagent, den Chatagenten. Dieser Helferagent wird beim Benutzeragenten registriert und erweitert dessen Funktionsumfang um eigene GUI-Bestandteile und einen Agenten, der damit interagiert. In Verbindung mit anderen Chatagenten wird der Benutzer so in die Lage versetzt, mit anderen Benutzern zu kommunizieren.

Der Benutzeragent und die Helferfabrik, die für diesen Prototypen implementiert wurden, werden in späteren Prototypen genauso weiterverwendet. Dies verdeutlicht ein Prinzip des HERA-Systems, dass für unterschiedliche Anwendungen die Grundelemente unverändert bleiben, konkrete Anwendunge sich lediglich in der Zusammensetzung der Helfer- und Ressourcenagenten

unterscheiden.

Der zweite Prototyp implementiert die Whiteboardanwendung, die bereits in Kapitel 6 als Beispiel verwendet worden war. Als neues Element kommt hier der Ressourcenagent für die Modellierung des Whiteboards hinzu. Dieser Agent stellt eine autonom im System existierende Ressource dar, auf die mittels Helferagenten zugegriffen werden kann. Der Ressourcenagent kann selbst entscheiden, inwieweit er verschiedenen Helferagenten Zugriff gewährt. Die Interaktionen für das Beschreiben des Whiteboards und die Veröffentlichung der Änderungen an Beobachter verdeutlicht, wie Helfer- und Ressourcenagenten sich auf gemeinsame Benutzungsmuster verständigen müssen.

Der folgende Prototyp zeigt die Verwendung des Pia-Systems. Ohne Elemente aus HeRA zu verwenden, wird eine Workflowanwendung für die Bearbeitung von Change Requests gezeigt. Die Prozessdefinition mittels Workflow-Referenznetzen und die Beschreibung der Rollen findet sich so auch in Potato wieder. Die Bearbeitung der Aufgaben erfolgt über die in Pia fest im Client implementierte Formularaufgabe, die lediglich über Ontologieobjekte konfiguriert wird.

Im Gegensatz dazu zeigt dann der finale Prototyp die Verwendung der spezialisierten Agenten aus HeRA zur Erweiterung der Prozessinfrastruktur. Anstelle eines Benutzeragenten, der die Funktionalität zum Zugriff auf das WfMS bereits besitzt, wird hier ein generischer HeRA-Benutzeragent verwendet, wie er bereits im ersten Prototypen gezeigt wurde. In diesen wird dann ein Workflow-Client Helferagent eingebunden, der es erlaubt, Workflows zu starten und Aufgaben aus einer Worklist zur Bearbeitung anzunehmen.

Der Workflow-Client Helfer implementiert aber die Aufgaben zur Bearbeitung nicht selbst, wie dies noch bei Pia der Fall war. Stattdessen werden Activity-Agenten verwendet, die im WfMS erzeugt werden. Der Workflow-Client sorgt dafür, dass diese spezialisierten Helferagenten im Benutzeragenten eingebunden werden, um dort die benötigte Funktionalität für die Aufgabe bereitzustellen.

Für eine neue prozessbasierte Agentenanwendung muss der Prozess unter anderem durch Angabe eines Workflow-Referenznetzes definiert werden. Zusätzlich muss dafür Sorge getragen werden, dass die benötigten Aktivitäten- und Ressourcentypen, die im Workflow verwendet werden, im System bekannt sind. Dies erlaubt ein hohes Maß an Flexibilität bei der Gestaltung der Prozesse, da jederzeit neue Agententypen hinzugefügt werden können.

8.7 Evaluation

In diesem Abschnitt werden die Ergebnisse der Arbeit diskutiert. In der Arbeit werden Konzepte entwickelt, um die Softwareentwicklung mit Petrinetzen und Multiagentensystemen zu verbessern. Hierfür wird das Beispiel einer Unterstützungsumgebung für verteilte Softwareentwicklungsprozesse verwendet. In

der Vergangenheit waren verschiedene Ansätze zur Verwendung von Petrinetzen für die Systementwicklung diskutiert worden. Diese führten zur Entwicklung des MULAN/CAPA-Systems. Während MULAN sich auf die Strukturierung von Referenznetzsystemen nach agentenorientierten Kriterien beschränkt, werden in dieser Arbeit nun neue Strukturierungen innerhalb dieser Multiagentensysteme entwickelt.

Durch diese Strukturierung wird die Entwicklung großer, komplexer Anwendungen ermöglicht. Die vorgeschlagene Strukturierung für Agentenanwendungen basiert auf zwei Dimensionen, die als Struktur und Prozess bezeichnet werden können. Im Folgenden werden die Konzepte zur Unterstützung dieser Dimensionen evaluiert und dann auf die Verbindung der Konzepte im POTATO-System eingegangen. Anschließend wird über die Architektur und den Entwurf und POTATO reflektiert und die Anwendung in der verteilten Softwareentwicklung diskutiert.

8.7.1 HERA

Für die Strukturdimension wird die Verwendung von Agenten zur Repräsentation von Gegenständen im Anwendungsbereich vorgeschlagen. Zu diesem Zweck werden Ideen aus dem objektorientierten Werkzeug und Material Ansatz verwendet, auf Multiagentensysteme übertragen und um weitere agentenorientierte Konzepte erweitert. Das daraus entstehende HERA-System erlaubt es, die Vorteile einer anwendungsnahen Modellierung für interaktive benutzerzentrierte Systeme mit der Flexibilität und Adaptivität von verteilten Multiagentensystemen zu kombinieren.

Agentenplattformen stellen physikalische und logische Orte in einem HERA-System dar. Artefakte können einem Ort oder einer Gruppe zugeordnet werden, indem die entsprechenden Agenten auf die zugehörige Plattform verschoben werden. Durch die Verwendung von Plattformen können temporäre und dauerhafte Organisationseinheiten geschaffen werden, innerhalb derer durch Helferagenten Interaktionen zwischen Benutzern stattfinden und Ressourcen ausgetauscht oder gemeinsam genutzt werden können. Verschiedene Entwicklungsstufen des HERA-System werden in [LEHMANN und MARKWARDT, 2004, LEHMANN et al., 2005, WILLMOTT et al., 2005, MARKWARDT und MOLDT, 2010] beschrieben.

8.7.2 PIA

Geschäftsprozesse werden durch Workflows modelliert, die in Workflow Management Systemen ausgeführt werden. Für die Modellierung eignen sich Petrinetze [OBERWEIS, 1996, AALST, 1998], die beispielsweise in RENEW modelliert und ausgeführt werden können [JACOB et al., 2002, JACOB, 2002b].

Die Prozessdimension spielte aber auch bei der Entwicklung mit MULAN/CAPA im PAOSE-Ansatz schon immer eine wichtige Rolle, da die Mo-

dellierung des Agentenverhaltens in MULAN durch Agentenprotokolle erfolgt, die auch als Workflownetze aufgefasst und untersucht werden können (vgl. hierzu auch [LEHMANN, 2003, LEHMANN und MOLDT, 2004]).

Eine weitere Formalisierung erfolgt nun über die Verwendung von Workflow Management Systemen als Infrastruktur für die Ausführung von Geschäftsprozessen in Multiagentensystemen. Dadurch ergibt sich eine Formalisierung der Interaktionsmuster und damit verbindlichere, kontrollierbare Interaktionen innerhalb, aber vor allem auch zwischen Organisationen oder Organisationseinheiten. Workflow Management Systeme bieten zudem Möglichkeiten der Prozessüberwachung, -steuerung und -optimierung.

Das PIA-System ist die Umsetzung eines Workflow Management Systems unter Verwendung interagierender Agenten [REESE et al., 2005]. Dieses System erlaubt nicht nur die Steuerung der Agenteninteraktionen durch Workflows [MARKWARDT et al., 2006], sondern durch die Nutzung verschiedener Plattformen innerhalb eines Multiagentensystems auch die Verteilung von Workflows innerhalb des Systems und damit auch zwischen Organisationen [REESE et al., 2006b, REESE et al., 2006a, REESE et al., 2006c]. Durch Strukturagenten etwa können eigenständige Workflows verschiedener Organisationen zu einem Gesamtworkflow orchestriert werden [WAGNER, 2009c].

8.7.3 Architektur von POTATO

Die Vereinigung der beiden Dimensionen von Struktur und Prozess ergibt ein komplettes Konzept für die Entwicklung von Agentenanwendungen für verteilte Zusammenarbeit. Durch die Prozessinfrastruktur wird definiert, *wer wann was* in einem System tut, Helferagenten definieren das *wie*, Ressourcenagenten das *womit*.

Dieses Konzept wurde dann in eine Architektur umgesetzt. Das Fernziel dabei ist eine komplette Integration von Struktur- und Prozessaspekten, so dass Agenten und Workflows zu einer Einheit im Sinne der Einheitentheorie (vgl. [TELL und MOLDT, 2005]) verschmelzen, wie dies bereits [REESE, 2010, S. 141ff] als Stufe V der Integration von Agenten und Workflow Management Systemen vorschlägt.

Im Rahmen der Arbeit wurde eine Vorstufe davon als Architektur gewählt. Helfer- und Ressourcenagenten dienen als Baustein in einem agentenorientierten Workflow Management System. Dadurch ist die Perspektive der Anwendungsentwicklung eine prozessorientierte Sichtweise. Die Verwendung von Helfer- und Ressourcenagenten erlaubt aber eine weitergehende Flexibilisierung gegenüber klassischen Workflow Management Systemen.

8.7.4 Entwurf und Anwendung

Das Architekturkonzept wurde im Rahmen des PAOSE-Ansatzes in einen agentenorientierten Entwurf überführt. Für die verschiedenen Teilsysteme wurden

Agentenrollen, Interaktionen und Ontologien definiert.

Im HeRA-System wurden vier verschiedene Agententypen identifiziert: Benutzeragenten repräsentieren den Arbeitsplatz eines Benutzers. Die Helferfabrik verwaltet und erzeugt Helferagenten. Helferagenten unterstützen Benutzer in ihrer Arbeit, indem sie sich in den Benutzeragenten integrieren und dort ihre Funktionalität zur Verfügung stellen. Ressourcenagenten kapseln Arbeitsgegenstände und Ressourcen und überwachen die Einhaltung der Zugriffsmuster darauf.

Beim Entwurf einer Agentenanwendung mit HeRA bleibt der Benutzeragent stets unverändert und auch in der Helferfabrik müssen lediglich die in der jeweiligen Anwendung verwendeten Helferagenten bekannt gemacht werden. Für eine konkrete Anwendung müssen die Helfer- und Ressourcenagenten definiert werden, die die Daten und die Funktionalität der Anwendung ausmachen. Dieses Vorgehen ist analog zu Mulan, wo die Plattform- und Agentennetze ebenfalls unverändert bleiben und neue Agenten nur durch Angabe der Wissensbasis (Daten) und Protokollnetze (Verhalten/Funktionalität) spezifiziert werden.

Die Struktur der Agentenanwendung ergibt sich aus der Konfiguration der verschiedenen beteiligten Plattformen und der darauf befindlichen Agenten. Die organisationale Struktur, sowie Kollaborationsbeziehungen werden durch entsprechende Plattformen dargestellt. Die Arbeitsplätze und die Verteilung der Ressourcen im System ergeben sich aus den Helfer- und Ressourcenagenten, die mit den einzelnen Benutzeragenten verbunden sind.

Diese Konfiguration kann sich dynamisch ändern, wenn zum Beispiel neue Helferagenten hinzugefügt oder Ressourcenagenten erzeugt oder zwischen Agenten oder Plattformen ausgetauscht werden. Ein Beispiel hierfür sind die Aktivitätsagenten, die in Potato für die Bearbeitung von Workflowaufgaben eingesetzt werden und beständig neu erzeugt und zusammen mit den benötigten Ressourcen zu den Workflowteilnehmern gesendet werden. Auch die Konfiguration der Plattformen unterliegt Änderungen, etwa durch organisationale Änderungen, aber auch durch die Formierung neuer Gruppen, die zur Kollaboration eine eigene Plattform erzeugen.

Das Pia-System besteht aus einer Reihe verschiedener Agenten(-rollen), die gemeinsam die Funktionalität eines Workflow Management Systems erbringen. Für die Entwicklung prozessorientierter Anwendungen müssen diese aber nicht weiter angepasst werden. Stattdessen wird die Prozessdefinition in Form eines Workflow-Referenznetzes sowie verschiedener Workflow-Steuerungsdaten (Rollen, Regeln, Aufgaben, Formulare) benötigt.

Die Integration von Pia und HeRA zum Potato-System zielt darauf ab, die Prozessinfrastruktur durch das Herauslösen von Funktionen und Daten und ihre Explizierung durch Helfer- und Ressourcenagenten zu flexibilisieren. Dies wurde anhand von Beispielen gezeigt [Markwardt et al., 2008b, Markwardt et al., 2009a].

Die Einbindung von Workflow-Teilnehmern in die Prozessinfrastruktur er-

folgt in POTATO über einen Workflow-Client Helferagenten. Dadurch kann der generische Benutzeragent aus HERA für alle Benutzer verwendet werden, unabhängig davon, ob sie das Workflow Management System nutzen oder nicht. Auch können Änderungen an den Zugriffsmustern und sogar an den Berechtigungen durch entsprechende Anpassungen des Helferagenten realisiert werden, ohne dass der Benutzeragent direkt davon betroffen ist.

Activity-Agenten erlauben es, bei der Bearbeitung von Workflowaufgaben beliebige Anwendungen zu verwenden und auf eventuell dafür bereits erstellte Helferagenten zuzugreifen [MARKWARDT et al., 2009b]. Der Workflow-Client muss diese Anwendungen selbst nicht kennen, wie das in PIA noch der Fall war. Ressourcenagenten erlauben darüber hinaus, Arbeitsgegenstände im Workflow zu modellieren und zwischen verschiedenen Agenten auszutauschen.

Verfahren zur Anpassung von Workflows zur Laufzeit und entsprechende Interaktionen wurden konzipiert [MARKWARDT et al., 2008a]. Auch hier können Helfer- und Ressourcenagenten sinnvoll zur Vergegenständlichung der Workflowdefinitionen sowie zur Unterstützung der Modellierung eingesetzt werden.

Die Möglichkeit Workflows anzupassen erhöht zum einen die Akzeptanz einer Anwendung bei den betroffenen Anwendern, da diese nicht so stark durch das System in ihren Möglichkeiten eingeschränkt werden müssen. Zum anderen können aber auch die verwendeten Prozesse stetig durch Anwendung und Abwandlung optimiert werden.

8.7.5 Prototypen

Eine Reihe von Prototypen zeigt die Anwendbarkeit der in der Arbeit vorgestellten Konzepte. Zum einen wird damit demonstriert, dass die Implementierung der Komponenten des POTATO-Systems wie beschrieben möglich ist, zum anderen wie ausgehend von diesem Framework auf einfache Weise verteilte Anwendungen entwickelt werden können.

Die Whiteboard-Anwendung zeigt die Erstellung von Agentenanwendungen auf Basis von HERA. Dafür müssen die Standardversionen des Helfer- und Ressourcenagenten um die erforderliche GUI sowie die Protokolle für die spezifische Interaktion erweitert werden. Die vorhandene Werkzeugunterstützung für PAOSE (z.B. [CABAC und MARKWARDT, 2009, CABAC et al., 2009a, CABAC, 2010]) erleichtert diese Arbeit erheblich, so dass ein erfahrener Entwickler in sehr kurzer Zeit eine komplette, verteilte Kollaborationsanwendung erstellen kann.

Die Erstellung von Workflow-Anwendungen mit PIA wird anhand einer Change Management Anwendung gezeigt. Auch hier ist die Entwicklung der tatsächlichen Anwendung sehr schnell, sofern die vom System angebotenen Standardoperationen ausreichend sind, im Beispiel wurde lediglich die Formularaufgabe verwendet. In diesem Fall beschränkt sich die Implementation auf eine Konfiguration des Prozesses und der beteiligten Rollen, Rechte und

Aufgaben.

Im finalen Prototypen wurden einige der vorgeschlagenen Möglichkeiten zur Integration von HeRA und Pia zum Potato-System gezeigt. Die gezeigte Anwendung verwendet den Workflow-Client Helferagenten für die Teilnehmereinbindung und Aktivitätsagenten für die Ausführung der Aufgaben. Die Erstellung einer kleinen Anwendung mit Potato ist aufwändiger als mit HeRA oder Pia alleine, da sowohl die Helfer- bzw. Aktivitätsagenten und Ressourcenagenten erstellt als auch der Prozess mit seinen Parametern definiert werden müssen. Andererseits sind aber mit Potato weitaus umfangreichere und flexiblere Anwendungen möglich.

8.7.6 Verteilte Softwareentwicklung

Das in der Arbeit verwendete Anwendungsbeispiel war ein System zur Unterstützung verteilter Softwareentwicklungsprozesse. Als Anforderungen an ein solches System wurden insbesondere flexible Werkzeuge und eine Prozessunterstützung zur Koordination verschiedener Prozessbeteiligter identifiziert. Diese Aspekte werden durch das in der Arbeit beschriebene Potato-System unterstützt.

Zur Verdeutlichung werden die in der Arbeit beschriebenen Konzepte auf das eingangs erwähnte Beispiel eines verteilten Softwareprojektes angewendet. Dabei werden insbesondere die in Abschnitt 2.1 genannten Ebenen des Application/IT-Alignments deutlich.

Der Anwendungskontext auf der obersten Ebene war in diesem Projekt die Unterstützung der weltweit verteilten Zweigstellen einer Studentenorganisation. Diese Organisation verwaltet Praktikumsplätze für Auslandspraktika in Unternehmen und geeignete Studenten, ermittelt passende Paarungen und betreut die Praktika vor Ort.

Die Software hierfür bildet die zweite Modellebene. In dem genannten Projekt ist sie nicht agentenorientiert entwickelt worden, eine entsprechende Betrachtung wäre aber möglich und naheliegend. Die einzelnen Zweigstellen können als Agentenplattformen aufgefasst werden, auf denen sich Agenten für die Studenten und Partnerunternehmen befinden.

Die Entwicklung dieses Systems ist die dritte Betrachtungsebene. Sie erfolgte in Kollaboration zwischen einer Entwicklungsabteilung in Indien, Administration in Deutschland und Vertretern der Kundenorganisation in den Niederlanden und gegen Ende weltweit.

Die vierte Ebene, die Entwicklungsunterstützung, bildeten in diesem Projekt eine Vielzahl verschiedener Anwendungsprogramme und Hilfsmittel, von Sourcecode Verwaltung über Bugtracker bis hin zu Chatprogrammen und natürlich Email und Telefon.

Eine Unterstützung dieses Gesamtsystems durch Potato wäre auf den Ebenen zwei und vier möglich gewesen. Sowohl die Anwendungssoftware als auch die Entwicklungsumgebung hätten als verteiltes Multiagentensystem im-

plementiert werden können, um so die Zusammenarbeit besser zu unterstützen. Agentenplattformen können in einer solchen verteilten Umgebung dazu verwendet werden, die am System beteiligten Organisationen und Organisationseinheiten sowie ihre Beziehungen untereinander zu modellieren [Wester-Ebbinghaus et al., 2007, Wester-Ebbinghaus, 2010].

Kapitel 9

Schlussbetrachtung

Abschließend werden hier die einzelnen Kapitel reflektiert und die jeweiligen Ergebnisse nochmals zusammengefasst. Anschließend wird ein Ausblick auf die Bereiche gegeben, die im Rahmen dieser Arbeit nicht abschließend behandelt werden konnten, und aufgezeigt, welche Aspekte in weiterführenden Arbeiten noch vertieft werden können.

9.1 Ergebnisse

Die vorliegende Arbeit stellt neue Ansätze und Konzepte für die Entwicklung von Unterstützungssystemen in verteilten Umgebungen vor. Die Arbeit zeigt, wie petrinetzbasierte Multiagentensysteme strukturiert werden können, um die Entwicklung komplexer Systeme damit besser zu ermöglichen.

Für die Softwareentwicklung mit Petrinetzen werden Modelle aus dem Werkzeug und Material Ansatz aufgegriffen und in Petrinetzmodelle überführt. Dadurch wird es möglich, die dort vorgefundenen, anwendungsnahen Metaphern und Entwurfsprinzipien auch für den Systementwurf mit Netzen zu nutzen.

Dies gelingt insbesondere durch die Verwendung von Referenznetzen und durch das Prinzip der Netze in Netzen, die einen mächtigen Formalismus für die Strukturierung und das Zusammenwirken verschiedener Systembestandteile ermöglichen.

Ferner werden diese Modelle angewendet auf die agentenorientierte Softwareentwicklung mit Petrinetzen. Der Paose-Ansatz wird verwendet, um eine Multiagentenanwendung zu erstellen, dabei werden unter anderem mit Helfer- und Ressourcenagenten neue Agententypen eingeführt, die für die Anwendungsentwicklung verwendet werden können.

Das Anwendungsbeispiel, das die Arbeit motiviert, ist die Unterstützung der verteilten, arbeitsteiligen Softwareentwicklung. Auf der Basis der Anforderungen, die sich aus diesem Anwendungsbereich ergeben, wird ein Konzept für das Potato-System (Process-Oriented Tool Agents for Team Organization)

aufgestellt. Das POTATO-System dient dazu, die Entwicklung agenten- und prozessorientierter Anwendungen zur Unterstützung verteilter Zusammenarbeit zu ermöglichen.

Das System selbst baut auf dem agentenorientierten Unterstützungssystem HERA (HElper and Ressource Agents) und der Prozessinfrastruktur für Agentenanwendung (PIA) auf. Die beiden Systeme sind miteinander kombiniert worden, so dass sie gemeinsam die agentenorientierte Architektur des POTATO-Systems bilden. Eine Reihe von Prototypen verdeutlicht die Konzepte und den Einsatz des POTATO-Systems.

9.1.1 Motivation

Kapitel 2 liefert eine Motivation und Einführung in das Thema der Arbeit. Business/IT-Alignment wird als Ziel organisationsorientierter Softwareentwicklung formuliert. Dafür werden verschiedene Ebenen der Betrachtung aufgestellt, die eine weitergehende Untergliederung der Arbeit motivieren. Das Anwendungsbeispiel der verteilten Softwareentwicklungsumgebung wird über die Einordnung in das Business/IT-Alignment motiviert und anhand der verschiedenen am System beteiligten Parteien dargestellt. Die verschiedenen Nutzergruppen werden durch individuell konfigurierbare Hilfsmittel und Ressourcen in ihrer Arbeit unterstützt. Die anfallende Arbeit wird durch eine Prozessinfrastruktur gesteuert.

Die Agentenorientierung, Workflowsteuerung sowie Unterstützung von Ressourcen und Hilfsmitteln werden als zentrale Entwurfsdimensionen herausgearbeitet und zueinander in Beziehung gesetzt. Die Synthese dieser Dimensionen führt zum Konzept des POTATO-Systems. Dieses System verwendet ein petrinetzbasiertes Multiagentensystem, um Hilfsmittel und Ressourcen in verteilten Workflowprozessen bereitzustellen.

9.1.2 Grundlagen

In Kapitel 3 werden die Grundlagen und Vorarbeiten aufbereitet, auf denen die Arbeit aufbaut. Ausgehend von den zentralen Dimensionen des Systementwurfs, die in Kapitel 2 für diese Arbeit herausgearbeitet wurden, wird hier ein Überblick über die verschiedenen Themenbereiche gegeben, die in der Arbeit zusammengebracht werden.

Petrinetze dienen als Modellierungs- und Implementationswerkzeug sowie für die Analyse von Workflows. Agenten und Multiagentensysteme sind das softwaretechnische Paradigma, das für die Umsetzung verwendet wird, der PAOSE-Ansatz liefert den dazu passenden Entwicklungsansatz. CSCW als verwandtes Thema beleuchtet Schwerpunkte der Benutzerinteraktion. Der Werkzeug und Material Ansatz liefert Strukturierungsideen, die mit dem HERA-System auf Multiagentensysteme übertragen und ausgebaut werden. Workflow Management Systeme sind die Grundlage für die Prozesssteuerung in POTATO.

Diese Aufbereitung der zugrundeliegenden Konzepte erlaubt es, die weitere Arbeit zu strukturieren und bei der Entwicklung des POTATO-System auf einem breiten fachlichen Fundament aufzubauen.

9.1.3 Verteilte Softwareentwicklungsprozesse

Das Anwendungsbeispiel für die Unterstützung verteilter Prozesse, das diese Arbeit behandelt, ist das Feld der verteilten Softwareentwicklung. Kapitel 4 beschäftigt sich daher mit der Definition dieses Anwendungsbereiches, um festzustellen, welche Art von Unterstützung hier erforderlich ist.

Der Begriff des Software Engineering ist dabei in zweierlei Hinsicht von Interesse, einerseits für das Vorgehen in der Arbeit, andererseits als Anwendungsbereich von POTATO. Aus den Kriterien zur Beurteilung der Softwarequalität sowie Prinzipien, mit denen die Qualität erhöht werden kann, ergeben sich eine Reihe von Ansatzpunkten, wie eine verteilte Entwicklungsumgebung durch geeignete Hilfsmittel und Prozessunterstützung zur Qualitätssteigerung beitragen kann.

Verteilte Softwaresysteme und Algorithmen weisen eine Reihe von Besonderheiten und Problemstellungen im Gegensatz zu nicht-verteilter Software auf. Serviceorientierte Architekturen sind ein Konzept für ihre Implementierung. Berührungspunkte und Unterschiede zum agentenorientierten Ansatz wurden beleuchtet. Aufgrund ihrer Flexibilität und Adaptivität werden Multiagentensysteme für die weitere Konzeption herausgestellt, zumal sich eine Dienstorientierung auch hiermit abbilden lässt.

Für die konkrete Unterstützung verteilter Softwareentwicklungsprozesse werden Entwicklungsumgebungen verwendet. Verschiedene existierende Systeme sowohl für die allgemeine, als auch für die verteilte Softwareentwicklung vereinen vor allem die Verwaltung verschiedener Hilfsmittel und Ressourcen, teilweise überwachen sie auch die Entwicklungsprozesse. Ausgehend davon wurde ein Konzept, das eine Werkzeugunterstützung mit einer Prozesssteuerung verbindet, für eine agentenorientierte, verteilte Entwicklungsumgebung aufgestellt.

Kapitel 4 behandelt damit den Anwendungsbereich der Arbeit. Als Ziel für die Bearbeitung wird die Bereitstellung von Hilfsmitteln und Ressourcen für die Softwareentwicklung, sowie die Steuerung der dabei verwendeten Prozesse herausgestellt. Dadurch wird dem Anwender optimal ermöglicht, seine anfallenden Aufgaben zu bearbeiten, da er die dafür erforderlichen Gegenstände an seinem Arbeitsplatz vorfindet, die flexibel an die jeweiligen Bedürfnisse angepasst werden können. Die Prozesssteuerung unterstützt ihn zudem in der Organisation seiner Arbeit und der Zusammenarbeit mit anderen. Dieses Konzept wird im weiteren Verlauf der Arbeit konkretisiert und agentenorientiert umgesetzt.

9.1.4 Das POTATO-System

Das Thema in Kapitel 5 ist die Entwicklung einer Architektur und eines Grobentwurfs für das POTATO-System als konzeptionelle Grundlage für die Entwicklung verteilter Unterstützungssysteme. Diese Architektur basiert auf einer Strukturierung von Multiagentensystemen nach zwei zentralen Kriterien, Struktur und Prozessen. Die in Kapitel 3 und 4 für CSCW allgemein und für die verteilte Softwareentwicklung im Speziellen ausgeführten Anforderungen dienen als Grundlage für den Entwurf der Architektur.

Die Hauptaspekte von POTATO sind die Unterstützung von Benutzern durch Helfer- und Ressourcenagenten und die Steuerung der Anwendungsprozesse durch ein flexibles, verteiltes Workflow Management System. Analog zu den Betrachtungen in [REESE, 2010] können verschiedene Architekturalternativen ausgewählt werden. Aus diesen verschiedenen Stufen der Integration verwendet POTATO ein Modell, bei dem Helfer- und Ressourcenagenten als Bausteine in einem agentenorientierten Workflow Management System verwendet werden. Die Prozesssteuerung setzt auf dem Konzept von Helfer- und Ressourcenagenten auf, um die Vorteile der beiden Systeme zu vereinen.

Ein Referenznetzmodell überführt die Grundidee von POTATO in ein erstes ausführbares Modell und stellt die verschiedenen Bestandteile von POTATO und ihre Interaktion dar. Die Verwendung von Referenznetzen für die Modellierung erlaubt es, ein komplexes, nebenläufiges System auf einfache und anschauliche Weise zu modellieren. In RENEW lässt sich dieses Modell in der Simulation ausführen. So können die Eigenschaften des Systems bereits in einem frühen Entwurfsstadium am laufenden System untersucht werden.

Gerade für verteilte, interagierende, autonome Einheiten wie Agenten, zeigt sich hier einmal mehr, dass Netze in Netzen ein mächtiges Modellierungswerkzeug für den Systementwurf sind. Auf der Basis dieses konzeptionellen Modells werden in den folgenden Kapiteln die verschiedenen Bestandteile des POTATO-Systems in eine konkrete agentenorientierte Architektur umgesetzt. Dabei wird dann gezeigt, dass Referenznetze ebenso für die Implementierung geeignet sind.

9.1.5 Helfer- und Ressourcenagenten

Kapitel 6 stellt vor, wie Entwurfskonzepte aus der objektorientierten Programmierung, in diesem Fall aus dem Werkzeug und Material Ansatz die agentenorientierte Softwareentwicklung unterstützen können. Statt Agenten ungeordnet oder nach ad-hoc definierten Mustern interagieren zu lassen, definiert das HERA-System konkrete Agentenrollen für Helfer und Ressourcenagenten, die durch Benutzeragenten verwendet werden können. Dadurch wird ein Artefaktkonzept für Multiagentensysteme eingeführt, das sich vollständig in bestehende Systeme integrieren lässt, da hierfür keine neuen Entitäten erforderlich sind. Gegenstände werden als spezielle Art von Agenten modelliert.

Mit dem Werkzeug und Material Ansatz wird hierfür ein vielfach bewährtes Vorgehensmodell aus der objektorientierten Softwareentwicklung aufgegriffen und für Multiagentensysteme nutzbar gemacht. Die Verwendung von Agenten anstelle von einfachen Objekten für diese Gegenstände des Anwendungsbereiches erlaubt adaptive, proaktive Helfer, die einen Benutzer in seiner Arbeit unterstützen, anstelle einfacher Werkzeuge, die lediglich benutzt werden. Die Möglichkeit, einen Benutzeragenten durch Helfer- und Ressourcenagenten beliebig in seiner Funktionalität zu erweitern, bietet neue strukturierte Möglichkeiten zur Erstellung flexibler Agentenanwendungen auf der Basis einer eingängigen Metapher. Zudem erlaubt die Verwendung von Multiagentensystemen, gegebenenfalls in Verbindung mit mobilen Agenten, den Austausch von Helfern und Ressourcen zwischen den verschiedenen Beteiligten des Systems, menschlichen Benutzern ebenso wie autonomen Softwarekomponenten. Die Helfer- und Ressourcenagenten, die einem Benutzer zugeordnet sind, bevölkern dessen Arbeitsplatz im System. Er kann sich diesen nach seinen Vorstellungen konfigurieren und an jedem Ort wieder herstellen lassen. Die Arbeitsplätze der verschiedenen Benutzer sind als verteilte Agentenplattformen in einem Multiagentensystem realisiert. Durch die Nutzung mobiler Agenten, die zwischen den Plattformen migrieren, lassen sich Gegenstände zwischen verschiedenen Arbeitsplätzen austauschen.

Die Konzeption des HeRA-Systems erfolgt entsprechend dem Paose-Vorgehen durch einen agentenorientierten Entwurf und eine Spezifikation der beteiligten Agentenrollen, der Interaktionen zwischen diesen Rollen sowie der verwendeten Ontologie. Das Beispiel einer Whiteboard-Anwendung zur gemeinsamen, verteilten Bearbeitung einer Ressource verdeutlicht diese Konzepte.

HeRA liefert eine Verbesserung der Architektur von Multiagentensystemen, da eine weitere Strukturierung des Systems nach klar definierten Kriterien erfolgt. Ähnlich, wie in Mulan die grundlegenden Plattform- und Agentennetze unverändert bleiben und das Verhalten im Wesentlichen durch die Inhalte der Wissensbasen sowie durch die Protokollnetze und Entscheidungskomponenten spezifiziert wird, bleiben in HeRA grundlegende Agenten wie der Benutzeragent und die Helferfabrik unverändert. Neue Funktionalität wird durch die Definitionen der jeweiligen spezifischen, anwendungsbezogenen Helferagenten hinzugefügt.

Der Paose Ansatz und das Renew-Werkzeug liefern eine Reihe aufeinander abgestimmter Modelle, die es erlauben, den Systementwurf über mehrere Stufen zu verfeinern, um am Ende zu einem ausführbaren Referenznetzmodell für eine Multiagentenanwendung zu gelangen.

HeRA erlaubt es, beim Systementwurf eine Reihe verschiedener Betrachtungsweisen zu vereinen. Auf der einen Seite lassen sich die Anforderungen des einzelnen Anwenders durch die Betrachtung der verschiedenen Arbeitsplätze und ihrer Gegenstände abbilden. Auf der anderen Seite können durch die Umsetzung als Multiagentensystem Akteurs- und Ortskonzepte integriert werden,

die eine bessere Darstellung und Unterstützung verteilter, kooperativer Arbeit ermöglichen.

Verschiedene Arten von Multiagentenanwendungen lassen sich mit Hilfe von HeRA realisieren, in dieser Arbeit wird der Fokus aber auf die Integration mit den prozessorientierten Ansätzen aus Pia gelegt.

9.1.6 Die agentenorientierte Prozessinfrastruktur

Kapitel 7 diskutiert Konzepte für die Unterstützung und Steuerung von Geschäftsprozessen mit Multiagentensystemen. Dafür wird die Prozessinfrastruktur für Agentenanwendungen (Pia) beschrieben [Reese, 2010]. Pia ermöglicht die Ausführung von Workflowprozessen in Multiagentensystemen und wurde parallel zum HeRA-System entwickelt. Im Rahmen dieser Arbeit wurden einzelne Teile davon mitentwickelt [Reese et al., 2005, Reese et al., 2006a]. Aus den Anforderungen für eine verteilte Softwareentwicklungsumgebung ergaben sich konkrete Fragestellungen, die sich in der Konzeption und der Architektur von Pia niederschlugen. Die konzeptionelle Ausarbeitung dieser Prozessinfrastruktur war aber nicht Teil der vorliegenden Arbeit.

In dieser Arbeit lag das Hauptaugenmerk vielmehr auf der Anwendung und Erweiterung von Pia in Verbindung mit den Helfer- und Ressourcenagenten aus HeRA. In der Architektur von Potato können die Helfer- und Ressourcenagenten des HeRA-Systems als Bausteine innerhalb der Prozessunterstützung verwendet werden. Die Verwendung der HeRA-Agenten in Pia erlaubt eine Flexibilisierung des Systems, da Funktionen und Ressourcen der Prozessinfrastruktur als eigenständige Agenten dargestellt werden können. Dadurch kann von der Flexibilität, Autonomie und Adaptivität profitiert werden, die Agenten bieten. Dies wurde auf verschiedene Arten in der Konzeption und später in den Prototypen umgesetzt.

Eine Flexibilisierung der Workflowrollen wurde beispielsweise erreicht, indem die Einbindung der Teilnehmer entkoppelt wird. Anstatt die Benutzungsoberfläche und Protokolle für den Zugriff auf das Workflow Management System im Benutzeragenten festzulegen, wie das im ursprünglichen Pia-System noch der Fall war, wird ein generischer HeRA-Benutzeragent verwendet. Die Funktionalität für den Zugriff auf das WfMS fügt ein Workflow-Client Helferagent hinzu.

Eine Flexibilisierung der Aufgabenbearbeitung und der Prozessdefinition ermöglicht der Activity-Agent durch eine Auslagerung der konkreten Methoden zur Bearbeitung einer Aufgabe aus dem Workflow-Client in einen spezialisierten Helferagenten je Aufgabentyp. Ressourcen, die im Workflow erforderlich sind, wie etwa konkrete Falldaten, werden durch Ressourcenagenten modelliert und können durch die Activity-Agenten bearbeitet und zwischen verschiedenen Teilnehmern ausgetauscht werden. Hierdurch können neue Arten von Aufgaben und Ressourcen dem System hinzugefügt werden, ohne den Workflow-Client anpassen zu müssen. Zusätzliche Flexibilität für die Anwender

des Workflow Management Systems bietet ein Petrinetz-basiertes Verfahren, das die Modifikation von Workflowprozessen während der Ausführung erlaubt. Dadurch können die vorgegebenen Prozesse bei Bedarf jederzeit an die tatsächlichen Bedürfnisse angepasst werden und so das Problem zu starrer Vorgaben eines WfMS reduziert werden. Mit der Integration von HeRA und Pia zum Potato-System steht nun ein komplettes konzeptionelles Framework für Multiagentenanwendungen zur Verfügung. Das in Kapitel 5 aufgestellte Referenznetzmodell für Potato steht als konkretes Multiagentensystem zur Verfügung, in dem HeRA und Pia als Bausteine Verwendung finden.

Für die Abbildung und Steuerung von Workflowprozessen sind Petrinetze ein beliebter Formalismus. Potato fügt weitere Geschäftsprozess-relevante Merkmale, wie die Modellierung der Akteure und ihrer Interaktionen, sowie insbesondere der im Workflow verwendeten Ressourcen und Aktivitäten hinzu. Durch die Verwendung eines durchgängigen Netzmodells können die Beziehungen zwischen diesen Entitäten deutlich gemacht werden. Netze in Netzen liefern dafür die nötige Abstraktion durch die Kapselung der einzelnen Einheiten.

9.1.7 Prototypen

Kapitel 8 demonstriert die Umsetzung der für Potato aufgestellten Konzepte in konkrete Implementierungen. Verschiedene iterative Prototypen zeigen, wie diese Konzepte die Entwicklung konkreter Systeme unterstützen können, indem Muster für die Interaktion der Agenten im System vorgegeben werden. Dabei werden auch die jeweils benötigten Bestandteile des Potato-Systems mit ihrer Implementation vorgestellt. Der erste Prototyp zu HeRA ist eine einfache Chatanwendung. Ein Benutzeragent wird durch die Verwendung eines Helferagenten um eine Chatfunktionalität erweitert, indem ein neuer Agent erzeugt und in den bestehenden eingebunden wird. Der Benutzeragent und die Helferfabrik finden in unveränderter Form auch in nachfolgenden Prototypen Verwendung.

Damit wurde gezeigt, wie ein dynamisch konfigurierbares Multiagentensystem verwendet werden kann, um Benutzern individuelle Funktionalität nach Bedarf anzubieten. Mit dem Helferagenten wird ein Agent als reaktiver Gegenstand verwendet und so ein Artefaktkonzept für Multiagentensysteme geschaffen. Der Benutzeragent als Arbeitsplatz wird als konzeptionelle Plattform für Helferagenten durch Nachrichtenkommunikation realisiert.

Die Umsetzung der Whiteboard-Anwendung aus Kapitel 6 fügt als neues Element den Ressourcenagenten hinzu. Der Whiteboard-Ressourcenagent erlaubt die Darstellung einer Ressource als autonome Entität im System, die über den Zugriff selbstständig entscheiden kann.

Der Materialagent führt das Artefaktkonzept einen Schritt weiter, indem ein Agent die Rolle eines passiven Gegenstandes einnimmt, der lediglich durch andere Agenten bearbeitet wird. Er erlaubt damit eine Benutzerinteraktion, die rein indirekt über den Whiteboardagenten als asynchrones Kommunikations-

medium erfolgt. Im vorgestellten Prototypen bleibt der Materialagent ortsfest, mobile Agenten sind aber ebenfalls eine Option für den Nachrichtenaustausch zwischen Agenten.

Bezüglich der Prozessunterstützung in POTATO zeigt zunächst ein Change Management-Workflow die Funktionsweise von PIA. Dieser Workflow verdeutlicht die Verwendung von Workflow-Referenznetzen zur Prozessdefinition sowie die Definition von Workflowrollen und -aufgaben.

Dieser Prototyp zeigt PIA [REESE, 2010] als petrinetzbasiertes Workflow Management System. Im Gegensatz zu implizit definierten Prozessen, die sich aus den Interaktionen der beteiligten Agenten ergeben, ermöglicht die Vorgabe der Prozesse eine Strukturierung auf höherer Ebene und damit eine zielgerichtetere Systementwicklung.

Die konkrete Ausführung der Aktivitäten im Workflow setzt bei den beteiligten Agenten voraus, dass sie alle im Workflow vorkommenden Aktivitäten beherrschen. Im Beispiel wurde daher eine generische Formularaufgabe verwendet, die über die Workflowdefinition parametrisiert wurde. An dieser Stelle setzt das POTATO-System an, um eine Flexibilisierung der Aufgaben zu erreichen, indem diese zusammen mit dem Workflow definiert werden.

Der finale Prototyp verwendet dann das komplette POTATO-System, mit verschiedenen interagierenden Agententypen. Ein Workflow-Client Helferagent ermöglicht den Zugriff auf das Workflow Management System, verschiedene Activity-Agenten und Ressourcenagenten sind für die Bearbeitung der Aufgaben in einem einfachen Beispielworkflow aus dem Softwareentwicklungsumfeld zuständig. Hierdurch wird das Zusammenspiel der verschiedenen Bestandteile von POTATO verdeutlicht. Eine Prozessteuerung koordiniert die Zusammenarbeit der verschiedenen Akteure im System, die konkrete Ausgestaltung der Tätigkeiten erfolgt durch die Bereitstellung der benötigten Ressourcen im Workflow ebenso wie die Helferagenten, mit denen diese bearbeitet werden können.

Durch den Activity-Agenten werden HERA und PIA miteinander verbunden, indem die Helferagenten für die Bearbeitung einer Workflowaufgabe dem zuständigen Bearbeiter direkt mitgegeben werden. Dies erlaubt eine komplette Prozessunterstützung in verteilten, kooperativen Anwendungen, die durch das Hinzufügen neuer Agenten dynamisch an völlig neue Aufgabenstellungen angepasst werden kann.

Durch diese Prototypen wird deutlich, wie der Entwurf von Multiagentensystemen von den durch POTATO beschriebenen Strukturen profitieren kann. Helferagenten erlauben eine einfache Erweiterung interaktiver Systeme um neue Funktionen, die an zentraler Stelle im System für alle interessierten Anwender hinterlegt werden können. Die Workflowsteuerung erlaubt, die Abläufe zwischen den Akteuren innerhalb einer Anwendung zu konfigurieren und direkt die Helfer und Ressourcen zuzuordnen, die hierfür erforderlich sind.

Zusammenfassend betrachtet stellt diese Arbeit Strukturen für die Strukturierung und den Entwurf verteilter Multiagentenanwendungen auf. Durch die Vorgabe spezialisierter Agententypen und ihres Zusammenspiels wird die

Entwicklung komplexer Systeme erleichtert. Das HeRA-System macht Konzepte aus der objektorientierten Softwareentwicklung für die agentenorientierte Softwareentwicklung nutzbar, indem spezielle Agententypen zur Abbildung von aktiven und passiven Artefakten im System verwendet werden.

Weiterhin wurde im Rahmen dieser Arbeit an der Entwicklung der Prozessinfrastruktur für Multiagentensysteme (Pia) mitgewirkt. Dieses agentenorientierte Workflow Management System erlaubt es, die Interaktion der verschiedenen Agenten in einem Multiagentensystem mit Hilfe formalisierter Prozessbeschreibungen zu spezifizieren und damit auf einer höheren Ebene zu strukturieren. Damit kann die Systementwicklung zielgerichteter die im Anwendungskontext vorkommenden Prozesse unterstützen.

Diese beiden Teilaspekte des agentenorientierten Systementwurfs wurden zu einem gemeinsamen System, Potato, zusammengeführt. Damit können sowohl die Aktionen und dafür verwendeten Hilfsmittel und Arbeitsgegenstände der Akteure, als auch die Interaktionen zwischen diesen Akteuren in einem gemeinsamen Framework entwickelt werden.

9.2 Ausblick

Abschließend gibt dieser Abschnitt einen Überblick über verschiedene Themenbereiche, die in dieser Arbeit nicht behandelt werden konnten, sowie über verwandte Themen und weiterführende Arbeiten. Die Beschreibung des Potato-Systems in dieser Arbeit ist vornehmlich konzeptioneller Natur. Im Folgenden werden einige Themenfelder beleuchtet, in denen Erweiterungen möglich sind.

Die Themengebiete Sicherheit und Benutzbarkeit sind wichtige Bereiche der praktischen Anwendungsentwicklung. Aufgrund der konzeptionellen Natur dieser Arbeit konnten diese Themen nur andiskutiert werden. Ebenso war eine effiziente Implementierung nicht das Ziel der Arbeit. Hier sind weiterführende Arbeiten denkbar, die diese Aspekte aufgreifen.

Bei der Beschreibung der Prozessinfrastruktur werden Login/Logout-Interaktionen beschrieben, mit denen sich Benutzer gegenüber dem System identifizieren. Auf der Basis dieser Anmeldung erfolgt die Zuweisung der Aufgaben im System. Verschiedene Interaktionen beinhalten außerdem eine Überprüfung der Berechtigung zur Ausführung durch den Benutzer. Bei der Konfiguration von Helferagenten durch die Helferfabrik ist der Rückgriff auf eine Sicherheitsinfrastruktur andiskutiert worden, um Helfer bei der Erstellung bereits so zu konfigurieren, dass nur erlaubte Operationen durchgeführt werden können.

In der Vergangenheit war das Vorgehen bei der Gestaltung von Benutzungsoberflächen im Paose Ansatz meist unstrukturiert. GUIs wurden in der Regel in Form spezialisierter Benutzeragenten für eine konkrete Anwendung implementiert. In dieser Arbeit wird ein Benutzeragent vorgestellt, der in verschie-

denen Multiagentenanwendungen wiederverwendet werden kann und zudem dem Anwender ein hohes Maß an Flexibilität bei der Gestaltung seines persönlichen Arbeitsplatzes einräumt. Dies wird erreicht durch einen generischen Benutzeragenten, der durch Helfer- und Ressourcenagenten erweitert werden kann.

Die Verwendung von Petrinetzen in Entwurf und Implementierung eröffnet verschiedene Möglichkeiten der Verifikation gewünschter Systemeigenschaften. Dies wurde beispielhaft für die Soundness von Workflowprozessen gezeigt. Weiterführende Arbeiten können die formale Fundierung etwa für die Komposition [Köhler-Bussmeier, 2009] und Vereinbarung [Aalst et al., 2010] interorganisationaler Workflows liefern.

Die Verwendung von Agenten in der Softwareentwicklung bietet verschiedene Möglichkeiten, die in dieser Arbeit nur angedeutet werden konnten. Mit der Verwendung adaptiver Agenten für die Selbstorganisation verteilter Softwaresysteme beschäftigt sich [Sudeikat, 2010]. Weitere interessante Aspekte sind Mobilität [Cabac et al., 2009b] und Agentensysteme für Organisationen [Wester-Ebbinghaus, 2010].

In [Wester-Ebbinghaus et al., 2007] wurden Ansätze vorgestellt, Organisationen und organisationales Handeln mit Hilfe von Multiagentensystemen zu modellieren. [Wester-Ebbinghaus, 2010] entwickelt dieses Modell weiter zu einer Referenzarchitektur für die organisationsorientierte Softwareentwicklung. Die dort behandelten Ansätze gehen bezüglich der Größe der behandelten Systeme über den Rahmen dieser Arbeit hinaus, sind dafür aber weniger konkret in der Ausgestaltung der beschriebenen Systeme. In dieser Arbeit wird ein kleinerer Bereich intra- und interorganisationaler Software dafür ausführlicher beschrieben.

Die prototypische Implementierung beschränkt sich bewusst darauf, die zentralen Konzepte des Potato-Systems darzustellen. Hier sind verschiedene Erweiterungen denkbar. Die GUI des Benutzeragenten wurde mit Hilfe des Eclipse RCP Frameworks implementiert. Eine weitergehende Integration mit der Eclipse IDE und Jazz könnte vorgenommen werden, um die Möglichkeiten der Unterstützung verteilter Entwicklung weiter auszuloten und mit bereits vorhandenen Ansätzen aus der Eclipse-Community zu integrieren.

Eine andere Möglichkeit besteht darin, alternative GUI-Frameworks und Konzepte auf ihre Eignung für die Implementation des Benutzeragenten und der Helferagenten zu untersuchen. In einem aktuellen Lehreprojekt wird die Verwendung eines Web-Gateways für Agenten erprobt. Webanwendungen haben den Vorteil, dass kein spezieller Client installiert werden muss, um auf das System zuzugreifen. Desktopanwendungen, wie die Rich Client Implementation des Prototypen sind meist schneller und haben einen größeren Funktionsumfang. Diese und andere Punkte sind bei einer Implementation für einen konkreten Anwendungsbereich abzuwägen.

Die Konzepte des Potato-Systems sind für Petrinetzagenten auf der Mulan/Capa Plattform dargestellt worden. Denkbar ist aber auch die Anwen-

dung in anderen Kontexten, sowohl in anderen Multiagentensystemen als auch beispielsweise in serviceorientierten Architekturen. In Abschnitt 4.2.2 wurde auf die Ähnlichkeiten zwischen Multiagentensystemen und serviceorientierten Architekturen hingewiesen. Die Konzepte aus POTATO könnten auf Services übertragen werden.

Ebenso wäre auch die Verwendung des agentenorientierten POTATO-Systems im Rahmen einer serviceorientierten Architektur denkbar, sowohl als Dienstanbieter wie auch als -nutzer. Agenten können als Implementation eines Dienstes verwendet werden, ein Webservice-Gatewayagent hierfür existiert bereits [WILLMOTT et al., 2005]. Ebenso können beispielsweise Helferagenten auf Web Services zugreifen, die in einer Organisation angeboten werden und diese zu ihren eigenen Zwecken orchestrieren [MOLDT und ORTMANN, 2004].

Das in der Arbeit betrachtete Anwendungsbeispiel einer verteilten Entwicklungsumgebung bietet ebenfalls Ansatzpunkte für weiterführende Arbeiten. Zur Unterstützung eines konkreten Entwicklungsansatzes, etwa des PAOSE-Ansatzes, müssten die erforderlichen Prozesse definiert, sowie die Helfer- und Ressourcenagenten implementiert werden. Der Netzeditorhelfer, der im Prototypen dargestellt wurde, ist ein Beispiel für die Verwendung von RENEW in einem Helferagenten. Weitere Werkzeuge aus dem PAOSE-Ansatz können auf ähnliche Art und Weise zu einer integrierten Entwicklungsumgebung ausgebaut werden. Bei Internetanwendungen ist aktuell ein Trend zu Community-Systemen und benutzergenerierten Inhalten zu beobachten, der unter den Stichworten „Social Media“ und „Web 2.0“ zusammengefasst wird [HASS et al., 2007, ALPAR und BLASCHKE, 2008]. POTATO liefert eine Infrastruktur, um Kollaborationswerkzeuge, wie sie in sozialen Netzwerken verwendet werden, auch in produktiven Geschäftsprozessen im Sinne eines „Enterprise 2.0“ [MCAFEE, 2006] einzusetzen.

Anhang A

Referenznetzmodell

In diesem Anhang werden die vollständigen Netze zu den Referenznetzmodellen aus Abschnitt 5.5 aufgeführt.

A.1 HeRA

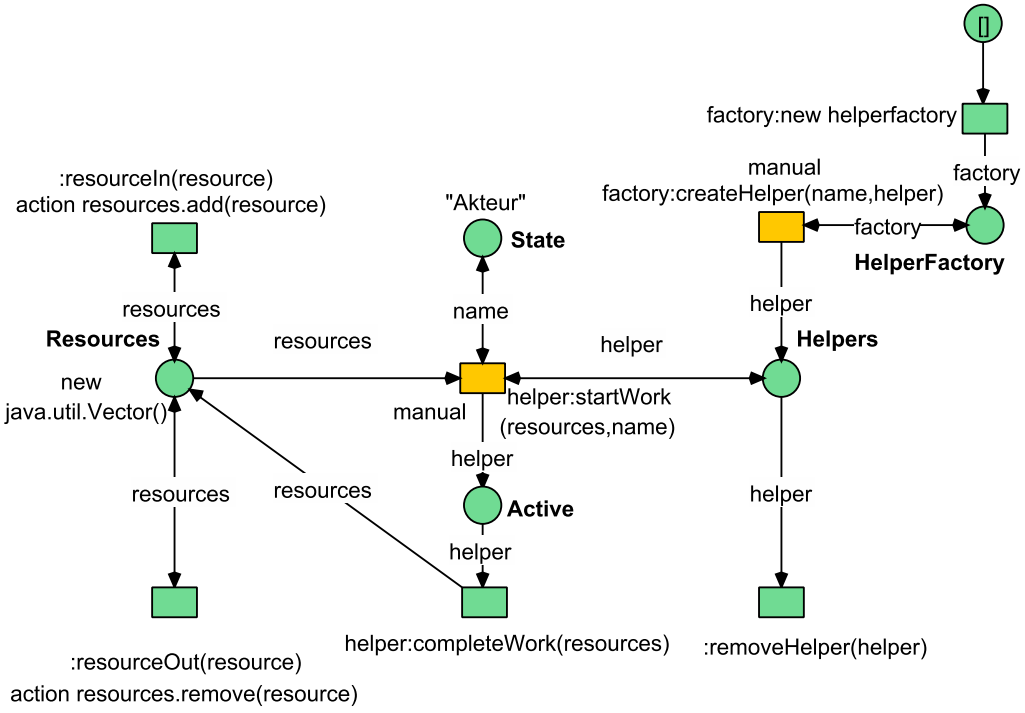

Abbildung A.1: `actor` Benutzeragent

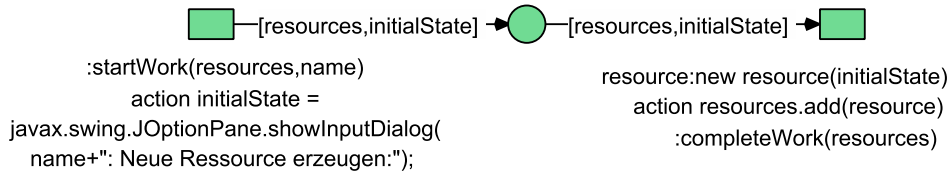

Abbildung A.2: `creatorhelper` Helferagent zum Erzeugen einer Ressource

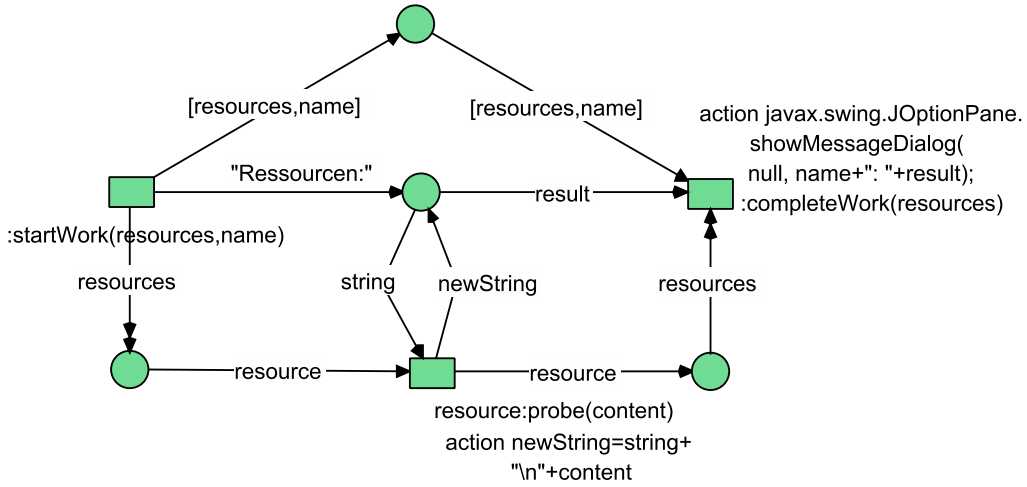

Abbildung A.3: `displayhelper` Helferagent zum Anzeigen einer Ressource

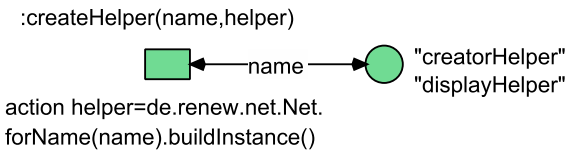

Abbildung A.4: `helperfactory` Helferfabrik zum Erzeugen der Helferagenten

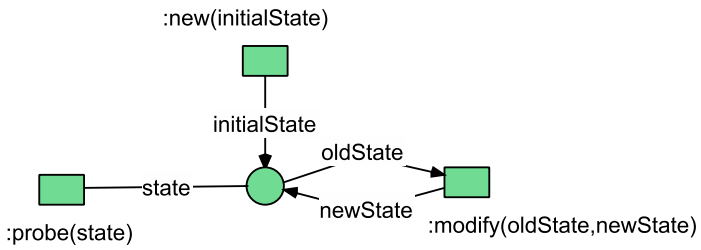

Abbildung A.5: `resource` Ressourcenagent

A.2 Pia

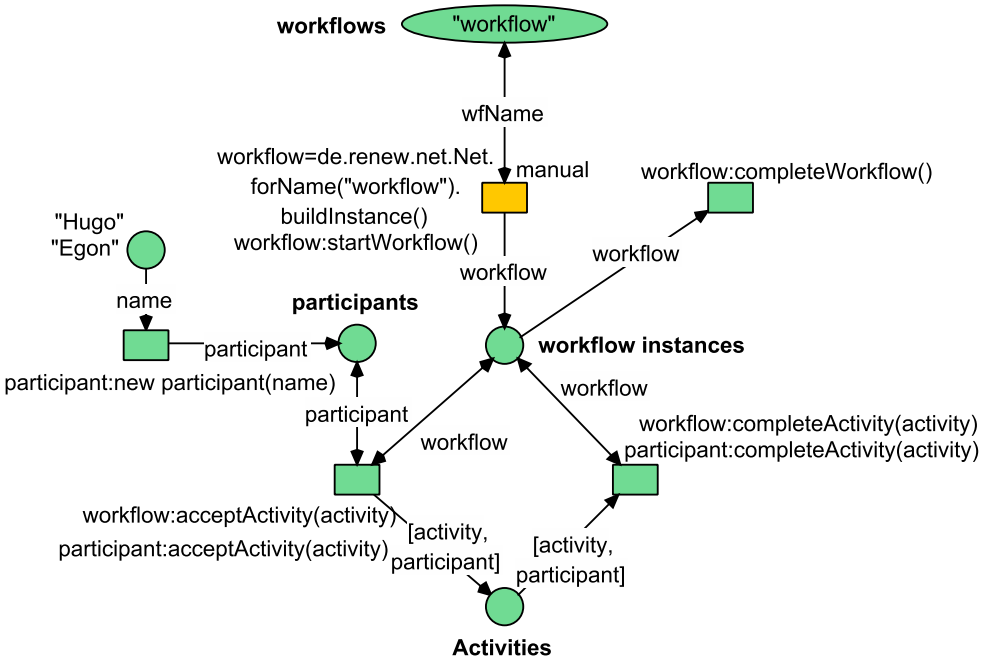

Abbildung A.6: `wfms` Workflow Management System

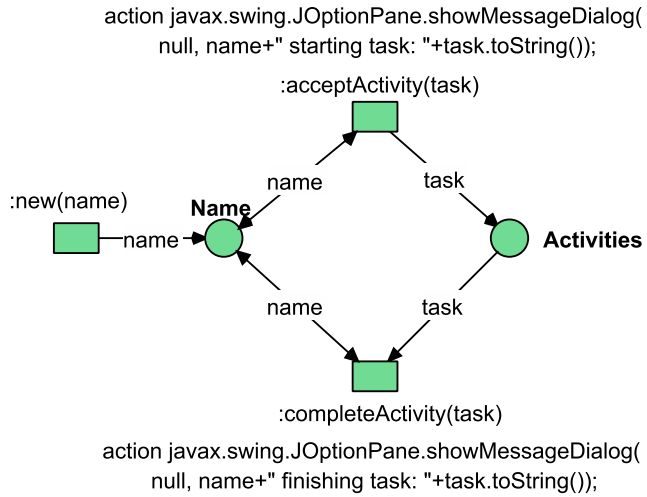

Abbildung A.7: `participant` Workflow Teilnehmer

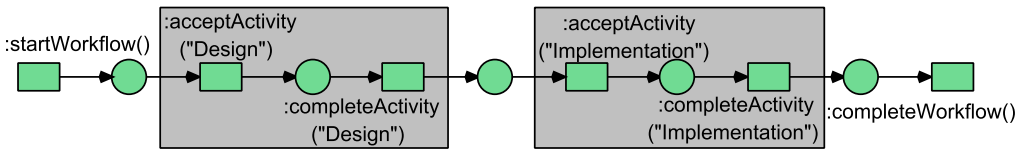

Abbildung A.8: `workflow` Beispiel-Workflow

A.3 Potato

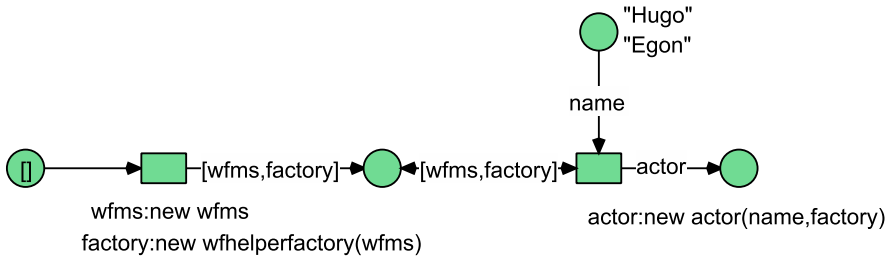

Abbildung A.9: `potatocontrol` Startnetz für Potato

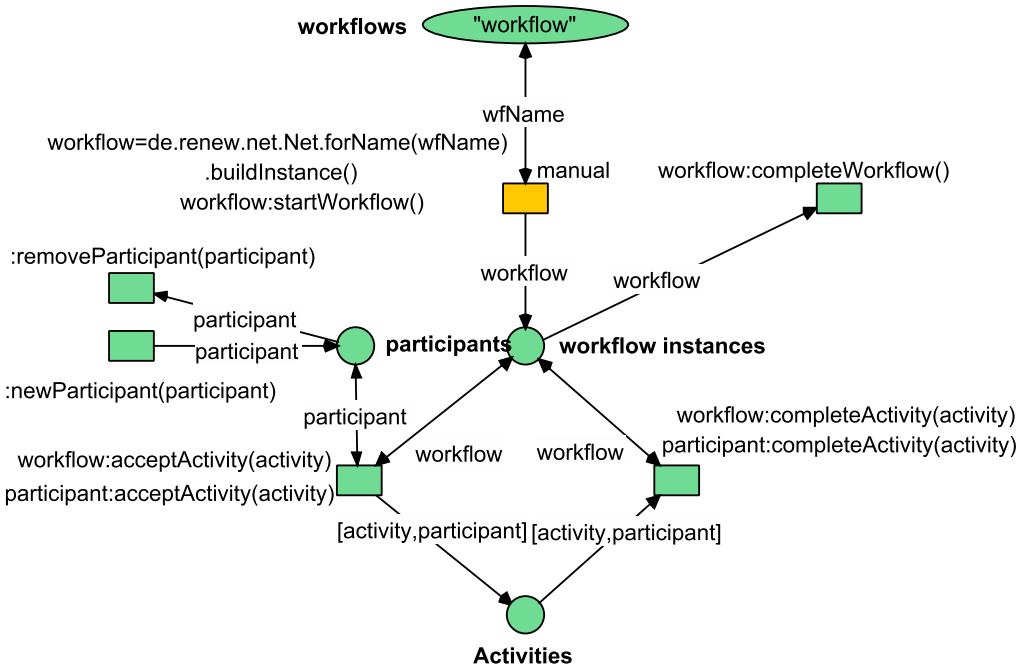

Abbildung A.10: `wfms` Workflow Management System

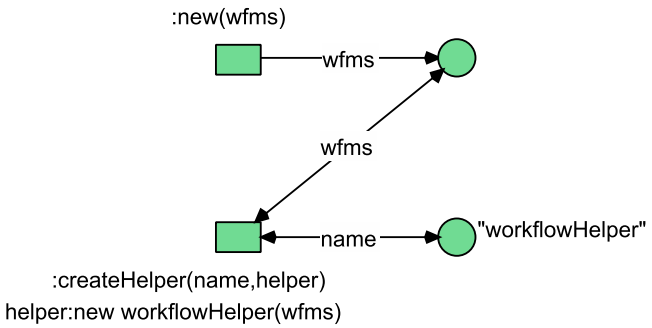

Abbildung A.11: `wfhelperfactory` Helferfabrik für POTATO

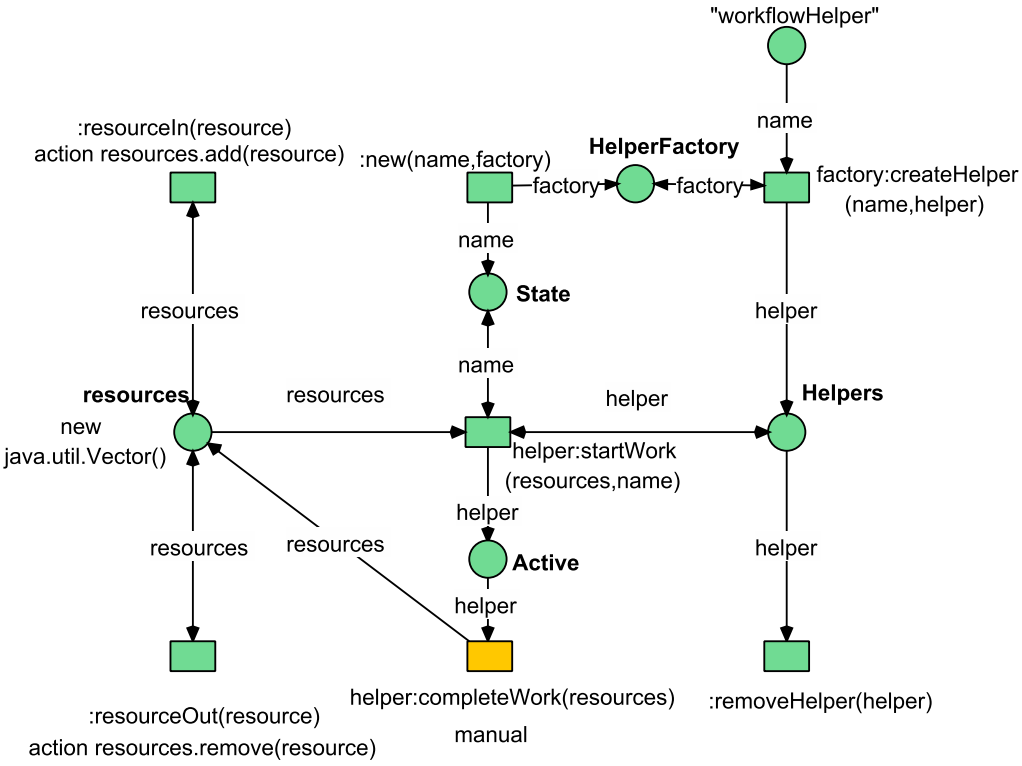

Abbildung A.12: `actor` Benutzeragent

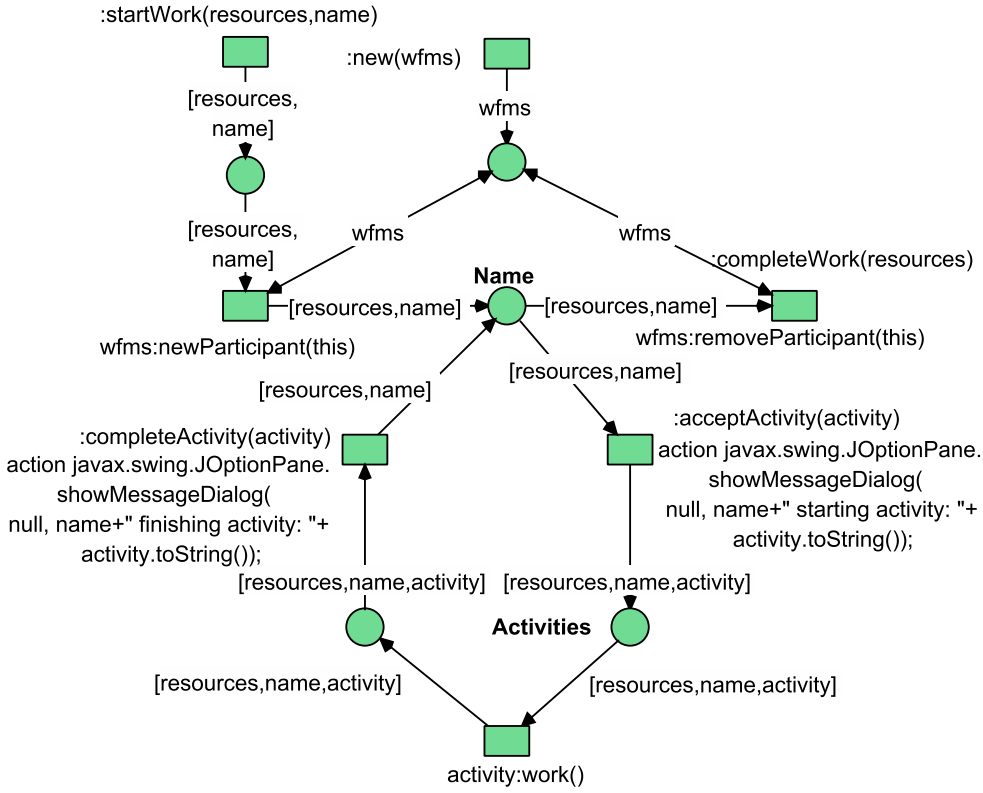

Abbildung A.13: `workflowHelper` Workflow-Client Helferagent

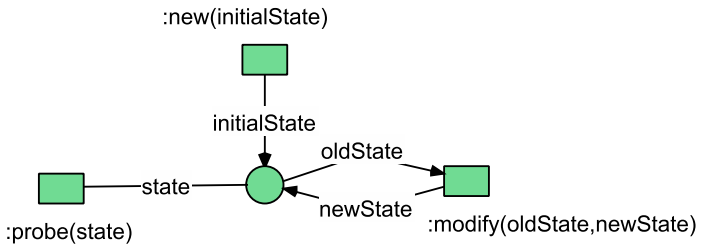

Abbildung A.14: `resource` Ressourcenagent

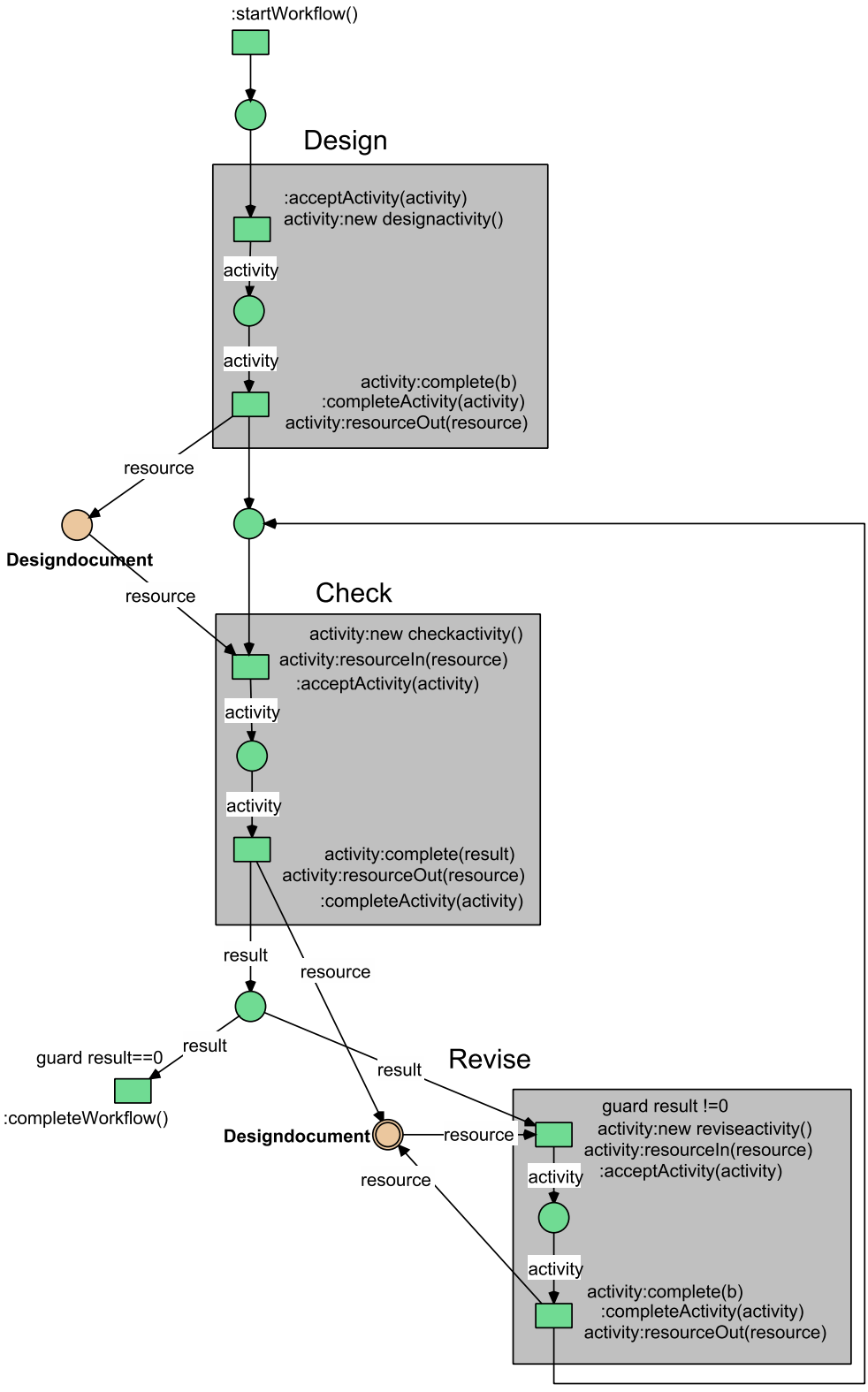

Abbildung A.15: `workflow` Beispiel-Workflow für POTATO

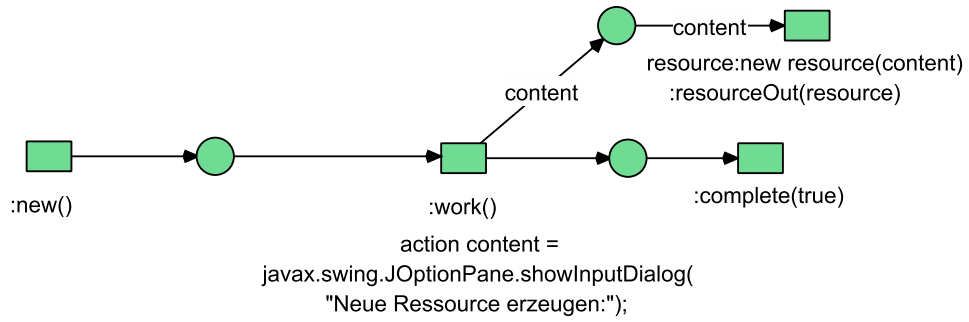

Abbildung A.16: `designactivity` Aktivitätsagent zum Erzeugen einer Ressource

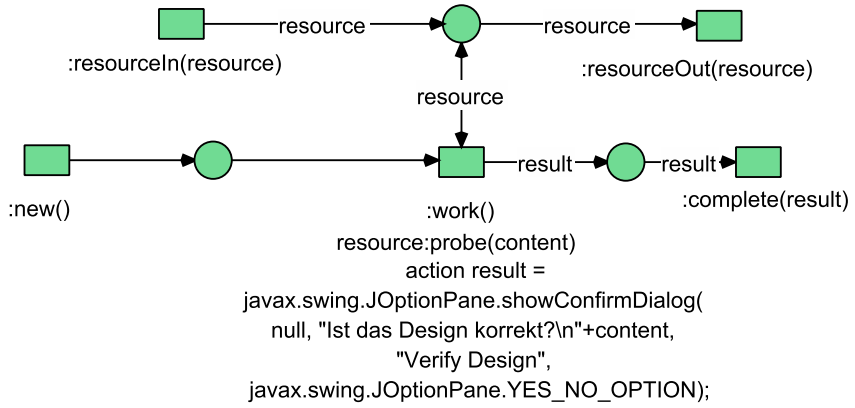

Abbildung A.17: `checkactivity` Aktivitätsagent zum Prüfen einer Ressource

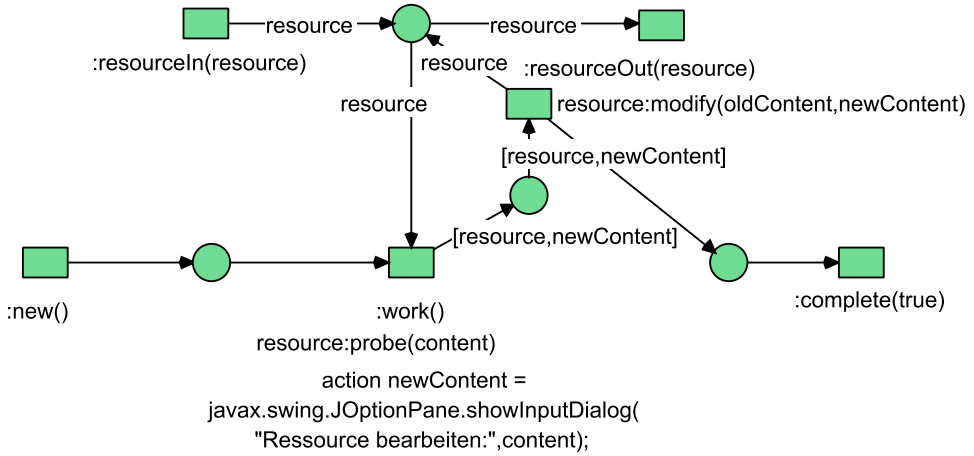

Abbildung A.18: `reviseactivity` Aktivitätsagent zum Überarbeiten einer Ressource

Abbildungsverzeichnis

Literaturverzeichnis

[Aalst, 1997] Aalst, Wil van der (1997). *Verification of Workflow Nets*. In: *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, Bd. 1248 d. Reihe *Lecture Notes in Computer Science*, S. 407–426, London, UK. Springer-Verlag.

[Aalst, 1998] Aalst, Wil van der (1998). *The Application of Petri Nets to Workflow Management*. The Journal of Circuits, Systems and Computers, 8(1):21–66.

[Aalst, 2001] Aalst, Wil van der (2001). *Exterminating the Dynamic Change Bug: A Concrete Approach to Support Workflow Change*. Information Systems Frontiers, 3(3):297–317.

[Aalst, 2011] Aalst, Wil van der (2011). *Woflan – The Woflan Website*. `http://is.tm.tue.nl/research/edlbp/software/woflan/woflan_2_3.htm` [Abgerufen 21. Dezember 2011].

[Aalst et al., 1994] Aalst, Wil van der, K. v. Hee und G. Houben (1994). *Modelling Workflow Management Systems with High-Level Petri Nets*. In: Michelis, G. de, C. Ellis und G. Memmi, Hrsg.: *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, S. 31–50.

[Aalst et al., 2010] Aalst, Wil van der, N. Lohmann, P. Massuthe, C. Stahl und K. Wolf (2010). *Multiparty Contracts: Agreeing and Implementing Interorganizational Processes*. The Computer Journal, 53(1):90–106.

[Aalst et al., 1999] Aalst, Wil van der, D. Moldt, R. Valk und F. Wienberg (1999). *Enacting Interorganizational Workflows Using Nets in Nets*. In: Becker, Jörg, M. Mühlen und M. Rosemann, Hrsg.: *Proceedings of the 1999 Workflow Management Conference Workflow-based Applications*, Working Paper Series of the Department of Information Systems, S. 117–136, University of Münster, Department of Information Systems. Working Paper No. 70.

[Alpar und Blaschke, 2008] Alpar, Paul und S. Blaschke (2008). *Web 2.0-eine empirische Bestandsaufnahme*. Vieweg+Teubner Verlag, Wiesbaden.

[Balzert, 2001] Balzert, Helmut (2001). *Lehrbuch der Software-Technik: Software-Entwicklung*. Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2. Aufl.

[Baumöl, 2006] Baumöl, Ulrike (2006). *Methodenkonstruktion für das Business/IT Alignment*. Wirtschaftsinformatik, 48(5):314–322.

[Beck, 1999] Beck, Kent (1999). *Embracing change with extreme programming*. Computer, 32(10):70–77.

[Becker-Pechau, 2009a] Becker-Pechau, Petra (2009a). *Quelltextannotationen für stilbasierte Ist-Architekturen*. In: Engels, Gregor, R. Reussner, C. Momm und S. Sauer, Hrsg.: *Design for Future - Langlebige Softwaresysteme. 1. Workshop des GI-Arbeitskreises „Langlebige Softwaresysteme (L2S2)“*, S. 3–14.

[Becker-Pechau, 2009b] Becker-Pechau, Petra (2009b). *Stilbasierte Architekturprüfung*. In: Fischer, Stefan, E. Maehle und R. Reischuk, Hrsg.: *GI Jahrestagung'09*, Bd. 154 d. Reihe *LNI*, S. 3264–3278. GI.

[Becker-Pechau et al., 2006] Becker-Pechau, Petra, B. Karstens und C. Lilienthal (2006). *Automatisierte Softwareüberprüfung auf der Basis von Architekturregeln*. In: Biel, Bettina, M. Book und V. Gruhn, Hrsg.: *Software Engineering*, Bd. 79 d. Reihe *LNI*, S. 27–37. GI.

[Bleek und Wolf, 2010] Bleek, Wolf-Gideon und H. Wolf (2010). *Agile Softwareentwicklung - Werte, Konzepte und Methoden*. dpunkt.verlag, Heidelberg, 2. aktualisierte und erw. Aufl.

[BmI, 2011] BmI (2011). *Die Beauftragte der Bundesregierung für Informationstechnik - Bundesministerium des Innern, Das V-Modell XT*. `http://www.v-modell-xt.de` [Abgerufen 21. Dezember 2011].

[Booth et al., 2004] Booth, David, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris und D. Orchard (2004). *Web Services Architecture*. World Wide Web Consortium Working Group Note.

[Bosch et al., 2002] Bosch, Tobias, O. Gries, H. Kausch, M. Klenski, K. Lehmann, M. Morales, V. Seegert und A. Vilner (2002). *Agentenorientierte Implementierung des Spiels "Die Siedler von Catan"*. Projektbericht, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Boulila et al., 2003] Boulila, Naoufel, A. H. Dutoit und B. Bruegge (2003). *D-Meeting: an Object-Oriented Framework for Supporting Distributed Modelling of Software*. In: Damian, Daniela und F. Lanubile, Hrsg.: *International Workshop on Global Software Development, International Conference on Software Engineering*, S. 34–38.

[Brand, 2009] Brand, Andreas (2009). *Softwareentwicklung im Netzwerk. Kooperation, Hierarchie und Wettbewerb in einem Open Source-Projekt*. Rainer Hampp Verlag, Mering.

[Bratman, 1987] Bratman, Michael (1987). *Intentions, Plans, and Practical Reason*. Harvard University Press, Cambridge, MA.

[Breitling et al., 2000] Breitling, Holger, C. Lilienthal, M. Lippert und H. Züllighoven (2000). *The JWAM Framework: Inspired by research, reality-tested by commercial utilization*. In: *OOPSLA 2000 Workshop: Methods and Tools for Object-Oriented Framework Development and Specialization*.

[Bresciani et al., 2004] Bresciani, Paolo, A. Perini, P. Giorgini, F. Giunchiglia und J. Mylopoulos (2004). *Tropos: An Agent-Oriented Software Development Methodology*. Autonomous Agents and Multi-Agent Systems, V8(3):203–236.

[Brooks, 1995] Brooks, Frederick P. (1995). *The mythical man-month (anniversary ed.)*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.

[Bugzilla, 2011] Bugzilla (2011). *Bugzilla Website*. `http://www.bugzilla.org` [Abgerufen 21. Dezember 2011].

[Cabac, 2007] Cabac, Lawrence (2007). *Multi-Agent System: A Guiding Metaphor for the Organization of Software Development Projects*. In: Petta, Paolo, Hrsg.: *Proceedings of the Fifth German Conference on Multiagent System Technologies*, Bd. 4687 d. Reihe *Lecture Notes in Computer Science*, S. 1–12, Leipzig, Germany. Springer-Verlag.

[Cabac, 2010] Cabac, Lawrence (2010). *Modeling Petri Net-Based Multi-Agent Applications*. Dissertation, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg. `http://www.sub.uni-hamburg.de/opus/volltexte/2010/4666/`.

[Cabac und Dörges, 2007] Cabac, Lawrence und T. Dörges (2007). *Tools for Testing, Debugging and Monitoring Multi-agent Applications*. In: [Moldt et al., 2007], S. 209–213.

[Cabac et al., 2005] Cabac, Lawrence, M. Duvigneau, M. Köhler, K. Lehmann, D. Moldt, S. Offermann, J. Ortmann, C. Reese, H. Rölke und V. Tell (2005). *PAOSE Settler Demo*. In: *First Workshop on High-Level Petri Nets and Distributed Systems (PNDS) 2005*, Vogt-Kölln Str. 30, D-22527 Hamburg. Universität Hamburg, Fachbereich Informatik.

[Cabac und Knaak, 2007] Cabac, Lawrence und N. Knaak (2007). *Process Mining in Petri Net-based Agent-oriented Software Development*. In: [Moldt et al., 2007], S. 7–21.

[Cabac et al., 2006] Cabac, Lawrence, N. Knaak, D. Moldt und H. Rölke (2006). *Analysis of Multi-Agent Interactions with Process Mining Techniques*. In: *Multiagent System Technologies. 4th German Conference, MATES 2006 Erfurt, Germany. Proceedings*, Bd. 4196 d. Reihe *Lecture Notes in Computer Science*, S. 12–23, Berlin, Heidelberg, New York. Springer-Verlag.

[Cabac und Markwardt, 2009] Cabac, Lawrence und K. Markwardt (2009). *Modeling the System Organization of Multi-Agent Systems in Early Design Stages with Coarse Design Diagrams*. In: [Moldt et al., 2009], S. 34–43.

[Cabac et al., 2009a] Cabac, Lawrence, K. Markwardt und J. Schlüter (2009a). *ImageNetDiff: Finding Differences in Models*. In: [Moldt et al., 2009], S. 156–161.

[Cabac et al., 2003] Cabac, Lawrence, D. Moldt und H. Rölke (2003). *A Proposal for Structuring Petri Net-Based Agent Interaction Protocols*. In: Aalst, Wil van der und E. Best, Hrsg.: *24th International Conference on Application and Theory of Petri Nets, Eindhoven, Netherlands, June 2003*, Bd. 2679 d. Reihe *Lecture Notes in Computer Science*, S. 102–120, Berlin, Heidelberg, New York. Springer-Verlag.

[Cabac et al., 2008] Cabac, Lawrence, D. Moldt und J. Schlüter (2008). *Adding Runtime Net Manipulation Features to MulanViewer*. In: *15. Workshop Algorithmen und Werkzeuge für Petrinetze, AWPN'08*, Bd. 380 d. Reihe *CEUR Workshop Proceedings*, S. 87–92. Universität Rostock.

[Cabac et al., 2009b] Cabac, Lawrence, D. Moldt, M. Wester-Ebbinghaus und E. Müller (2009b). *Visual Representation of Mobile Agents – Modeling Mobility within the Prototype MAPA*. In: [Duvigneau und Moldt, 2009], S. 7–28.

[Cardoso et al., 2007] Cardoso, Jorge, M. Hepp und M. D. Lytras, Hrsg. (2007). *The Semantic Web: Real-World Applications from Industry*, Bd. 6 d. Reihe *Semantic Web And Beyond Computing for Human Experience*. Springer-Verlag, New York.

[Carl, 2004] Carl, Timo (2004). *Entwicklung eines agentenbasierten verteilten Workflow-Management-Systems mit Referenznetzen*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

[Chinnici et al., 2007] Chinnici, Roberto, J.-J. Moreau, A. Ryman und S. Weerawarana (2007). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. World Wide Web Consortium, Recommendation REC-wsdl20-20070626.

[CruiseControl, 2011] CruiseControl (2011). *CruiseControl Homepage*. `http://cruisecontrol.sourceforge.net` [Abgerufen 21. Dezember 2011].

[CVS, 2011] CVS (2011). *Concurrent Versions System Website*. `http://www.nongnu.org/cvs/` [Abgerufen 21. Dezember 2011].

[Desel, 2005] Desel, Jörg (2005). *Process Modeling Using Petri Nets*. In: Hofstede, Marlon Dumas; Wil van der Aalst; Arthur ter, Hrsg.: *Process-Aware Information Systems. Bridging People and Software through Process Technology*, S. 147–177. Wiley-Interscience, Hoboken, New Jersey.

[Dirkner, 2006] Dirkner, Ragna (2006). *Roundtrip-Engineering im PAOSE-Ansatz*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Duvigneau, 2002] Duvigneau, Michael (2002). *Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Duvigneau und Moldt, 2009] Duvigneau, Michael und D. Moldt, Hrsg. (2009). *Proceedings of the Fifth International Workshop on Modeling of Objects, Components and Agents, MOCA'09, Hamburg*, Nr. FBI-HH-B-290/09 in *Bericht*, Vogt-Kölln Str. 30, D-22527 Hamburg. Universität Hamburg, Department Informatik.

[Duvigneau et al., 2003] Duvigneau, Michael, D. Moldt und H. Rölke (2003). *Concurrent Architecture for a Multi-agent Platform*. In: *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002. Revised Papers and Invited Contributions*, Nr. 2585 in *Lecture Notes in Computer Science*, S. 59–72, Berlin Heidelberg New York.

[Duvigneau et al., 2006] Duvigneau, Michael, H. Rölke und F. Wienberg (2006). *Informal Introduction to the Feature Structure Nets Tool – A Tool for Process and Information Modeling*. In: Moldt, Daniel, Hrsg.: *Proceedings of the 13th Workshop Application and Tools for Petri Nets. AWPN'06*, Nr. FBI-HH-B-267/06 in *Report of the Department of*

Informatics, S. 85–91, Vogt-Kölln Str. 30, D-22527 Hamburg, Germany. Universität Hamburg, Department Informatik.

[Eclipse, 2011] Eclipse (2011). *Eclipse Project*. `http://www.eclipse.org/` [Abgerufen 21. Dezember 2011].

[Ellis et al., 2005] Ellis, Clarence A., P. Barthelmess, J. Chen und J. Wainer (2005). *Person-to-Person Processes: Computer-Supported Collaborative Work*. In: Hofstede, Marlon Dumas; Wil van der Aalst; Arthur ter, Hrsg.: *Process-Aware Information Systems. Bridging People and Software through Process Technology*, S. 37–60. Wiley-Interscience, Hoboken, New Jersey.

[Ellis et al., 1991] Ellis, Clarence A., S. J. Gibbs und G. Rein (1991). *Groupware: some issues and experiences*. Communications of the ACM, 34(1):39–58.

[Finck, 2007] Finck, Matthias (2007). *Usability-Engineering in der Open-Source-Softwareentwicklung - Perspektiven, Vorgehensweisen und Techniken*. Dissertation, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[FIPA, 2005] FIPA (2005). *FIPA Spezifikationen nach Kategorien*. `http://www.fipa.org/repository/bysubject.html` [Abgerufen 21. Dezember 2011].

[FIPA, 2011a] FIPA (2011a). *The FIPA Agent UML (AUML) Web Site*. `http://www.auml.org` [Abgerufen 21. Dezember 2011].

[FIPA, 2011b] FIPA (2011b). *Foundation for Intelligent Physical Agents*. `http://www.fipa.org` [Abgerufen 21. Dezember 2011].

[Gamma et al., 1995] Gamma, Erich, R. Helm, R. Johnson und J. Vlissides (1995). *Design Patterns*. Addison-Wesley Professional.

[German, 2003] German, Daniel (2003). *The GNOME project: a case study of open source, global software development.*. Software Process: Improvement and Practice, 8(4):201–215.

[Ghezzi et al., 2002] Ghezzi, Carlo, M. Jazayeri und D. Mandrioli (2002). *Fundamentals of Software Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[Gibbins et al., 2004] Gibbins, Nicholas, S. Harris und N. Shadbolt (2004). *Agent-based semantic web services*. Web Semantics: Science, Services and Agents on the World Wide Web, 1(2):141–154.

[Git, 2011] Git (2011). *Git Website*. `http://git-scm.com` [Abgerufen 21. Dezember 2011].

[Gross und Koch, 2007] Gross, Tom und M. Koch (2007). *Computer-Supported Cooperative Work (Interaktive Medien)*. Oldenbourg Wissenschaftsverlag, München.

[Gross und Pekkola, 2008] Gross, Tom und S. Pekkola (2008). *Analysing a workflow management system: three levels of failure*. In: *Proceedings of the 2nd ACM Symposium on Computer Human Interaction for Management of Information Technology*, S. 13. ACM.

[Gryczan, 1996] Gryczan, Guido (1996). *Prozessmuster zur Unterstützung kooperativer Tätigkeit*. Deutscher Universitäts-Verlag, Wiesbaden.

[Gudgin et al., 2007] Gudgin, Martin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar und Y. Lafon (2007). *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Technischer Bericht, W3C, XML Protocol Working Group.

[Gumm, 2009] Gumm, Dorina (2009). *A Model of Distributed Requirements Engineering: Understanding Interdependencies*. Sierke Verlag, Göttingen.

[Hass et al., 2007] Hass, Berthold, G. Walsh und T. Kilian, Hrsg. (2007). *Web 2.0: Neue Perspektiven für Marketing und Medien*. Springer-Verlag, Berlin.

[van Hee et al., 2010] Hee, Kees van, J. Hidders, G.-J. Houben, J. Paredaens und P. Thiran (2010). *On-the-Fly Auditing of Business Processes*. Transactions on Petri Nets and Other Models of Concurrency, 4:144–173.

[Hendler, 2001] Hendler, Jim (2001). *Agents and the semantic web*. IEEE Intelligent systems, 16(2):30–37.

[Hollingsworth et al., 2004] Hollingsworth, David et al. (2004). *The Workflow Reference Model: 10 Years On*. In: Fischer, Layna, Hrsg.: *Workflow Handbook 2004*, S. 295–312, Lighthouse Point, FL, USA. Future Strategies Inc.,.

[Houy et al., 2011] Houy, Constantin, P. Fettke, P. Loos, W. M. P. van der Aalst und J. Krogstie (2011). *Geschäftsprozessmanagement im Großen*. Wirtschaftsinformatik, 53(6):377–381.

[Hudson, 2011] Hudson (2011). *Hudson Continous Integration*. `http://hudson-ci.org` [Abgerufen 21. Dezember 2011].

[Huhns und Stephens, 2000] Huhns, Michael N. und L. M. Stephens (2000). *Multiagent Systems and Societies of Agents*. In: [Weiss, 2000], Kap. 2, S. 79–120.

[IBM, 2011] IBM (2011). *IBM Corporation Software Group - The business value of open collaboration*. `http://public.dhe.ibm.com/common/ssi/ecm/en/raw14207usen/RAW14207USEN.PDF` [Abgerufen 21. Dezember 2011].

[IBM, 2011] IBM (2011). *IBM Rational Unified Process (RUP)*. `http://www-01.ibm.com/software/awdtools/rup/` [Abgerufen 21. Dezember 2011].

[Jack, 2011] Jack (2011). *JACK Autonomous Software*. `http://www.agent-software.com.au/products/jack/` [Abgerufen 21. Dezember 2011].

[Jacob, 2002a] Jacob, Thomas (2002a). *Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Jacob, 2002b] Jacob, Thomas (2002b). *Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

[Jacob et al., 2002] Jacob, Thomas, O. Kummer, D. Moldt und U. Ultes-Nitsche (2002). *Implementation of Workflow Systems using Reference Nets – Security and Operability Aspects*. In: Jensen, Kurt, Hrsg.: *Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, Ny Munkegade, Bldg. 540, DK-8000 Aarhus C, Denmark. University of Aarhus, Department of Computer Science. DAIMI PB: Aarhus, Denmark, August 28–30, number 560.

[Java, 2011] Java (2011). *Java Website*. `http://www.oracle.com/technetwork/java/index.html` [Abgerufen 21. Dezember 2011].

[Jazz, 2011a] Jazz (2011a). *IBM Rational Jazz Website*. `http://www-01.ibm.com/software/rational/jazz/` [Abgerufen 21. Dezember 2011].

[Jazz, 2011b] Jazz (2011b). *Jazz Community Website*. `http://www.jazz.net` [Abgerufen 21. Dezember 2011].

[Jennings et al., 1998] Jennings, Nick, K. Sycara und M. Wooldridge (1998). *A roadmap of agent research and development*. In [Weiss, 2000], S. 7–38.

[Jensen, 1986] Jensen, Kurt (1986). *Coloured Petri Nets*. In: Brauer, Wilfried, W. Reisig und G. Rozenberg, Hrsg.: *Advances in Petri Nets*, Bd. 254 d. Reihe *Lecture Notes in Computer Science*, S. 248–299. Springer.

[Jira, 2011a] Jira (2011a). *Jira Website*. `http://www.atlassian.com/software/jira` [Abgerufen 21. Dezember 2011].

[Jira, 2011b] Jira (2011b). *PhpBB Website*. `http://www.phpbb.com` [Abgerufen 21. Dezember 2011].

[JUnit, 2011] JUnit (2011). *JUnit Homepage*. `http://www.junit.org` [Abgerufen 21. Dezember 2011].

[Keller et al., 1992] Keller, Gerhard, M. Nüttgens und A. Scheer (1992). *Semantische Prozessmodellierung auf der Grundlage „Ereignisgesteuerter Prozessketten (EPK)“*. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Saarbrücken, Heft 89.

[Kiel und Eng, 2003] Kiel, Lori und P. Eng (2003). *Experiences in Distributed Development: A Case Study*. In: *International Workshop on Global Software Development, International Conference on Software Engineering*, S. 44–47.

[Klenski und Willner, 2007] Klenski, Maxim und A. Willner (2007). *Graphische Informationsmodellierung für Mulan-Agenten*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Klussmann, 2000] Klussmann, Niels (2000). *Lexikon der Kommunikations- und Informationstechnik : Telekommunikation, Datenkommunikation, Multimedia, Internet*. Hüthig, Heidelberg, 2. erw. und aktualisierte Aufl.

[Köhler-Bussmeier, 2009] Köhler-Bussmeier, Michael (2009). *Hornets: Nets within Nets combined with Net Algebra*. In: Wolf, Karsten und G. Franceschinis, Hrsg.: *International Conference on Application and Theory of Petri Nets (ICATPN'2009)*, Bd. 5606 d. Reihe *Lecture Notes in Computer Science*, S. 243–262. Springer-Verlag.

[Köhler et al., 2001] Köhler, Michael, D. Moldt und H. Rölke (2001). *Modelling the Structure and Behaviour of Petri Net Agents*. In: Colom, José Manuel und M. Koutny, Hrsg.: *ICATPN*, Bd. 2075 d. Reihe *Lecture Notes in Computer Science*, S. 224–241. Springer-Verlag.

[Krasner und Pope, 1988] Krasner, Glenn und S. Pope (1988). *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*. Journal of Object-oriented programming, 1(3):49.

[Kreplin, 1998] Kreplin, Klaus-Dieter (1998). *Konkordanz englischer und deutscher Begriffe des Workflow Management*. `http://www.wfmc.org/Download-document/Terminology-and-Glossary-German.html` [Abgerufen 21. Dezember 2011].

[Kruchten, 2004] Kruchten, Philippe (2004). *The rational unified process: an introduction*. Addison-Wesley Professional.

[Kummer, 2002] Kummer, Olaf (2002). *Referenznetze*. Logos Verlag, Berlin.

[Kummer et al., 2009] Kummer, Olaf, F. Wienberg und M. Duvigneau (2009). *Renew – User Guide Release 2.2*. University of Hamburg, Department of Informatics, Theoretical Foundations Group, Hamburg.

[Lehmann, 2003] Lehmann, Kolja (2003). *Analyse und Bewertung von Agentenprotokollen auf Basis von Petrinetzen*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Lehmann et al., 2005] Lehmann, Kolja, L. Cabac, D. Moldt und H. Rölke (2005). *Towards a Distributed Tool Platform Based on Mobile Agents*. In: *Proceedings of the Third German Conference on Multi-Agent System Technologies (MATES)*, Bd. 3550 d. Reihe *Lecture Notes on Artificial Intelligence*, S. 179–190. Springer-Verlag.

[Lehmann und Markwardt, 2004] Lehmann, Kolja und V. Markwardt (2004). *Proposal of an Agent-based System for Distributed Software Development*. In: Moldt, Daniel, Hrsg.: *Third Workshop on Modelling of Objects, Components and Agents (MOCA 2004)*, S. 65–70, Aarhus, Denmark.

[Lehmann und Moldt, 2004] Lehmann, Kolja und D. Moldt (2004). *Modelling and Analysis of Agent Protocols with Petri Nets*. In: Lindemann, Gabriela, J. Denzinger und I. J. e. a. Timm, Hrsg.: *Multiagent System Technologies: Second German Conference, MATES 2004, Erfurt, Germany, September 29-30, 2004. Proceedings*, Bd. 3187 d. Reihe *Lecture Notes in Computer Science*, S. 85–98, Berlin, Heidelberg, New York. Springer-Verlag.

[Liske et al., 2009] Liske, Nannette, N. Lohmann, C. Stahl und K. Wolf (2009). *Another Approach to Service Instance Migration*. In: Baresi, Luciano, C.-H. Chi und J. Suzuki, Hrsg.: *ICSOC/ServiceWave*, Bd. 5900 d. Reihe *Lecture Notes in Computer Science*, S. 607–621.

[Loftus et al., 1996] Loftus, Chris, E. Sherratt, R. Gautier, P. Grandi, D. Price und M. Tedd (1996). *Distributed software engineering*. Prentice Hall, London.

[Malone und Crowston, 1993] Malone, Thomas W. und K. Crowston (1993). *The Interdisciplinary Study of Coordination*. Working Paper Series 157, MIT Center for Coordination Science.

[Markwardt et al., 2009a] Markwardt, Kolja, L. Cabac und C. Reese (2009a). *A Process-Oriented Tool-Platform for Distributed Development*. In: [Moldt et al., 2009], S. 44–52.

[Markwardt und Moldt, 2010] Markwardt, Kolja und D. Moldt (2010). *Helper Agents as a Means of Structuring Multi-Agent Applications*. In: Schwarick, Martin und M. Heiner, Hrsg.: *Proceedings of the 17th German Workshop on Algorithms and Tools for Petri Nets (AWPN 2010), Cottbus, Germany, October 7-8, 2010*, Bd. 643 d. Reihe *CEUR Workshop Proceedings*, S. 100–105. CEUR-WS.org.

[Markwardt et al., 2006] Markwardt, Kolja, D. Moldt, S. Offermann und C. Reese (2006). *Using Multi-Agent Systems for Change Management Processes in the Context of Distributed Software Development Processes*. In: Sadiq, Shazia, M. Reichert und K. Schulz, Hrsg.: *The 1st International Workshop on Technologies for Collaborative Business Process Management (TCoB 2006)*, S. 56–66.

[Markwardt et al., 2008a] Markwardt, Kolja, D. Moldt und J. Ortmann (2008a). *Proposal for Editing Workflows in a Distributed Software Development Environment*. In: Lohmann, Niels und K. Wolf, Hrsg.: *AWPN*, Bd. 380 d. Reihe *CEUR Workshop Proceedings*, S. 31–36. CEUR-WS.org.

[Markwardt et al., 2008b] Markwardt, Kolja, D. Moldt und C. Reese (2008b). *Support of Distributed Software Development by an Agent-based Process Infrastructure*. In: *MSVVEIS 2008*.

[Markwardt et al., 2009b] Markwardt, Kolja, D. Moldt und T. Wagner (2009b). *Net Agents for Activity Handling in a WFMS*. In: Freytag, Thomas und A. Eckleder, Hrsg.: *16th German Workshop on Algorithms and Tools for Petri Nets, AWPN 2009, Karlsruhe, Germany, September 25, 2009, Proceedings*, CEUR Workshop Proceedings.

[McAfee, 2006] McAfee, Andrew (2006). *Enterprise 2.0: The Dawn of Emergent Collaboration*. MIT Sloan Management Review, 47(3):21–28.

[McAffer und Lemieux, 2005] McAffer, Jeff und J.-M. Lemieux (2005). *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional.

[Microsoft, 2011] Microsoft (2011). *Visual Studio Team System 2008*. `http://msdn.microsoft.com/de-de/library/fda2bad5(v=VS.90).aspx` [Abgerufen 21. Dezember 2011].

[Mitreiter, 2008] Mitreiter, Klaus (2008). *Einbetten der grafischen Benutzungsschnittstelle von Renew in Eclipse*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Moldt, 2005] Moldt, Daniel (2005). *Petrinetze als Denkzeug*. In: Farwer, Berndt und D. Moldt, Hrsg.: *Object Petri Nets, Processes, and Object Calculi*, Nr. FBI-HH-B-265/05 in *Bericht des Fachbereichs Informatik*, S. 51–70, Vogt-Kölln Str. 30, D-22527 Hamburg. Universität Hamburg, Fachbereich Informatik.

[Moldt, 2006] Moldt, Daniel (2006). *PAOSE: A Way to Develop Distributed Software Systems Based on Petri Nets and Agents*. In: Barjis, Joseph, U. Ultes-Nitsche und J. C. Augusto, Hrsg.: *Proceedings of The Fourth International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS'06), May 23-24, 2006 – Paphos, Cyprus 2006*, S. 1–2.

[Moldt et al., 2007] Moldt, Daniel, F. Kordon, K. van Hee, J.-M. Colom und R. Bastide, Hrsg. (2007). *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE'07)*, Siedlce, Poland. Akademia Podlaska.

[Moldt und Ortmann, 2004] Moldt, Daniel und J. Ortmann (2004). *A Conceptual and Practical Framework for Web-Based Processes in Multi-Agent Systems*. In: Sonenberg, Liz und C. Sierra, Hrsg.: *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*, S. 1464–1465. Computer Society.

[Moldt et al., 2010] Moldt, Daniel, J. Quenum, C. Reese und T. Wagner (2010). *Improving a Workflow Management System with an Agent Flavour*. In: Duvigneau, Michael und D. Moldt, Hrsg.: *Proceedings of the International Workshop on Petri Nets and Software Engineering, PNSE'10, Braga, Portugal*, Nr. FBI-HH-B-294/10 in *Bericht*, S. 55–70, Vogt-Kölln Str. 30, D-22527 Hamburg. Universität Hamburg, Department Informatik.

[Moldt und Rölke, 2003] Moldt, Daniel und H. Rölke (2003). *Pattern Based Workflow Design Using Reference Nets*. In: Aalst, Wil van der, A. t. Hofstede und M. Weske, Hrsg.: *Proceedings of International Conference on Business Process Management, Eindhoven, NL*, Bd. 2678 d. Reihe *Lecture Notes in Computer Science*, S. 246–260. Springer-Verlag.

[Moldt et al., 2009] Moldt, Daniel, U. Ultes-Nitsche und J. C. Augusto, Hrsg. (2009). *Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems – MSVVEIS 2009, In conjunction with ICEIS 2009, Milan, Italy, May 2009*, Portugal. INSTICC PRESS.

[Müller, 2001] Müller, Klaus (2001). *Zustandsautomaten zur Vorgangssteuerung in WAM-Anwendungen*. Diplomarbeit, Universität Hamburg.

[Müller-Lindenberg, 2005] Müller-Lindenberg, Matthias (2005). *Führung in zeitkritischen und komplexen Projekten*. Gabler Edition Wissenschaft. Deutscher Universitäts-Verlag, Wiesbaden.

[Nagi, 2001] Nagi, Khaled (2001). *Transactional Agents - Towards a Robust Multi-Agent System*, Bd. 2249 d. Reihe *Lecture Notes in Computer Science*. Springer-Verlag, Berlin Heidelberg New York.

[Nourie, 2005] Nourie, Dana (2005). *Getting started with an integrated development environment (IDE)*. `http://java.sun.com/developer/technicalArticles/tools/intro.html` [Abgerufen 21. Dezember 2011].

[OASIS, 2005] OASIS (2005). *Specification TC*. UDDI Version 3 Specification.

[OASIS, 2011] OASIS (2011). *Web Services Business Process Execution Language Version 2.0*. `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html` [Abgerufen 21. Dezember 2011].

[Oberquelle, 1987] Oberquelle, Horst (1987). *Sprachkonzepte für benutzergerechte Systeme*, Bd. 144 d. Reihe *Informatik-Fachberichte*. Springer-Verlag.

[Oberweis, 1996] Oberweis, Andreas (1996). *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Teubner-Reihe Wirtschaftsinformatik. Teubner-Verlag, Wiesbaden.

[Oberweis, 2005] Oberweis, Andreas (2005). *Person-to-Application Processes: Workflow Management*. In: Hofstede, Marlon Dumas; Wil van der Aalst; Arthur ter, Hrsg.: *Process-Aware Information Systems. Bridging People and Software through Process Technology*, S. 21–36. John Wiley and Sons.

[OMG, 2011] OMG (2011). *Object Management Group/Business Process Management Initiative*. `http://www.bpmn.org` [Abgerufen 21. Dezember 2011].

[Omicini et al., 2008] Omicini, Andrea, A. Ricci und M. Viroli (2008). *Artifacts in the A&A meta-model for multi-agent systems*. Autonomous Agents and Multi-Agent Systems, 17(3):432–456.

[OSGi, 2011] OSGi (2011). *OSGi Alliance Website*. `http://www.osgi.org/Main/HomePage` [Abgerufen 21. Dezember 2011].

[OSLC, 2011] OSLC (2011). *Open Services for Lifecycle Collaboration*. `http://open-services.net` [Abgerufen 21. Dezember 2011].

[Padgham und Winikoff, 2002] Padgham, Lin und M. Winikoff (2002). *Prometheus: A Methodology for Developing Intelligent Agents*. In: Giunchiglia, Fausto, J. Odell und G. Weiss, Hrsg.: *AOSE*, Bd. 2585 d. Reihe *Lecture Notes in Computer Science*, S. 174–185. Springer.

[Pankoke-Babatz, 2003] Pankoke-Babatz, Uta (2003). *Designkonzept für Systeme zur computergestützten Zusammenarbeit unter Nutzung der Behavior-Setting-Theorie*. Shaker Verlag, Aachen.

[PAOSE, 2011] PAOSE (2011). *Paose Website*. `https://paose.informatik.uni-hamburg.de/paose/` [Abgerufen 21. Dezember 2011].

[Pokahr et al., 2003] Pokahr, Alexander, L. Braubach und W. Lamersdorf (2003). *Jadex: Implementing a bdi-infrastructure for jade agents*. EXP-in search of innovation (Special Issue on JADE), 3(3):76–85.

[Protégé, 2011] Protégé (2011). *The Protégé Ontology Editor and Knowledge Acquisition System*. `http://protege.stanford.edu` [Abgerufen 21. Dezember 2011].

[Rational, 2011] Rational (2011). *Rational Team Concert*. `http://www-01.ibm.com/software/awdtools/rtc/` [Abgerufen 21. Dezember 2011].

[RCP, 2011] RCP (2011). *Rich Client Platform Website*. `http://www.eclipse.org/home/categories/rcp.php` [Abgerufen 21. Dezember 2011].

[Reese, 2010] Reese, Christine (2010). *Prozess-Infrastruktur für Agentenanwendungen*, Bd. 3 d. Reihe *Agent Technology – Theory and Applications*. Logos Verlag, Berlin. Dissertation. Pdf: `http://www.sub.uni-hamburg.de/opus/volltexte/2010/4497/`.

[Reese et al., 2006a] Reese, Christine, K. Markwardt, S. Offermann und D. Moldt (2006a). *Distributed Business Processes in Open Agent Environments*. In: Manolopoulos, Yannis, J. Filipe, P. Constantopoulos und J. Cordeiro, Hrsg.: *International Conference on Electronic Information Systems (ICEIS) 2006*, S. 81–86.

[Reese et al., 2006b] Reese, Christine, S. Offermann und D. Moldt (2006b). *Architektur für verteilte, agentenbasierte Workflows*. In: Schoop, Mareike, C. Huemer, M. Rebstock und M. Bichler, Hrsg.: *Service-oriented Electronic Commerce im Rahmen der Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006)*, Bd. P-80 d. Reihe *Lecture Notes in Informatics (LNI) - Proceedings*, S. 73–87, Bonn. Gesellschaft für Informatik, Köllen Druck+Verlag GmbH.

[Reese et al., 2005] Reese, Christine, J. Ortmann, D. Moldt, S. Offermann, K. Lehmann und T. Carl (2005). *Architecture for Distributed Agent-Based Workflows*. In: Henderson-Sellers, Brian und M. Winikoff, Hrsg.: *Proceedings of the Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005), Utrecht, Niederlande, as part of AAMAS 2005 (Autonomous Agents and Multi Agent Systems), July 2005*, S. 42–49.

[Reese et al., 2006c] Reese, Christine, J. Ortmann, D. Moldt, S. Offermann, K. Lehmann und T. Carl (2006c). *Fragmented Workflows Supported by an Agent Based Architecture*. In: Kolp, Manuel, P. Bresciani, B. Henderson-Sellers und M. Winikoff, Hrsg.: *Agent-Oriented Information Systems III 7th International Bi-Conference Workshop, AOIS 2005, Utrecht, Netherlands, July 26, 2005, and Klagenfurt, Austria, October 27, 2005, Revised Selected Papers*, Bd. 3529 d. Reihe *Lecture Notes in Computer Science*, S. 200–215. Springer-Verlag.

[Reichert und Dadam, 1998] Reichert, Manfred und P. Dadam (1998). *ADEPT$_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control*. Journal of Intelligent Information Systems, 10(2):93–129.

[Reichert et al., 2003] Reichert, Manfred, P. Dadam und T. Bauer (2003). *Dealing With Forward and Backward Jumps in Workflow Management Systems*. International Journal of Software and Systems Modeling (SoSyM), 2(1):37–58.

[Reichert und Dadam, 1997] Reichert, Manfred und P. Dadam (1997). *A Framework for Dynamic Changes in Workflow Management Systems*. In: *Proceedings of the 8th International Workshop on Database and Expert Systems Applications*, DEXA '97, S. 42–48, Washington, DC, USA. IEEE Computer Society.

[Renew, 2007] Renew (2007). Renew – *The Reference Net Workshop homepage*. `http://www.renew.de/`.

[RFC4120, 2011] RFC4120 (2011). *RFC4120: The Kerberos Network Authentication Service (V5)*. `http://tools.ietf.org/html/rfc4120` [Abgerufen 21. Dezember 2011].

[RFC4511, 2011] RFC4511 (2011). *RFC4511: Lightweight Directory Access Protocol (LDAP): The Protocol*. `http://tools.ietf.org/html/rfc4511` [Abgerufen 21. Dezember 2011].

[RMI, 2011] RMI (2011). *Remote Method Invocation Home*. `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html` [Abgerufen 21. Dezember 2011].

[Rölke, 1999] Rölke, Heiko (1999). *Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen*. Diplomarbeit, Universität Hamburg.

[Rölke, 2004] Rölke, Heiko (2004). *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, Bd. 2. Logos Verlag, Berlin.

[Roock und Wolf, 1998] Roock, Stefan und H. Wolf (1998). *Die Raummetapher zur Entwicklung kooperationsunterstützender Softwaresysteme für Organisationen*. Diplomarbeit, Universität Hamburg.

[Royce, 1970] Royce, Winston (1970). *Managing the development of large software systems*. In: *proceedings of IEEE WESCON*, Bd. 26. Los Angeles.

[Russell et al., 2009] Russell, Nick, W. van der Aalst und A. ter Hofstede (2009). *Designing a workflow system using coloured Petri nets*. Transactions on Petri Nets and Other Models of Concurrency III, S. 1–24.

[Ryu et al., 2008] Ryu, Seung Hwan, F. Casati, H. Skogsrud, B. Benatallah und R. Saint-Paul (2008). *Supporting the dynamic evolution of web service protocols in service-oriented architectures*. ACM Transactions on the Web (TWEB), 2(2):1–46.

[Sauer, 2010] Sauer, Joachim (2010). *Architekturzentrierte agile Anwendungsentwicklung in global verteilten Projekten*. Doktorarbeit, Universität Hamburg. Pdf: `http://ediss.sub.uni-hamburg.de/volltexte/2011/4959/` [Abgerufen 21. Dezember 2011].

[Schleinzer, 2007] Schleinzer, Benjamin (2007). *Flexible und hierarchische Multiagentensysteme – Modellierung und prototypische Erweiterung von Mulan und Capa*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Scrum Alliance, 2011] Scrum Alliance (2011). *The Scrum Alliance Homepage*. `http://www.scrumalliance.org` [Abgerufen 21. Dezember 2011].

[Selenium, 2011] Selenium (2011). *Selenium Homepage*. `http://seleniumhq.org` [Abgerufen 21. Dezember 2011].

[Sommerville, 2007] Sommerville, Ian (2007). *Software engineering*. Addison-Wesley New York, 8. Aufl.

[Starke, 1990] Starke, Peter H. (1990). *Analyse von Petri-Netz-Modellen*. B.G. Teubner, Stuttgart.

[Subversion, 2011] Subversion (2011). *Apache Subversion Website*. `http://subversion.apache.org` [Abgerufen 21. Dezember 2011].

[Sudeikat, 2010] Sudeikat, Jan (2010). *Engineering Self-Organizing Dynamics in Distributed Systems: a Systemic Approach*. Suedwestdeutscher Verlag fuer Hochschulschriften, Saarbrücken.

[SWT, 2011] SWT (2011). *SWT: The Standard Widget Toolkit*. `http://www.eclipse.org/swt/` [Abgerufen 21. Dezember 2011].

[Tell, 2005] Tell, Volker (2005). *Schaffung der Grundlagen für die prototypische Umsetzung eines Multiagentensystem basierten Leitmodells*. Diplomarbeit, Universität Hamburg, Fachbereich Informatik.

[Tell und Moldt, 2005] Tell, Volker und D. Moldt (2005). *Ein Petrinetzsystem zur Modellierung selbstmodifizierender Petrinetze*. In: Schmidt, Karsten und C. Stahl, Hrsg.: *Proceedings of the 12th Workshop on Algorithms and Tools for Petri Nets (AWPN 05)*, S. 36–41. Humboldt Universität zu Berlin, Fachbereich Informatik.

[Teuber, 1995] Teuber, Klaus (1995). *Die Siedler von Catan*. Kosmos-Verlag, Stuttgart. `http://www.catan.de`.

[Treinen und Miller-Frost, 2006] Treinen, James und S. L. Miller-Frost (2006). *Following the sun: case studies in global software development*. IBM Syst. J., 45(4):773–783.

[UML, 2011] UML (2011). *Unified Modeling Language^{TM}(UML®) specification overview*. `http://www.omg.org/spec/UML/` [Abgerufen 21. Dezember 2011].

[Valk, 1998] Valk, Rüdiger (1998). *Petri Nets as Token Objects - An Introduction to Elementary Object Nets*. In: Desel, Jörg und M. Silva, Hrsg.: *19th International Conference on Application and Theory of Petri nets, Lisbon, Portugal*, Nr. 1420 in *Lecture Notes in Computer Science*, S. 1–25, Berlin, Heidelberg, New York. Springer-Verlag.

[Valk, 2004] Valk, Rüdiger (2004). *Object Petri Nets – Using the Nets-within-Nets Paradigm*. In: Desel, Jörg, W. Reisig und G. Rozenberg, Hrsg.: *Advances in Petri Nets: Lectures on Concurrency and Petri Nets*, Bd. 3098 d. Reihe *Lecture Notes in Computer Science*, S. 819–848. Springer-Verlag, Berlin, Heidelberg, New York.

[Verbeek et al., 2001] Verbeek, Eric, T. Basten und W. van der Aalst (2001). *Diagnosing Workflow Processes using Woflan*. The Computer Journal, 44(4):246–279.

[Vliet, 2008] Vliet, Hans van (2008). *Software engineering: principles and practice*. John Wiley and Sons, 3. Aufl.

[Wagner, 2009a] Wagner, Thomas (2009a). *A Centralized Petri Net- and Agent-based Workflow Management System*. In: [Duvigneau und Moldt, 2009], S. 29–44.

[Wagner, 2009b] Wagner, Thomas (2009b). *Modeling of a Centralized Petri Net- and Agent-based Workflow Management System*. Baccalaureatsarbeit, Universität Hamburg, Department Informatik.

[Wagner, 2009c] Wagner, Thomas (2009c). *Prototypische Realisierung einer Integration von Agenten und Workflows*. Diplomarbeit, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg.

[Wagner et al., 2011] Wagner, Thomas, J. Quenum, D. Moldt und C. Reese (2011). *Providing an Agent Flavored Integration for Workflow Management*. LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), LNCS 6900. To be published.

[Weiss, 2000] Weiss, Gerhard, Hrsg. (2000). *Multiagent systems: a modern approach to distributed artificial intelligence*. The MIT Press, Cambridge, MA, USA, 2. Aufl.

[Werner und Fischer, 2007] Werner, Christian und S. Fischer (2007). *Architecture and standardisation of web services*. In: Studer, Rudi, S. Grimm und A. Abecker, Hrsg.: *Semantic Web Services: Concepts, Technologies, and Applications*, Kap. 2, S. 25–48. Springer-Verlag, Berlin Heidelberg.

[Wester-Ebbinghaus, 2010] Wester-Ebbinghaus, Matthias (2010). *Von Multiagentensystemen zu Multiorganisationssystemen – Modellierung auf Basis von Petrinetzen*. Dissertation, Universität Hamburg, Department Informatik, Vogt-Kölln Str. 30, D-22527 Hamburg. `http://www.sub.uni-hamburg.de/opus/volltexte/2011/4974/`.

[Wester-Ebbinghaus et al., 2007] Wester-Ebbinghaus, Matthias, D. Moldt, C. Reese und K. Markwardt (2007). *Towards Organization–Oriented Software Engineering*. In: Züllighoven, Heinz, Hrsg.: *Software Engineering Konferenz 2007 in Hamburg: SE'07 Proceedings*, Bd. 105 d. Reihe *LNI*, S. 205–217. GI.

[WfMC, 1995] WfMC (1995). *WfMC Workflow Reference Model*. `http://www.wfmc.org/standards/docs/tc003v11.pdf` [Abgerufen 21. Dezember 2011].

[WfMC, 1999] WfMC (1999). *WfMC Terminology & Glossary*. `http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf` [Abgerufen 21. Dezember 2011].

[WfMC, 2011a] WfMC (2011a). *Workflow Management Coalition Homepage*. `http://www.wfmc.org` [Abgerufen 21. Dezember 2011].

[WfMC, 2011b] WfMC (2011b). *XPDL Support and Resources*. `http://www.wfmc.org/xpdl.html` [Abgerufen 21. Dezember 2011].

[Wieland et al., 2009] Wieland, Matthias, C. Längerer, F. Leymann, O. Siemoneit und C. Hubig (2009). *Methods for Conserving Privacy in Workflow Controlled Smart Environments*. In: *2009 Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, S. 16–21. IEEE.

[Willmott et al., 2005] Willmott, Steven, M. Beer, R. Hill, D. Greenwood, M. Calisti, I. Mathieson, L. Padgham, C. Reese, K. Lehmann, T. Scholz und M. O. Shafiq (2005). *NETDEMO: openNet Networked Agents Demonstration*. In: Pechoucek, Michael, D. Steiner und S. Thompson, Hrsg.: *AAMAS 2005. Proceedings (Industry Track)*, S. 129–130. 2 individual demos: (1) CAPA: The CAPA Mobile Chat Agent & Web Services Gateway Agent and (2) Settler: AgentBased Settler Game.

[Wooldridge und Jennings, 1995] Wooldridge, M. J. und N. R. Jennings (1995). *Intelligent Agents: Theory and Practice*. The Knowledge Engineering Review, 2(10):115–152.

[Wooldridge, 2000] Wooldridge, Michael (2000). *Intelligent Agents*. In: [Weiss, 2000], Kap. 1, S. 27–77.

[Wooldridge et al., 2000] Wooldridge, Michael, N. R. Jennings und D. Kinny (2000). *The Gaia Methodology for Agent-Oriented Analysis and Design*. Journal of Autonomous Agents and Multi-Agent Systems, 3(3):285–312.

[Wulf, 1995] Wulf, Martina (1995). *Konzeption und Realisierung einer Umgebung zur Koordination rechnergestützter Tätigkeiten in kooperativen Arbeitsprozessen*. Diplomarbeit, Universität Hamburg. `http://swt-www.informatik.uni-hamburg.de/uploads/media/Martina.Wulf.Diplomarbeit.ps.gz`.

[Zambonelli et al., 2003] Zambonelli, Franco, N. R. Jennings und M. Wooldridge (2003). *Developing multiagent systems: The Gaia methodology*. ACM Trans. Softw. Eng. Methodol., 12(3):317–370.

[Züllighoven, 2004] Züllighoven, Heinz (2004). *Object-Oriented Construction Handbook*. dpunkt Verlag, Heidelberg.

Index