Realzeit-Szeneninterpretation mit ontologiebasierten Regeln

Dissertation

zur Erlangung des akademischen Grades

Dr. rer. nat.

an der Fakultät für Mathematik, Informatik und Naturwissenschaften

der Universität Hamburg

eingereicht beim Fach-Promotionsausschuss Informatik von

Wilfried Bohlken

aus Jever (Deutschland)

Juni 2012

Gutachterinnen/Gutachter

Prof. Wolfgang Menzel

Prof. Bernd Neumann

Tag der Disputation: 21.12.2012

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als Doktorand und wissenschaftlicher Mitarbeiter am Arbeitsbereich Kognitive Systeme der Universität Hamburg.

Zuerst möchte ich mich bei Prof. Bernd Neumann für die Betreuung der Arbeit, interessante Diskussionen und Anregungen und die gute Zusammenarbeit bedanken. Außerdem gilt mein Dank meiner weiteren Betreuerin Prof. Leonie Dreschler-Fischer und meinem weiteren Betreuer Prof. Wolfgang Menzel.

Bedanken möchte ich mich auch bei Patrick Koopmann, Roberto Fraile und Lothar Hotz, für die gute Zusammenarbeit, anregende Diskussionen und eine unvergessene nächtliche Programmiersession in einem Hotelzimmer in Toulouse. Weiterhin bedanke ich mich bei Michaela Zimmer, Benjamin Seppke, Rainer Herzog, Johannes Hartz, Christian Bähnisch, Kasim Terzić, Peer Stelldinger, Leonid Tcherniavski und Arne Kreutzmann.

Schließlich möchte ich mich bei allen Mitarbeiterinnen und Mitarbeitern des Arbeitsbereichs Kognitive Systeme und meinen Eltern bedanken.

Zusammenfassung

Diese Arbeit beschäftigt sich mit dem Thema „Szeneninterpretation“, deren Ziel das automatische „Verstehen“ von Bildern und Videosequenzen ist, d.h. die Zuweisung von Bedeutung, oberhalb der Ebene einzeln erkannter Objekte, die menschlichen Konzeptbeschreibungen entspricht. Es wird ein generisches Rahmenwerk für die wissensbasierte Szeneninterpretation präsentiert, realisiert durch das Szeneninterpretationssystem SCENIOR (SCENe Interpretation with Ontology-based Rules). Aktionsmodelle werden mit Hilfe einer OWL (Web Ontology Language) Ontologie definiert, erweitert durch SWRL-Regeln, zum Beschreiben von Constraints. Diese konzeptuelle Wissensbasis wird automatisch in eine regelbasierte Wissensbasis für SCENIOR übersetzt, welche auf Jess-Regeln basiert. Interpretationsziele werden in hierarchisch strukturierte Hypothesen transformiert, die mit Constraints verknüpft sind. Der inkrementelle Interpretationsprozess ist als Strahlsuche mit multiplen parallelen Interpretations-Threads organisiert. Bei jedem Schritt wird eine kontextabhängige probabilistische Bewertung für jede alternative partielle Interpretation berechnet. Niedrig bewertete Threads werden aussortiert, abhängig von der Strahlbreite. Vollständig instanziierte Hypothesen dienen als Eingabe für Hypothesen, die höher in der Hierarchie angeordnet sind. Fehlende Evidenz kann, abhängig vom Kontext, „halluziniert“ werden. Das Interpretationssystem wurde anhand von Experimenten für die Domäne der Flughafenvorfeldaktivitäten und die Domäne intelligenter Wohnumgebungen evaluiert.

Abstract

This thesis is concerned with the subject of “high-level scene interpretation”, which can be roughly defined as understanding images or video streams at abstraction levels above single objects. A generic framework for knowledge-based scene interpretation is presented, realised with an interpretation system called SCENIOR (SCENe Interpretation with Ontology-based Rules). Activity models are defined in an ontology using OWL (Web Ontology Language), extended by SWRL rules for describing constraints. This conceptual knowledge base is transformed into a high-level scene interpretation system based on Jess rules. Interpretation goals are transformed into hierarchical hypotheses structures associated with constraints. The incremental interpretation process is organised as a Beam Search with multiple parallel interpretation threads. At each step, a context-

dependent probabilistic rating is computed for each partial interpretation. Low-rated threads are discarded depending on the beam width. Fully instantiated hypotheses may be used as input for higher-level hypotheses. Missing evidence may be "hallucinated" depending on the context. The system has been evaluated for the domain of aircraft service activities and for smart home environments.

Publikationen

[1] Bohlken, W.; Neumann, B.: Generation of Rules from Ontologies for High-level Scene Interpretation. In: G. Governatori et al. (Hrsg.): Rule Interchange and Applications, Proc. International Symposium RuleML 2009, Springer, LNCS 5858, 2009, S. 93 - 107.

[2] Bohlken, W.; Neumann, B.; Hotz, L.; Koopmann, P.: Ontology-Based Realtime Activity Monitoring Using Beam Search. In: Crowley, J.L. et al. (Hrsg.): ICVS 2011, Springer, Heidelberg, LNCS 6962, 2011, S. 112 - 121.

[3] Bohlken, W.; Koopmann, P.; Neumann, B.: SCENIOR: Ontology-based Interpretation of Aircraft Service Activities. Technical Report FBI-HH-B - 297/11, Department of Informatics, University of Hamburg, 2011.

[4] Bohlken W.; Koopmann, P.; Hotz, L.; Neumann, B.: Towards Ontology-based Realtime Behaviour Interpretation. Human Behaviour Recognition Technologies: Intelligent Applications for Monitoring and Security, IGI Global. Voraussichtliche Veröffentlichung: 2013.

Inhaltsverzeichnis

1 Einleitung

Die Arbeit beginnt mit einer Motivation des Themas (Abschnitt 1.1), einer Beschreibung des Anwendungsgebiets (Abschnitt 1.2) und einer Vorstellung der Ziele und Anforderungen (Abschnitt 1.3). Am Ende des ersten Kapitels (Abschnitt 1.4) folgt eine Übersicht über die Gliederung der Arbeit.

1.1 Motivation

Die Szeneninterpretation befasst sich mit dem maschinellen Verstehen von Bildern und Videosequenzen auf hohem Abstraktionsniveau und berührt damit verschiedene Teilgebiete der *Künstlichen Intelligenz* (KI), die wiederum einen Zweig der Informatik darstellt. Zentrale Teilgebiete sind Wissensbasierte Systeme, Spracherkennung, Bildverarbeitung und Mustererkennung, Robotik, Planung, maschinelles Lernen und automatisches Beweisen. Auch wenn sich eine präzise Definition des Begriffs „künstliche Intelligenz“ als schwierig erweist, da es bereits an einer genauen Definition des Begriffs „Intelligenz“ mangelt, so kann die künstliche Intelligenz als Wissenschaft aufgefasst werden, die sich mit der *Automatisierung intelligenten Verhaltens* beschäftigt.

Längst haben KI-basierte Verfahren und Methoden in Anwendungen der Industrie oder des täglichen Lebens Einzug gehalten, z.B. bei Expertensystemen, Suchmaschinen, Spracherkennung zum Diktieren von Texten, maschinellem Übersetzen von Texten, Handschriftenerkennung bei PDAs, Schachcomputern, Computerspielen, Navigationssystemen, Fahrerassistenzsystemen, Computeralgebrasystemen (Mathematica, Maple) und Computer-Vision-Systemen zum Überwachen von sicherheitsrelevanten Plätzen oder Produktionsprozessen.

Bei der Szeneninterpretation werden Methoden der Bildverarbeitung, wissensbasierter Systeme und häufig auch des maschinellen Lernens miteinander verknüpft. Ziel der Szeneninterpretation ist das automatische „Verstehen“ von Bildern und Videosequenzen, d.h. die Zuweisung von Bedeutung, oberhalb der Ebene einzeln erkannter Objekte, die menschlichen Konzeptbeschreibungen entspricht. Typi-

scherweise gilt es dabei, zeitliche und räumliche Constraints zu berücksichtigen. Bei statischen Bildern sind das Beschreibungen, die komplexe Einheiten oder das Bild als Ganzes definieren – z.B. eine Häuserfassade oder einen Balkon – während es in Videosequenzen Definitionen ganzer Handlungsabläufe sind – z.B. eine Person, die einen Banküberfall verübt oder ein Flugzeug, das betankt wird. Die Modelle, auf die sich derartige Interpretationen stützen, werden in einer *konzeptuellen Wissensbasis* definiert.

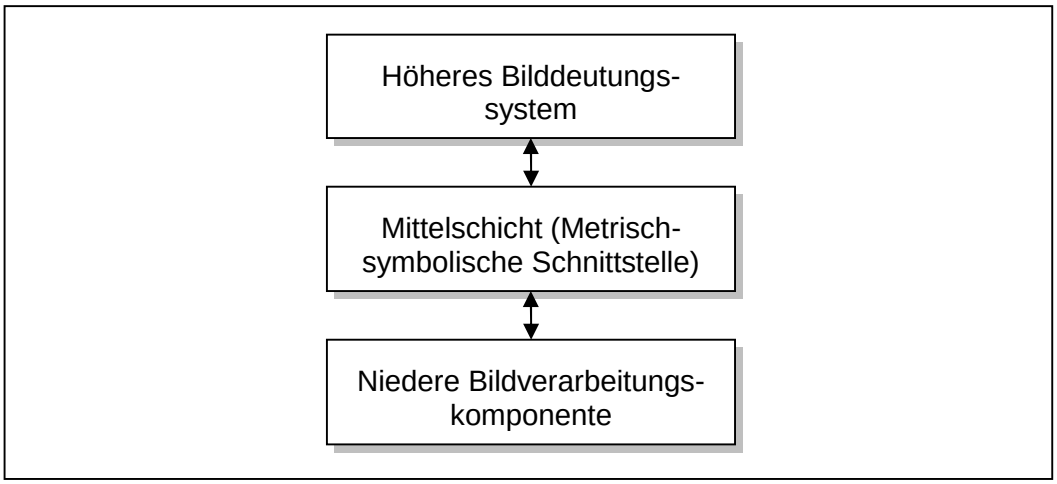


Abbildung 1: Allgemeiner Aufbau eines Szeneninterpretationssystems.

Der allgemeine Aufbau eines Szeneninterpretationssystems besteht aus drei Schichten: einer niederen Bildverarbeitungskomponente, einer metrisch-symbolischen Schnittstelle (Mittelschicht) und einem höheren Bilddeutungssystem (s. Abbildung 1**)**.

Die Bildverarbeitungskomponente hat die Aufgabe der Segmentierung, der Objekterkennung und Objektverfolgung, d.h. Berechnung von Trajektorien beweglicher Objekte, mit dem Ziel, geometrische Szenenbeschreibungen zu erstellen. Diese dienen als Eingabe für die metrisch-symbolische Schnittstelle, deren Aufgabe es ist, aus der geometrischen Szenenbeschreibung symbolische Entitäten zu generieren, die von unwichtigen, oft quantitativen Details abstrahieren, und diese primitiven Konzepten der Wissensbasis zuzuordnen. Die primitiven Konzepte – hier *Evidenzen* genannt – werden dann als Eingabedaten an das höhere Bilddeutungssystem weitergereicht, um mit Hilfe der konzeptuellen Wissensbasis komplexere Konzepte abzuleiten (s. [64]). Eine Zuordnung von primitiven Evidenzen zu Modellen dieser konzeptuellen Wissensbasis wird als *Interpretation* bezeichnet. Die Pfeile in Abbildung 1 deuten an, dass es dabei auch Rückkopplungen von höheren zu niedrigeren Schichten geben kann. Im weiteren Verlauf der Arbeit ist der Begriff „Szeneninterpretationssystem“ oftmals mit der obersten Schicht asso-

ziiert und nicht im Sinne eines vollständigen Interpretationssystems zu verstehen, welches alle drei Schichten umfasst.

Bestehende Szeneninterpretationssysteme sind oftmals mit folgenden Problemen konfrontiert:

- Mangel an Flexibilität und Anpassungsfähigkeit,
- aufwändige Verifizierung und Wartung,
- keine Echtzeitfähigkeit,
- Mangel an Robustheit.

Die ersten beiden Punkte resultieren in erster Linie aus der Tatsache, dass für die konzeptuellen Wissensbasen häufig eigenständige, nicht-standardisierte Repräsentationsformen benutzt werden. Szeneninterpretationssysteme für komplexe Anwendungsprobleme, die in dieser Arbeit betrachtet werden, erfordern fundierte Wissensrepräsentation, standardisierte Inferenzverfahren und ein generisches Rahmenwerk, das mit wenig Aufwand an unterschiedliche Domänen angepasst werden kann.

Für viele Anwendungsprobleme ist es essentiell, dass ein Szeneninterpretationssystem in der Lage ist, Eingabedaten in Realzeit zu verarbeiten und jederzeit ein optimales Interpretationsergebnis abzuliefern, basierend auf den aktuell vorliegenden Daten. Oftmals ist auch eine zeitliche Abschätzung oder Vorhersage des weiteren Verlaufs bestimmter Aktionen wünschenswert. Diese Anforderung wird besonders bei sicherheitsrelevanten Anwendungen, wie Fahrerassistenzsystemen, Überwachungsaufgaben von kriminellen Aktivitäten oder Flughafenvorfeldaktivitäten offensichtlich, welche in dieser Arbeit vorwiegend zur Veranschaulichung der entwickelten Ansätze betrachtet werden. Um diese Anforderungen zu erfüllen, sind performante Algorithmen und Parallelverarbeitung notwendig.

Die Voraussetzung für ein robustes Interpretationssystem ist die Fähigkeit, unsichere und unvollständige Eingabedaten verarbeiten zu können. Da die Evidenzen für die höhere Interpretationskomponente von niederen Bildverarbeitungskomponenten geliefert werden, ist die Szeneninterpretation mit den für die Bildverarbeitung typischen Problemen konfrontiert. Dies sind z.B. Tracking- oder Segmentierungsfehler, verursacht durch Verdeckung oder ungünstige Lichtverhältnisse. Sie führen dazu, dass die Evidenzen bei komplexen Anwendungen der realen Welt unvollständig, mit Unsicherheit behaftet und mit Rauschen durchsetzt sind. Mehrdeutigkeiten sind eine grundsätzliche Eigenschaft der Szeneninterpretation, da es oftmals mehrere Möglichkeiten gibt, eine Evidenz im Modell zuzuordnen. Dieses Problem wird durch unsichere Evidenzen und Rauschen verschärft. Robuste In-

terpretationssysteme müssen daher über reine Deduktion hinausgehen und Mechanismen zum Ergänzen fehlender Evidenzen integrieren und probabilistische Methoden einsetzen.

Die vorliegende Arbeit soll einen Beitrag zur Lösung der dargestellten Probleme liefern. Sie stellt einen neuartigen, generischer Ansatz für ein allgemeines Rahmenwerk für die Szeneninterpretation vor, welcher auf Ontologien basiert, formuliert in einer standardisierten Beschreibungssprache. Eine Ontologie wird vollautomatisch in ein performantes regelbasiertes System übersetzt, welches den Anforderungen der Echtzeitfähigkeit und Robustheit durch massive Parallelverarbeitung, Mechanismen zur Ergänzung fehlender Evidenz und Integration probabilistischer Verfahren begegnet.

1.2 Anwendungsgebiete

Die Anwendungsmöglichkeiten für Szeneninterpretationssysteme sind vielfältig, sie können z.B. eingesetzt werden für

- Überwachungsaufgaben,
- intelligente Wohnumgebungen,
- Erkennung krimineller Aktivitäten,
- Fahrerassistenzsysteme,
- Bild- und Videosuche,
- semantische Aspekte im Internet der Dinge,
- automatisches Analysieren von Sportspielen,
- Perzeptionsaufgaben in die Robotik.

1.2.1 Überwachungsaufgaben

Szeneninterpretationssysteme eignen sich für Überwachungsaufgaben, bei denen die Einhaltung korrekter Abläufe oder Verhaltensweisen gewährleistet werden soll. Das kann sowohl industrielle Produktionsabläufe beinhalten, als auch Handlungen, die von Menschen ausgeführt werden. Grundsätzlich eignen sich für die

wissensbasierte Szeneninterpretation strukturierte Abläufe, die sich hinreichend gut durch Modelle beschreiben lassen.

Der wesentliche Teil dieser Arbeit entstand während des EU Projekts *Co-Friend*[1] [17]. Ziel des Projekts war die automatische Erkennung von Flugzeug-Serviceaktivitäten, wie z.B. Ankunftsvorbereitung, Be- und Entladen des Gepäcks, Betankung, Catering, Heran- und Wegfahren der Passagierbrücke, Abfahrt des Flugzeugs. Dazu wurden am Toulouse Blagnac Flughafen feste und PTZ- (*Pan, Tilt, Zoom*) Kameras installiert, um ein dediziertes Vorfeld zu observieren. Das System sollte Warnungen bei fehlenden Aktivitäten geben, oder bei denjenigen, die nicht den Sicherheitsregularien entsprachen. Eine weitere Anforderung war das Prognostizieren des zeitlichen Verlaufs einer Abfertigung.

In ähnlicher Weise könnten Szeneninterpretationssysteme für die automatische Verkehrsüberwachung eingesetzt werden. Sie dient zur Lenkung des fließenden Verkehrs und zur Durchsetzung von Regeln und Gesetzen im Straßenverkehr. Bei der Verkehrsknotenüberwachung werden Kreuzungen mit Videoanlagen observiert, unterstützt durch Flusssensoren und Zählschwellen. Aufgrund der Analyse könnte in die Ampelsteuerung eingegriffen werden, um den Verkehrsstrom zu verbessern. Gleiches gilt für die Verkehrsflussüberwachung, die ebenfalls das Ziel hat, den Verkehrsfluss durch bestimmte lenkende Eingriffe zu optimieren. Weitere Beispiele aus der Industrie, in denen Videoüberwachung eingesetzt wird und die daher potentiell für die Anwendung von Szeneninterpretation interessant sein könnten, sind Raumfahrt, Kernreaktoren, Wiederaufbereitungsanlagen, Schifffahrt und Ölbohrplattformen.

1.2.2 Intelligente Wohnumgebungen

Ein anderes Anwendungsgebiet ist die Erschaffung intelligenter Haus- und Wohnungsumgebungen. Zielsetzungen sind dabei zum Einen, den Komfort für die Bewohner zu verbessern, und zum Anderen, bei älteren oder pflegebedürftigen Menschen im Notfall automatisch Hilfe zu rufen. Durch Kameras und Sensoren werden die Räume und Personen observiert und der Zustand verschiedener Gegenstände registriert (z.B. Ofen an/aus, Dusche an/aus, Sessel besetzt/frei, Telefon wird benutzt/wird nicht benutzt etc.). Auf der Basis dieser Daten wird versucht, durch Methoden der Szeneninterpretation komplexe Handlungen abzuleiten (z.B.

[1] EC Grant 214975, Projekt Co-Friend.

Aufstehen, Frühstücken, Kochen). Diese Handlungen können Aktionen auslösen, um den Bewohner zu unterstützen. Untersuchungen zu dieser Thematik sind Teil dieser Arbeit (s. Abschnitt 7.2) und basierend auf Daten des *CASAS Smart-Home-Projekts* [2] und des *Living-Place-Projekts* [5] der HAW[2].

1.2.3 Erkennung krimineller Aktivitäten

Die Welt wird von einer stetig steigenden Anzahl Kameras beobachtet: In einem Bericht aus dem Jahr 2007 wird die Zahl der Kameras des *Closed Circuit Television* (CCTV), die auf britische Bürger gerichtet sind, auf 4,2 Millionen geschätzt (s. [74]). In Hamburg überwachten seit 2006 zwölf PTZ-Kameras die Reeperbahn. Diese wurden jedoch im Juni 2010 wieder abgeschaltet, aufgrund eines Gerichtsbeschlusses, der die Persönlichkeitsrechte der Anwohner stärkte. Bis dahin wurde die Reeperbahn von zwölf Beamten in vier Schichten rund um die Uhr in einem Monitorraum überwacht (s. [20]). Es befinden sich Kameras beispielsweise in Räumen und Fahrzeugen von öffentlichen Verkehrsmitteln, Banken, Flughäfen und auf öffentlichen Plätzen. Wenn man davon ausgeht, dass die Mehrzahl dieser Kameras nicht nur durch ihre bloße Existenz zur Abschreckung dienen, dann wird der Bedarf an automatischer Bildanalyse deutlich, denn die Überwachung durch Menschen kann aus Kostengründen sicherlich nur stichprobenhaft erfolgen und dürfte darüber hinaus auch recht eintönig und ermüdend sein. Wünschenswert wäre hier ein automatisches Realzeit-Monitoringsystem, welches eine Vielzahl von Kameras überwacht, verdächtige oder bedrohliche Verhaltensmuster entdeckt und ggf. Alarm schlägt.

1.2.4 Fahrerassistenzsysteme

Die Einführung passiver Sicherheitssysteme, wie Sicherheitsgurt oder Airbag, haben einen wesentlichen Beitrag zur Reduzierung der Unfallzahlen im Straßenverkehr geleistet (s. [21]). Der Nutzen solcher passiver Sicherheitssysteme ist jedoch weitgehend ausgereizt und kommt außerdem erst im Falle eines Unfalls zum Tragen. Abbildung 2 zeigt, dass ein wesentlicher Teil tödlicher Unfälle auf Bundesautobahnen durch Wahrnehmungsstörungen, Fehleinschätzungen und unvorhergesehene Ereignisse verursacht wird (s. [101], S. 2).

[2] Hochschule für Angewandte Wissenschaften

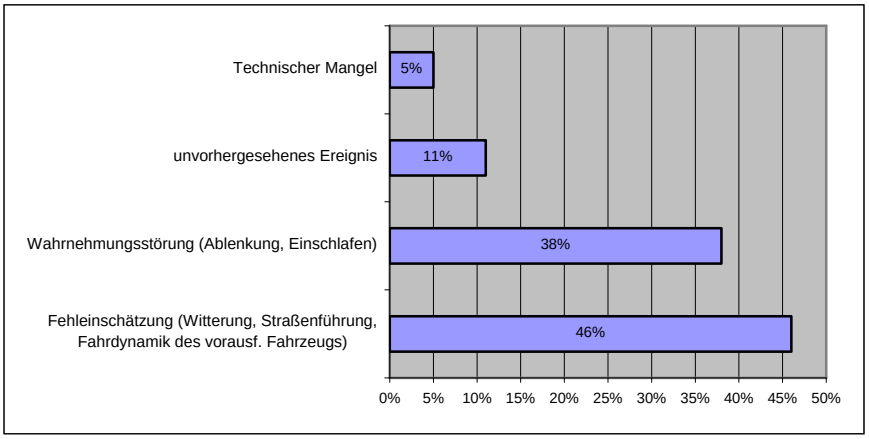


Abbildung 2: Unfallauslösende Ereignisse auf Bundesautobahnen mit Getöteten (2004).

Das unterstreicht den potentiellen Nutzen von Fahrerassistenzsystemen (FAS), die selbstständig in die Fahrdynamik eingreifen und damit aktiv die Unfallvermeidung unterstützen. Beispiele für ältere FAS sind das Anti-Blockier-System (ABS) und das Elektronische Stabilitätsprogramm (EPS). Durch maschinelle Wahrnehmung der Fahrzeugumwelt konnten neue FAS entwickelt werden, wie z.B. der Spurhalteassistent (LDW), der Spurwechselassistent (BSD) oder der Abstandsregeltempomat (ACC). Diese neueren FAS arbeiten mit Radarsensoren, Kameras und Laserscannern. Für das Verhindern eines potentiellen Unfalls sind die letzten zwei Sekunden ausschlaggebend. In dieser Zeitspanne wird bei modernen FAS versucht, ein möglichst umfassendes Bild der Fahrzeugumwelt zu erlangen (s. [101], S. 3).

FAS auf der Basis von Szeneninterpretationssystemen, wie sie in dieser Arbeit betrachtet werden, könnten einen Schritt weiter gehen: Aufgrund der verfügbaren Modelle könnten Vorhersagen über den weiteren Verlauf einer Verkehrssituation generiert werden, die wiederum bestimmte Eingriffe in die Fahrdynamik auslösen. Wird z.B. ein Ball detektiert, der über die Strasse rollt, könnte das als Beginn der Handlung „Ball rollt über die Strasse, gefolgt von einem Kind” erkannt werden und zu einer automatischen Vollbremsung führen, während ein anderer Gegenstand, wie z.B. ein Karton, diese Maßnahme nicht auslösen würde, da sie ein gewisses Gefahrenpotential birgt. Dieses Beispiel verdeutlicht noch einmal die Wichtigkeit der Realzeitverarbeitung eines Szeneninterpretationssystems. Die Qualität der Ergebnisse der Bildverarbeitungskomponenten stellt jedoch weiterhin einen limitierenden Faktor dar: Aktuell arbeiten BMW, Mercedes und Opel an Systemen, die Verkehrsschilder (Tempolimits und Überholverbote) erkennen. Diese haben zurzeit eine Erkennungsquote von 80%, bei Nacht 60% (s. [11]). Es erscheint also fraglich, ob auf der Basis solcher Eingabedaten auf die Fahrdynamik des Fahrzeugs Einfluss genommen werden sollte. In jedem Fall könnten Szenen-

interpretationssysteme eine verbesserte Wahrnehmung der Umwelt erbringen und so wichtige Warnungen oder Hinweise für den Fahrer generieren.

1.2.5 Bild- und Videosuche

Szeneninterpretation kann für die multimodale Informationsbeschaffung von Bildern und Videos im Internet verwendet werden, bei der sowohl visuelle, als auch textuelle Objekte einbezogen werden.

Das Internet zeichnet sich durch eine unüberschaubare Menge an Informationen aus, die größtenteils unstrukturiert vorliegt und deren Repräsentation auf den Menschen als Endnutzer ausgerichtet ist. Er kann den Inhalt einer Webseite weitgehend problemlos erfassen, während eine Maschine dies in aller Regel nicht leisten kann. Daraus ergibt sich das Problem, dass eine bestimmte Information zwar prinzipiell verfügbar ist, angesichts der Fülle verfügbarer Daten aber nur schwer gefunden werden kann. Suchmaschinen wie Google, Yahoo, etc. leisten hier durch den Einsatz statistischer Methoden zwar bereits Erstaunliches, letztlich basieren diese Techniken jedoch auf der Suche von Zeichenketten in Texten (s. [63] , S. 10) oder auf simplen Bildvergleichen durch Histogrammanalysen und ähnlichen Techniken.

Es gibt zwei grundsätzlich verschiedene Ansätze, um dieses Problem zu lösen. Zum einen könnte versucht werden, durch Methoden der Künstlichen Intelligenz, die Informationen in ihrem derzeitigen Zustand semantisch zu verarbeiten. Eine Suchmaschine, die dieses Ziel verfolgt, ist *Wolfram Alpha* [107]. Die Antwortmöglichkeiten sind gegenwärtig jedoch begrenzt (s. [45]) und ein durchschlagender Erfolg erscheint zumindest in der näheren Zukunft als eher fragwürdig (s. [63], S.11). Zum anderen könnten die Informationen von vornherein in einer Weise repräsentiert werden, welche die Verarbeitung durch Maschinen ermöglicht. Das ist die Idee des *Semantischen Webs*. Ein Gremium zur Schaffung grundlegender Standards von Informations-Spezifikationssprachen wie XML, RDFS und OWL (*Web Ontology Language*) ist das *World Wide Web Konsortium* [22]. Das konzeptuelle Wissen wird dabei durch *Ontologien* repräsentiert (s. Abschnitt 4.2). Dadurch eröffnet sich die Möglichkeit, dass durch Inferenzregeln neue Zusammenhänge entdeckt werden, die zuvor nicht erkennbar waren. Weiterhin wird auf diese Weise die Grundlage für den Einsatz wissensbasierter Szeneninterpretation geschaffen. Suchanfragen, die typischerweise durch Beschreibungen medialer Daten auf höherer Ebene formuliert sind, die von Details abstrahieren, können dann sowohl Texte als auch Bilder und Videosequenzen einbeziehen (s. [76]). Wenn z.B. nach dem Begriff „Hochsprung“ gesucht wird, dann impliziert dies die Existenz mehrerer Objekte im Bild, wie eine Person und eine waagerechte Latte, welche bestimmte räumliche und – im Falle von Videosequenzen – zeitliche Be-

dingungen erfüllen müssen. Die Konzepte und Constraints sind dabei durch Ontologien repräsentiert, während einzelne Objekte in Bildern und Videosequenzen durch Bildverarbeitungsmethoden erkannt werden.

1.2.6 Internet der Dinge

Das Internet dient heute dem Austausch von Daten zwischen verschiedenen Computernetzwerken. Diese Netzwerke sind jedoch kaum mit der physischen Welt verknüpft. Die Idee des *Internets der Dinge* ist, dass eindeutig identifizierbare physische Objekte mit einer virtuellen Repräsentation im Internet verknüpft sind. Dadurch eröffnen sich neue Formen der Kommunikation zwischen Personen und Dingen und zwischen den Dingen selbst. Als technische Grundlage wird oft die automatische Identifikation und Lokalisierung der Objekte mittels RFID (*radio-frequency identification*) angesehen. Darüber hinaus können die Chips mit eingebauten Sensoren aber auch z.B. Temperatur, Feuchtigkeit, Vibration oder Helligkeit messen. Anwendungsbeispiele sind Paketverfolgung über das Internet, automatisches Nachbestellen von Druckerpatronen, wenn eine bestimmte Füllmenge unterschritten wird, oder Kühlschränke, die selbständig neue Artikel bestellen. Hier gibt es Berührungspunkte und ähnliche Ideen zu den intelligenten Wohnungen (s. Abschnitt 1.2.2). Visionäre Anwendungsmöglichkeiten werden in [18] beschrieben: Durch Verknüpfung mit Nanotechnologie könnte z.B. die Trinkwasserqualität überwacht werden oder die Diagnose und Behandlung von Krankheiten.

Szeneninterpretation könnte dort zum Einsatz kommen, wo das Zusammenspiel vieler Objekte (und möglicherweise auch Personen), welches in der Regel räumliche und zeitliche Constraints umfasst, die Erkennung komplexer Zustände oder Handlungen erfordert, die dann entsprechende (Re)aktionen auslösen.

1.2.7 Automatisches Analysieren von Sportspielen

Computerspiele haben sich in den letzten Jahren signifikant weiterentwickelt. Moderne Spiele zeichnen sich durch hochdetaillierte Grafik, realistische Physikeffekte und Computergegner aus, die einen gewissen Grad intelligenten Verhaltens simulieren. Der Einsatz von Methoden der Künstlichen Intelligenz bei Computerspielen ist vielfältig, abhängig vom jeweiligen Genre (Arcade, Puzzle, Rollenspiele, Simulation, Sportspiele, Strategie, etc.). Eine interessante Anwendungsmöglichkeit für die Szeneninterpretation ergibt sich bei Sportspielen, wie z.B. Fußball, Eishockey, Basketball, etc. Diese Spiele bieten oftmals das Feature einer TV-ähnlichen Präsentation, also auch das eines Kommentators, der die aktuellen Spielsituationen kommentiert (s. [86]). Die Herausforderung ist demnach, kom-

plexe Spielmuster zu erkennen und passenden Beschreibungen zuzuordnen. Die Evidenzen – im Wesentlichen Spieler, Ball und deren Positionen – werden dabei zwar nicht von einer Bildverarbeitungskomponente geliefert, die Interpretationsverfahren auf der höheren Ebene sind jedoch die gleichen.

1.2.8 Robotik

Industrieroboter stellen eine Schlüsselkomponente für die Automation dar. Abbildung 3 zeigt die Entwicklung des Einsatzes von Industrierobotern seit 1973 (s. [72]). Es ist jedoch ein neuer Trend feststellbar, dass immer mehr Roboter nicht nur für den Einsatz im industriellen Umfeld entwickelt werden, sondern auch für die Ausführung spezieller Dienste im gemeinsamen Umfeld mit dem Menschen (s. [106]). Dieses Umfeld ist in der Regel sehr viel dynamischer und komplexer, als die vordefinierte Arbeitsumgebung von Industrierobotern. Roboter dieser Art bezeichnet man gewöhnlich als *Serviceroboter*.

Eine vorläufige Definition eines Serviceroboters findet sich bei [6]:

> *"A service robot is a robot which operates semi- or fully autonomously to perform services useful to the well-being of humans and equipment, excluding manufacturing operations."*

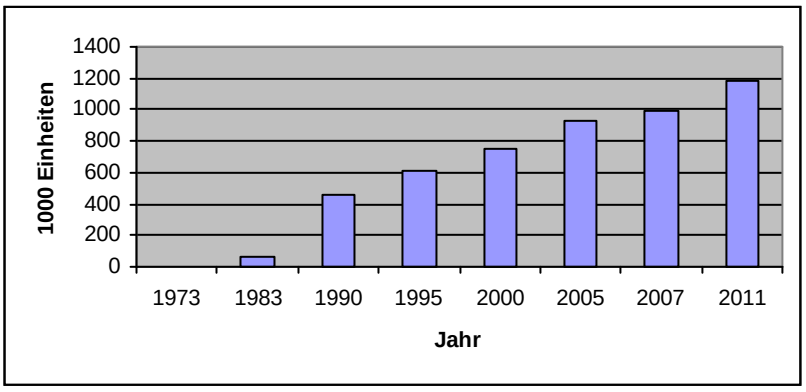


Abbildung 3: Entwicklung des Einsatzes von Industrierobotern seit 1973.

Im Gegensatz zu industriellen Robotern sind die meisten Serviceroboter mit wesentlich mehr Sensorik und Rechenkraft ausgestattet, um ihre Umwelt wahrzunehmen und die Ergebnisse für autonomes Verhalten auszuwerten. Dadurch gewinnen Forschungsgebiete wie Maschinelles Sehen und Szeneninterpretation in diesem Bereich zunehmend an Bedeutung.

Die Einsatzmöglichkeiten für semi- und vollautonome Roboter sind vielfältig (s. [99]). Sie können in technischen Laboren und generell in einer dynamischen Umgebung einfache oder sich oft wiederholende Tätigkeiten ausführen. Als Überwachungsroboter führen sie Patrouillengänge innerhalb oder außerhalb von Gebäuden durch. Weiterhin übernehmen sie Aufgaben in Umgebungen, die für Menschen gefährlich sind, wie z.B. in radioaktiv belasteten Gebieten, unter Wasser, im Weltraum und auf anderen Himmelkörpern. Haushaltsroboter können bereits Rasenmähen, Staubsaugen und Fensterputzen. Weitere mögliche Aufgaben wären Geschirr einräumen, Wäsche zusammenlegen, Essen oder Medizin für Personen bringen, die in ihrer Mobilität eingeschränkt sind, u.a.m.

Derzeit sind diese Serviceroboter oft auf ein spezielles Aufgabengebiet beschränkt (wie z.B. Rasenmähen) und benötigen daher z.T. nur einfache Sensoren. Aus wirtschaftlichen und praktischen Gründen wäre es von Vorteil, wenn ein Roboter vielfältige Aufgaben übernehmen könnte. Grundsätzlich gilt: Die Qualität und Vielseitigkeit der Aufgaben ist wesentlich davon abhängig, wie gut diese Roboter ihre Umwelt wahrnehmen können. Die Szeneninterpretation kann einen entscheidenden Beitrag leisten, um diese Wahrnehmung zu verbessern, da diese sich nicht mehr ausschließlich auf die Sensorik stützt, sondern zusätzlich auf die Modelle der konzeptuellen Wissensbasis.

1.3 Ziele und Anforderungen

Die vorliegende Arbeit verfolgt das Ziel, ein generisches Rahmenwerk für ein wissensbasiertes Szeneninterpretationssystem zu entwickeln. Dabei handelt es sich um die oberste Schicht einer allgemeinen Architektur für ein vollständiges Interpretationssystem (s. Abbildung 1). Für ein derartiges Rahmenwerk müssen wesentliche Anforderungen erfüllt sein:

- Fähigkeit der statischen Verarbeitung von Bildern und der dynamischen Verarbeitung von Videosequenzen,
- Verwendung standardisierter Wissensrepräsentationsmethoden,
- Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses,
- Echtzeitfähigkeit,
- Verarbeitung unsicherer und unvollständiger Eingabedaten.

Statische und dynamische Verarbeitung. Das System muss in der Lage sein, sowohl statische Bilder als auch dynamische Videosequenzen zu verarbeiten. Der wichtige Unterschied zwischen beiden Verfahren ist, dass bei der Analyse von Bildern keine zeitlichen Constraints existieren und alle Evidenzen im Prinzip instantan vorliegen, während sie bei Videosequenzen Schritt für Schritt, mit einem entsprechenden Zeitstempel, geliefert werden. Die Verarbeitung der zeitlichen Constraints ist in der Regel von essentieller Bedeutung für die Analyse von Videosequenzen und wird daher auch in dieser Arbeit, die sich vorwiegend mit der dynamischen Szeneninterpretation befasst, eingehend behandelt.

Standardisierte Wissensrepräsentationsmethoden. Ein generisches Interpretationssystem muss sich einerseits leicht an Veränderungen innerhalb einer Anwendung adaptieren lassen und andererseits mit geringem Aufwand für unterschiedliche Domänen eingesetzt werden können. Dazu muss sowohl die Wissensbasis als auch die Inferenzkomponente effizient erstellt und gewartet werden können. Das setzt die Verwendung standardisierter Wissensrepräsentationsmethoden voraus. Sie eröffnen die Möglichkeit für automatische Konsistenzprüfungen, standardisierte Inferenzprozeduren und den Einsatz bestehender Software-Werkzeuge zum Bearbeiten von Ontologien (s. Abschnitt 4.2).

Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses. Für ein generisches Rahmenwerk für die Szeneninterpretation ist es wichtig, dass die deklarative Beschreibung der Modelle der Wissensbasis klar getrennt ist von den Kontrollstrukturen des Interpretationsprozesses. Durch diese Trennung ist gewährleistet, dass Änderungen an der Wissensbasis vorgenommen werden können, ohne, dass komplizierte Programmänderungen erforderlich werden. Andererseits ist es möglich, den Interpretationsprozess zu erweitern oder zu modifizieren, ohne vorhandene Wissensbasen anzupassen.

Echtzeitfähigkeit. Für viele Anwendungen ist es Voraussetzung, dass Videosequenzen in Echtzeit verarbeitet und interpretiert werden können (s. Abschnitt 1.2). Folglich muss das System in der Lage sein, dem Benutzer jederzeit Interpretationsergebnisse abzuliefern, basierend auf den bis dahin zur Verfügung stehenden Evidenzen. Wie bereits in Abschnitt 1.1 dargestellt, ist die Mehrdeutigkeit eine intrinsische Eigenschaft der Szeneninterpretation, welche durch die Verrauschtheit der Eingabedaten verstärkt wird. Ein echtzeitfähiges System muss daher auch in einem frühen Stadium der sich entwickelnden Szene Interpretationsentscheidungen treffen können, mit nur geringen Kontextinformationen und hoher Mehrdeutigkeit. Dazu ist es notwendig, verschiedene mögliche Interpretationsalternativen zu verfolgen und zu bewerten, beispielsweise durch Backtracking oder Parallelverarbeitung. Weiterhin wird ein Präferenzmaß benötigt, um Zwischen- oder Endergebnisse von alternativen Interpretationen zu bewerten. Bereits in [84] wurde gezeigt, dass Szeneninterpretation im Wesentlichen als ein Suchproblem im Raum dieser alternativen Möglichkeiten aufgefasst werden kann.

Verarbeitung unsicherer und unvollständiger Eingabedaten. Wie bereits in Abschnitt 1.1 erwähnt, sind die von der Mittelschicht gelieferten Evidenzen oft verrauscht, unvollständig und mit Unsicherheit behaftet, aufgrund von allgemein bekannten Problemen der Objekterkennung. Diese werden beispielsweise durch Verdeckungen oder andere Einflussfaktoren, wie ungünstige Licht- oder Witterungsverhältnisse, verursacht. Deshalb ist ein rein deduktiver Szeneninterpretationsansatz wenig erfolgversprechend. Er sollte sich vielmehr auf eine Mischung aus datengetriebenen bottom-up und abduktiven top-down Verarbeitungsschritten stützen. Letztere können durch modellbasierte Hypothesen realisiert werden, deren Instanziierung die vorhandene Evidenz „erklärt“. Hypothesen stellen Kontextwissen der konzeptuellen Wissensbasis zur Verfügung und ermöglichen es unter bestimmten Umständen, fehlende Evidenzen zu ergänzen und Annahmen über unsichere Informationen zu untermauern. Es ist die wesentliche Aufgabe und Herausforderung eines Interpretationssystems, das Zusammenspiel von bottom-up und top-down Schritten so zu kontrollieren, dass es zu der besten Lösung gelangt, basierend auf den unsicheren und unvollständigen Evidenzen und den zur Verfügung stehenden Modellen.

1.4 Gliederung der Arbeit

Nachdem einleitend das Thema der Arbeit und Anwendungsgebiete betrachtet sowie Ziele und Anforderungen formuliert wurden, befasst sich das nun folgende Kapitel 2 mit dem Stand der Forschung im Bereich der Szeneninterpretation. Anschließend wird in Kapitel 3 der konzeptionelle Ansatz für ein generisches Rahmenwerk für ein wissensbasiertes Szeneninterpretationssystem erarbeitet, basierend auf den Zielen und Anforderungen und unter Berücksichtigung der gewonnenen Erkenntnisse der bestehenden Ansätze aus Kapitel 2.

In Kapitel 4 werden die Grundlagen dargestellt, auf denen der in dieser Arbeit entwickelte Ansatz basiert. Die zentralen Themen sind dabei OWL Ontologien und regelbasierte Systeme.

In Kapitel 0 wird die Entwicklung eines generischen Rahmenwerks für die wissensbasierte Szeneninterpretation dargestellt. Die Techniken werden anhand von Beispielen aus der Domäne der Flughafenvorfeldaktivitäten verdeutlicht. Dazu wird zunächst auf die Repräsentation von Aktionsmodellen in einer OWL Ontologie eingegangen. Anschließend wird erarbeitet, wie aus einer derartigen Ontologie automatisch Regeln für ein regelbasiertes Szeneninterpretationssystem generiert werden. Weiterhin wird der Interpretationsprozess beschrieben und Betrachtungen zur Komplexität und zum Determinismus angestellt. Daraufhin wird erläutert, wie

der Interpretationsprozess durch ein probabilistisches Präferenzmodell gesteuert wird.

In Kapitel 6 wird die Implementierung des in dieser Arbeit entwickelten Szeneninterpretationssystems SCENIOR vorgestellt. Dabei wird auch auf Konfigurations- und Erweiterungsmöglichkeiten eingegangen.

In Kapitel 7 werden die vorgestellten Verfahren und das Interpretationssystem SCENIOR anhand von Experimenten zur Domäne der Flughafenvorfeldaktivitäten und zur Domäne intelligenter Wohnumgebungen demonstriert.

Den Abschluss bildet Kapitel 8 mit einer Zusammenfassung der wichtigsten Ergebnisse dieser Arbeit und einem Ausblick über weitere Entwicklungsmöglichkeiten.

2 Stand der Forschung

Obwohl der Szeneninterpretation bisher deutlich weniger Aufmerksamkeit im Forschungsbereich des Maschinellen Sehens gewidmet wurde als beispielsweise der Objekterkennung, so gibt es doch eine ganze Reihe verwandter Arbeiten auf diesem Gebiet. Dieses Kapitel soll einen Überblick über wichtige Arbeiten geben, von den Anfängen der Szeneninterpretation bis zum Stand der aktuellen Forschung, und die Einordnung der vorliegenden Arbeit in diesen Rahmen darstellen.

2.1 Die Anfänge

Bereits im Jahr 1975 gab es Arbeiten zur symbolischen Szeneninterpretation, z.B. die Dissertation von Badler [28], in der Szenenbeschreibungen aus künstlich generierten Skizzen einer Straßenszene abgeleitet werden. Für die Verarbeitung von natürlichen Szenen fehlten damals noch entsprechend leistungsfähige Bildanalyseverfahren. Badler benutzt räumliche Relationen zwischen Paaren von Objekten, um eine Momentaufnahme einer Szene zu beschreiben, und Veränderungen dieser relationalen Struktur, um die zeitliche Entwicklung darzustellen. Zeitliche Konzepte, die Bewegungen beschreiben, wie z.B. „überqueren“, werden dann durch Regeln mit entsprechenden Vor- und Nachbedingungen abgeleitet. Diese Arbeit zeigt, dass räumliche Prädikate ein wichtiges Verbindungsglied zwischen quantitativen niederen Bildverarbeitungsdaten und qualitativen höheren Beschreibungen darstellen. Interessanterweise wird in der Zusammenfassung der Arbeit erwähnt, dass von der Bildverarbeitung nicht immer eine korrekte Objektidentifizierung erwartet werden kann und, dass es nützlich wäre, die Identifizierung auch aus der Verwendung des Objekts abzuleiten. Wie im weiteren Verlauf dieser Arbeit deutlich werden wird, ist die robuste Objektidentifizierung auch heute noch ein diffiziles Problem, welches die Szeneninterpretation erschwert. Die Idee, eine Objektidentifizierung aus der Funktion des Objekts abzuleiten, wurde in dieser Arbeit umgesetzt.

Ein erster systematischer Ansatz zur Bewegungsanalyse findet sich bei Tsotsos [102] aus dem Jahr 1980. Dort wird ein Rahmenwerk vorgestellt, welches Bewe-

gungen in Bildsequenzen erkennt und sie klassifiziert, entsprechend verschiedener Bewegungskonzepte, die in einer Wissensbasis abgelegt sind. Die Bewegungskonzepte werden in Form von *Frames* repräsentiert. Im Sinne der objektorientierten Programmierung kann man sich Frames als Objekte ohne Methoden vorstellen, mit anderen Worten, sie sind in eine hierarchische Vererbungsstruktur eingebettet und besitzen Attribute, sogenannte *Slots*. Übergeordnete Frames können ihre aktuellen Slotwerte, die *Rollenfüller*, an untergeordnete Frames vererben. Auf diese Weise wird eine *Taxonomie*, also ein Klassifikationsschema von generellen und spezifischen Bewegungskonzepten definiert. Komplexere Bewegungsabläufe werden als Komposition von elementaren Bewegungskonzepten beschrieben. Dadurch wird eine kompositionelle Hierarchie definiert.

Der Erkennungsprozess arbeitet nach dem Paradigma *Hypothetisieren-und-Testen*: Ausgehend vom ersten Bild der Sequenz und dessen Objekten, welche durch Bildverarbeitungsverfahren identifiziert werden, generiert das System Hypothesen über den weiteren Verlauf der Bewegung der Objekte, basierend auf den Modellen der Wissensbasis. Die Hypothesen beeinflussen dabei die niedere Bildverarbeitung der nachfolgenden Bilder, z.B. durch Eingrenzung des Suchbereichs von Objekten. Hier wird also bereits ein Feedback-Mechanismus von höheren zu niedrigen Verarbeitungsschichten realisiert (s. Abbildung 1). Im Testzyklus werden dann neue Daten der Bildverarbeitungskomponente mit den Vorhersagen der Hypothesen abgeglichen und bewertet. Dieser Vorgang wird in einer Schleife wiederholt. Das System wurde anhand von pathologischen Herzbewegungen von Menschen getestet.

Bewegungen mit Hilfe taxonomischer und kompositioneller Hierarchien zu strukturieren spielt in vielen Ansätzen zur Szeneninterpretation eine signifikante Rolle, so auch in einer frühen Arbeit von Neumann [80]: Das Projekt NAOS hat die Beschreibung von Objektbewegungen in einer Straßenszene mittels natürlicher Sprache zum Ziel. Hier werden bereits – im Gegensatz zur Arbeit von Badler [28] – reale Bilder als Eingabedaten verwendet. Allerdings werden Objekte, deren Positionen und weitere Eigenschaften pro Frame weitgehend manuell ermittelt. Das Ergebnis ist eine geometrische Szenenbeschreibung (GSB). Die Abläufe (*Ereignisse*) werden durch deklarative *Aggregate* beschrieben, die aus einem Kopf bestehen, welcher ein Prädikat der Szene darstellt, z.B. „überholen“, und einem Rumpf, der spezifizierte, wie dieses Prädikat verifiziert werden kann. Die Ablaufmodelle werden hierarchisch definiert: Der Rumpf komplexer Ereignisse wird durch andere komplexe oder primitive Ereignisse beschrieben. Ereignisse sind primitiv, wenn sie nicht weiter zerlegt werden können, und der Rumpf besteht dann aus einer Prozedur, die sich mit Hilfe der GSB verifizieren lässt, z.B. „prüfe, ob die Distanz zwischen `Objekt_1` und `Objekt_2` abnimmt“.

Eine zusätzliche Errungenschaft dieser Arbeit ist die Trennung der Ablaufmodelle von den Kontrollstrukturen des Ereigniserkennungsprozesses. Dadurch ist ge-

währleistet, dass Änderungen der Wissensbasis keine komplizierten Programmänderungen nach sich ziehen. Die Ablaufmodelle haben eine klare logische Interpretation; sie spezifizieren, dass der Kopf und der Rumpf logisch äquivalent sind:

$$< Kopf > \Leftrightarrow < Rumpf > . \tag{2-1}$$

Aus der Äquivalenz von Kopf und Rumpf kann sowohl der datengetriebene bottom-up Schritt abgeleitet werden:

$$< Rumpf > \Rightarrow < Kopf > , \tag{2-2}$$

welcher für die Szenenerkennung essentiell ist, als auch der modellgetriebene top-down Schritt

$$< Kopf > \Rightarrow < Rumpf > , \tag{2-3}$$

der das Aggregat in seine konjunktiven Bestandteile zerlegt. So kann z.B. aus der Existenz eines Überholvorgangs abgeleitet werden, dass es auch mindestens ein Bewegungsereignis geben muss, da dieser ein Teil des Überholvorgangs ist. Zeitliche Relationen zwischen den Ereignissen werden als Constraints formuliert und mit einem Constraint-System berechnet.

Bereits in diesen frühen Arbeiten sind grundlegende Konzepte entwickelt worden, welche in die Realisierung des hier vorgestellten Rahmenwerks einfließen: die Verwendung taxonomischer und kompositioneller Hierarchien und die Modellrepräsentation in Form von Aggregaten. Taxonomische Hierarchien bieten nützliche Inferenzmöglichkeiten, wie z.B. Vererbung, vermeiden Redundanz und verbessern die Wartbarkeit einer Wissensbasis. Kompositionelle Hierarchien bilden das Rückgrat für die Szeneninterpretation. Die Trennung der Ablaufmodelle von den Kontrollstrukturen des Ereigniserkennungsprozesses und die Kombination aus datengetriebenen bottom-up und modellgestützten top-down Interpretationsschritten wurden ebenfalls in die Anforderungen übernommen (s. Abschnitt 1.3). Auch die Verarbeitung zeitlicher Constraints in einem separaten Constraint-System ist ein wesentlicher Aspekt dieser Arbeit.

2.2 SIGMA – ein wissensbasiertes Bildanalysesystem

Ein komplexes Rahmenwerk für die wissensbasierte Szenenanalyse namens SIGMA wurde 1990 von Matsuyama und Hwang [75] vorgestellt. Es dient zur

Analyse von Luftbildaufnahmen und beschränkt sich – zumindest in der Beschreibung – auf statische Bilder. Folgende Probleme werden als die zentralen Schwierigkeiten bei der Szeneninterpretation genannt:

- unzuverlässige Segmentierung,
- Repräsentation und Schließen, basierend auf geometrischen Informationen,
- Schließen mit unvollständigen Informationen.

Dies sind wichtige Aspekte, die sich in den Anforderungen an das in dieser Arbeit angestrebte Ziel eines allgemeinen Rahmenwerks für die Szeneninterpretation widerspiegeln (s. Abschnitt 1.3). Die Systemarchitektur von SIGMA besteht, resultierend aus diesen identifizierten Problemen, aus drei Modulen und einem Schnittstellenmodul:

- dem *Low-Level Vision Expert* (LLVE), für wissensbasierte Bildsegmentierung,
- dem *Geometric Reasoning Expert* (GRE), für räumliches Schließen,
- dem *Model Selection Expert* (MSE), für die Modellselektion,
- dem *Question and Answer* Module, für Anfragen an das System.

Low-Level Vision Expert. Die Bildsegmentierung wird durch den MSE gesteuert: je nach Vorgabe des MSE und den daraus resultierenden Bildobjekten, die zu extrahieren sind (z.B. polygon-basierte Objekte), werden passende Operatoren und Parameteranpassungen der Bildverarbeitungsalgorithmen gewählt, die für diese Objekte optimal sind. Das entsprechende Wissen ist in einer eigenen Bildverarbeitungsdatenbank gespeichert, die domänenunabhängig ist und für beliebige Szenen verwendet werden kann.

Geometric Reasoning Expert. Der GRE ist das zentrale Modul des Systems und konstruiert eine Beschreibung der Szene durch Etablieren von räumlichen Relationen zwischen den Objekten. Die Objekte werden mit objektorientierten Strukturen repräsentiert, ähnlich den Frames. Jede Objektinstanz ist selbst ein aktiver Agent, der Hypothesen zugehöriger Objekte generiert, basierend auf dem Wissen, welches in der Objektklasse gespeichert ist. Beispielsweise generiert die Instanz für ein Haus Hypothesen über benachbarte Häuser und die zugehörige Straße. Auf diese Weise wird der Erkennungsprozess durch individuelle Objekte und den GRE geleitet: Die Objektinstanzen führen Schlussfolgerungen über die lokale Umgebung aus, während der GRE die Konstruktion einer global konsistenten Beschreibung der Szene koordiniert. Bei der bottom-up Analyse werden dabei räumliche Relationen etabliert, während die top-down Analyse die Suche nach neuen (fehlenden) Objekten aktiviert. Dazu beauftragt der GRE den MSE, Objekte gemäß

des spezifizierten Zielobjekts zu detektierten. Die neu erkannten Objekte werden mit den Hypothesen abgeglichen, die dann evtl. in einem fehlergetriebenen Verfahren modifiziert werden, bis eine konsistente Beschreibung der Szene konstruiert werden kann.

Model Selection Expert. Der MSE wird durch die top-down Analyse des GRE aktiviert, um ein spezifiziertes Zielobjekt zu detektieren. Der MSE beauftragt dazu den LLVE, die entsprechenden Bildeigenschaften zu extrahieren. Das geschieht in drei Prozessschritten:

1. *Selektion eines spezialisierten Objektmodells*. Die Objekte werden mit Hilfe einer `A-KIND-OF`-Relation in einer taxonomischen Hierarchie repräsentiert. Kontextinformationen werden benutzt, um eine geeignete Spezialisierung des zu suchenden Objekts auszuwählen. Beispielsweise wird zunächst angenommen, dass ein Haus eine ähnliche Bauweise besitzt wie die Häuser in der Nachbarschaft.

2. *Dekomposition von komplexen Objekten*. Mit Hilfe einer `PART-OF`-Relation wird eine partonomische Hierarchie definiert. Wenn das zu suchende Objekt aus mehreren Teilen besteht, wird es vom MSE in diese zerlegt, und dem LLVE mitgeteilt, nach Bildeigenschaften dieser Teilobjekte zu suchen.

3. *Berechnung der 2D-Erscheinung*. Nachdem ein Objektmodell mit Hilfe der ersten beiden Schritte berechnet wurde, gilt es im dritten Schritt die 2D-Erscheinung dieses Objekts zu bestimmen, also das Objektmodell der Szene in eine Erscheinung des Bildes zu transformieren. Anschließend kann eine Zielspezifikation für den LLVE generiert werden.

Wenn die Bildeigenschaften der gesuchten Objekte erfolgreich durch den LLVE extrahiert werden können, wird vom MSE eine Objektinstanz des Zielobjekts generiert und an den GRE zurückgegeben.

Die bottom-up Instanziierung von komplexen Objekten verläuft folgendermaßen: Die Instanz eines Objekts führt zur Instanziierung des Elternobjekts; dieses ist *teilinstanziiert*, solange nicht alle seine Teile instanziiert sind. Dabei werden dann Hypothesen für noch fehlende Teile generiert. Nur ein vollständig instanziiertes Objekt führt zur Instanziierung des Elternobjekts. Durch die Hypothesen wird die oben beschrieben top-down Analyse angestoßen.

Zusammenfassend kann man sagen, dass mit dem SIGMA Rahmenwerk bereits viele interessante Ansätze zur Szeneninterpretation realisiert wurden:

- Die Trennung von domänenspezifischem und domänenunabhängigem Wissen, welches wiederverwendet werden kann. Diese Trennung ist für die Flexibilität, Adaptivität und Wartbarkeit eines Szeneninterpretationssystems unerlässlich und wird auch in dem hier vorgestellten Rahmenwerk realisiert.

- Wie in den oben erwähnten Arbeiten werden auch hier taxonomische und kompositionelle Hierarchien verwendet.

- Ebenfalls wird eine Kombination von bottom-up und top-down Verarbeitungsschritten präsentiert, um auch unvollständige Informationen verarbeiten zu können.

2.3 Logikbasierte Ansätze

In Abschnitt 1.3 wurde die Notwendigkeit der Verwendung standardisierter Wissensrepräsentationsmethoden für komplexe Anwendungen dargestellt. Deshalb sind logikbasierte Ansätze für diese Arbeit von zentralem Interesse. Es gibt eine Reihe von Arbeiten in der Literatur zum Thema Szeneninterpretation, die sich auf logikbasierte Ansätze stützen. Einige wichtige dieser Arbeiten werden in diesem Abschnitt vorgestellt.

2.3.1 Modellkonstruktion

Einer der ersten logikbasierten Rahmenwerke für die Szeneninterpretation geht auf Reiter und Mackworth [88] aus dem Jahr 1987 zurück. Sie zeigten, dass die Szeneninterpretation für eine endliche Domäne als logische Modellkonstruktionsaufgabe formuliert werden kann. Diese stellt sich als Suche nach einer Instanziierung der Axiome der Wissensbasis dar, welche die Domäne beschreiben und die konsistent ist mit der tatsächlichen Evidenz. Bei der Wissensrepräsentation wird prinzipiell unterschieden zwischen dem Wissen über das Bild und dem Wissen über die Szene und der Darstellungsrelation, also wie sich ein Objekt der Szene im Bild darstellte. Dementsprechend gibt es drei Arten von Axiomen:

- *Bildaxiome*. Mit den Bildaxiomen wird modelliert, welche primitiven Objekte es gibt und welche Relationen zwischen diesen Objekten bestehen.

- *Szenenaxiome*. Mit den Szenenaxiomen werden die primitiven und komplexen Szenenobjekte modelliert, deren kompositionelle und taxonomische Hierarchie und Relationen.

- *Abbildungsaxiome*. Mit den Abbildungsaxiomen wird ein Objekt im Bild genau einem primitiven Objekt der Szene zugeordnet. Beispielweise kann eine `Linie` ein Bildobjekt sein, welches einem der primitiven Szenenobjekte `Strasse, Fluss` oder `Ufer` zugeordnet werden kann.

Mit dieser Axiomatisierung kann eine formale Definition einer Szeneninterpretation wie folgt gegeben werden: die Interpretation eines Bildes ist ein Modell (im streng logischen Sinne) für die Bild-, Szenen- und Abbildungsaxiome.

Im Allgemeinen ist das Problem unentscheidbar, ob ein solches Modell für eine beliebige Menge von Formeln der Prädikatenlogik erster Ordnung, wie sie hier verwendet wird, überhaupt existiert. Weiterhin könnte es unendlich viele verschiedene Modelle geben. Und selbst im Falle von endlich vielen Modellen könnte sich die Berechnung auch mit wenigen Objekten als praktisch nicht durchführbar erweisen. Es stellte sich also die Frage, ob es bestimmte Eigenschaften der Szeneninterpretation gibt, die dieses Problem entschärfen. Auch wenn diese Frage nicht abschließend beantwortet werden kann, so wird darauf hingewiesen, dass es nur endlich viele primitive Objekte in einem Bild gibt und, bedingt durch die Abbildungsaxiome, auch nur endlich viele Szenenobjekte und somit auch nur endlich viele Modelle; damit wird das Problem – zumindest prinzipiell – berechenbar.

Später wurde das Paradigma der Szeneninterpretation als Modellkonstruktion von Neumann und Möller [82] auf *Beschreibungslogiken* (s. Abschnitt 4.1.1) übertragen. Die meisten Beschreibungslogiken (und auch die in dieser Arbeit betrachteten) sind eine Untermenge der Prädikatenlogik erster Ordnung, im Gegensatz zu dieser aber entscheidbar. Eine weitere wünschenswerte Eigenschaft von Beschreibungslogiken ist die Möglichkeit der objektzentrierten Repräsentation von Konzepten. Die sogenannte *TBox* (terminological box) enthält das terminologische Wissen, mit anderen Worten, das Wissen über die Konzepte, während in der *ABox* (assertional box) das Wissen über die Entitäten bzw. Instanzen dieser Konzepte enthalten ist, sowie deren Beziehung untereinander. Die ABox repräsentiert den Zustand der modellierten Welt.

In der Arbeit von Neumann und Möller wird die Szeneninterpretation anhand der Domäne einer Tischdeckszene erläutert. Es gibt verschiedene Abstraktionsebenen der Beschreibung: Auf einer niedrigen Ebene spricht man z.B. vom „Platzieren einer Gabel neben einem Teller“, während man die Situation auf einer höheren Ebene als „Der Tisch wird für das Frühstück gedeckt“ beschreiben kann. Diese verschiedenen Abstraktionsebenen werden durch taxonomische und kompositionelle Hierarchien in einer konzeptuellen Wissensbasis repräsentiert. Außerdem besitzt eine derartige Szene eine charakteristische räumlich-zeitliche Struktur: Einerseits gibt es räumliche Constraints, z.B., dass die Gabel links und das Messer rechts vom Teller liegen, und andererseits zeitliche Constraints, z.B., dass normalerweise zuerst der Teller, dann die Untertasse und dann die Tasse platziert wird.

Die wesentliche Struktur zum Beschreiben der Konzepte der Wissensbasis ist das *Aggregat*. Ein Aggregat definiert ein Konzept, dessen Teile und die Constraints, die zwischen den Teilen gelten. Das können sowohl physikalische Objekte als auch Ereignisse sein. Das Aggregat für `Gedeck-Platzieren` wird z.B. folgendermaßen definiert: der Slot `Eltern: Aktivität` ordnet das Aggregat als Unterkonzept von `Aktivität` ein; Aggregate sind also in einer taxonomischen Hierarchie strukturiert. Als Teile werden drei Transportereignisse, jeweils mit einem Teller, einer Untertasse und einer Tasse, definiert und die Teile des Gedecks (Teller, Untertasse, Tasse). Die Teile selbst können wieder Aggregate sein, so dass eine kompositionelle Hierarchie aufgebaut wird. Außerdem werden Zeitpunkte definiert, jeweils einer für den Beginn und einer für das Ende eines Ereignisses. Mit dem Slot `Constraints` werden Identitäts-, zeitliche und räumliche Constraints definiert. Identitäts-Constraints legen hier beispielsweise fest, dass die Teile der Transportereignisse mit den entsprechenden Teilen des Gedecks identisch sein müssen. Ausgedrückt mit Hilfe der Beschreibungslogik hat das Aggregat folgende generische Struktur (s. [82]):

$$\begin{aligned}&\texttt{Aggregate\_Concept} \equiv \texttt{Parent\_Concept} \sqcap \\ &\quad \exists_{=1}\texttt{hasPartRole}_1.\texttt{Part\_Concept}_1 \sqcap \ldots \sqcap \\ &\quad \exists_{=1}\texttt{hasPartRole}_k.\texttt{Part\_Concopt}_k \sqcap \\ &\quad \texttt{conceptual constraints}\end{aligned} \tag{2-4}$$

Das Aggregat mit dem Namen `Aggregat_Concept` ist definiert durch das taxonomische Elternkonzept `Parent_Concept`, durch die Teile, die durch eine ausgezeichnete Rolle (hier: `hasPartRole`) definiert werden und durch die konzeptuellen Constraints, welche Beziehungen zwischen dem Aggregatkonzept und den Teilen definieren oder zwischen den Teilen untereinander. Über weitere Rollen können zusätzliche Eigenschaften des Aggregats definiert werden, beispielsweise Rollen wie `hasStartTime` und `hasFinishTime` zum Zuweisen von Start- und Endzeitpunkten.

Entsprechend der in Abbildung 1 beschriebenen Architektur wird eine geometrische Szenenbeschreibung (GSB) als Schnittstelle zwischen der niederen Bildverarbeitung und der höheren Bildanalyse verwendet. Aus der GSB werden primitive Ereignisse abgeleitet, die als Basis für komplexere Ereignisse dienen. Primitive Ereignisse sind z.B.

- Objektbewegung,
- Annäherung oder Entfernung eines Objekts relativ zu einem anderen Objekt,
- drehende Objektbewegung,
- Aufwärts- oder Abwärtsbewegung eines Objekts.

Im Gegensatz zu Reiter und Mackworth [88] wird die Szeneninterpretation als *partielle* Modelkonstruktion aufgefasst. „Modell“ ist dabei im streng logischen Sinne zu verstehen: Gesucht wird also eine Zuordnung von Symbolen (Konstanten, Prädikaten und Funktionen) logischer Formeln zu Objekten bzw. Ereignissen, Prädikaten und Funktionen einer Domäne der realen Welt, so dass die Formeln wahr werden. Es gibt drei Arten von Formeln:

- für generisches Wissen über die Welt,
- für spezifisches Wissen über die Szene, in Form von Evidenzen und Kontext,
- für Aussagen, die generiert werden, um die Szene zu beschreiben.

Dabei sind letztere Formeln – für die Beschreibung der Szene – nicht gegeben, sondern werden inkrementell konstruiert. „Partiell“ besagt, dass nicht alle Formeln der Wissensbasis Domänenobjekten zugeordnet werden müssen, da die niedere Bildverarbeitung im Allgemeinen nicht perfekt ist.

Ein DL-System stellt in der Regel verschiedene Inferenzprozeduren und Konsistenzchecks für TBox und ABox zur Verfügung, z.B. für zwei Klassen C und D:

- Klassenäquivalenz (testet, ob die Klassen C und D äquivalent sind),
- Unterklassenbeziehung (testet, ob C eine Unterklasse von D ist),
- Klassendisjunktheit (testet, ob die Klassen C und D disjunkt sind),
- Globale Konsistenz (testet, ob eine Wissensbasis im Sinne der Prädikatenlogik widerspruchsfrei ist),
- Instanzprüfung (testet, ob ein Individuum a zur Klasse C gehört),
- Klasseninstanzen (liefert alle in der ABox enthaltenen Instanzen einer Klasse C).

Es ist eine zentrale Idee der Verwendung von Beschreibungslogiken, diese gegebenen Inferenzen und Konsistenzchecks zu verwenden, um eine konsistente Sze-

neninterpretation zu gewährleisten. Dies ist für die Adaptivität und Wartbarkeit eines Szeneninterpretationssystems von großem Nutzen. Deshalb sind Beschreibungslogiken ein zentraler Aspekt der in dieser Arbeit entwickelten Ansätze.

Der Interpretationsprozess. Bereits durch die Arbeit von Neumann und Weiss [84] wurde gezeigt, dass das Konstruieren einer Szeneninterpretation im Wesentlichen ein Suchproblem im Raum der möglichen Interpretationen ist, der durch taxonomische und kompositionelle Relationen aufgespannt wird. Folgende zentrale Interpretationsschritte wurden dazu identifiziert:

- *Aggregatinstanziierung*. Instanziiert ein Aggregat, wenn die Teile (nicht notwendigerweise alle) instanziiert und die Constraints erfüllt sind (Aufwärtsschritt in der kompositionellen Hierarchie).
- *Aggregatexpandierung*. Instanziiert die Teile eines Aggregats, wenn dieses bereits instanziiert ist (Abwärtsschritt in der kompositionellen Hierarchie).
- *Instanzspezialisierung*. Spezialisiert ein Konzept (Abwärtsschritt in der taxonomischen Hierarchie).
- *Instanzvereinigung*. Verschmelzen von identischen Instanzen, die separat erzeugt wurden (z.B. durch Generierung einer neuen Instanz und einer Aggregatexpandierung).

Basierend auf der GSB wird nun ein Individuum in die ABox eingetragen und zunächst überprüft, welchem Aggregat es zugeordnet werden könnte; die Instanzprüfung unterstützt diesen Schritt. Es gibt jedoch keine Unterstützung für die Strategie, *welchem* Aggregat – bei möglicherweise mehreren Kandidaten – das Individuum zugewiesen werden soll. Das erfordert ein Präferenzmaß, welches sich außerhalb des Rahmens des DL-Systems befindet. Der nächste Schritt ist Instanzspezialisierung, der ebenfalls durch ein DL-System unterstützt wird. Die Aggregatexpandierung ist determiniert und kann als ein neuer Service in ein DL-System integriert werden. Die Instanzverschmelzung wird durch Query-Services unterstützt.

Für die Berechnung des Präferenzmaßes wird ein probabilistischer Ansatz mit baumartigen Bayes-Netzen vorgeschlagen, der zugeschnitten ist auf die speziellen Strukturen von Aggregaten und Anforderungen der Szeneninterpretation. Auf diesen probabilistischen Ansatz wird im weiteren Verlauf dieser Arbeit noch detaillierter eingegangen (s. Abschnitte 2.4.3 und 5.4).

Ein hybrider Ansatz mittels Modellkonstruktion und statistischen Methoden wurde kürzlich von Riboni und Bettini [89] vorgestellt, anhand der Domäne der Betreuung älterer Menschen. Ein vorrangiges Ziel war das Ableiten von Aktivitäten

anhand von GPS-Daten und weiteren Sensoren. Das System wurde für Mobiltelefone entwickelt.

Zunächst wird in einer Trainingsphase mit Methoden des überwachten Lernens ein statistisches Modell von einer gegebenen Menge von Aktivitäten gelernt. Das Ergebnis ist ein Merkmalsvektor für jede Aktivität, bei der jedes Merkmal einem Maß entspricht, typischerweise einer Statistik über Messwerte verschiedener Sensoren. Bei diesem gebräuchlichen Ansatz wird implizit oftmals die Unabhängigkeit der einzelnen Aktivitäten angenommen. Diese Unabhängigkeit ist aber für viele Domänen der Aktivitätserkennung nicht gegeben, beispielsweise erfolgt die Aktivität „frühstücken“ in der Regel nach der Aktivität „sich-waschen“ und nicht in beliebiger Reihenfolge. Deshalb wurden die statistischen Lernverfahren um Methoden erweitert, um diese Abhängigkeiten zu berücksichtigen (für weitere Details s. [89]).

Für die Repräsentation der Aktivitäten wird eine Ontologie definiert, basierend auf der Beschreibungssprache OWL DL (entscheidbare OWL-Variante, DL steht für *Description Logic*, s. Abschnitt 4.1.1). Eine Aktivität wird im Wesentlichen beschrieben durch:

- eine Person, die die Aktion ausführt,
- einen Ort, an dem die Aktion ausgeführt wird,
- Artefakte, die für die Aktion benutzt werden,
- eine Zeitdauer der Aktion.

Zunächst wird mit dem sogenannten DPA-Algorithmus (*Derivation of Possible Activities*) in einer Offline-Phase des Systems festgestellt, welche Aktivitäten potenziell an welchen Orten stattfinden können. Ausgeführt wird der Algorithmus mit Hilfe des DL-Reasoners *RacerPro* [10].

Hybrides statistisch-ontologisches Schließen. Das Verfahren wird an folgendem Beispiel erklärt: eine Person spaziert durch den Wald und führt ihr Mobiltelefon mit sich. Anhand der GPS-Daten kann der Ort bestimmt werden: hier repräsentiert durch das Konzept `Wald`. Ebenfalls kann aus den GPS-Daten eine Bewegungsform geschlossen werden, repräsentiert durch die Instanz i, z.B. „langsame Abwärtsbewegung“. Das statistische Modell stellt nun Konfidenzwerte für die verschiedenen Aktivitäten zur Verfügung, die in der Ontologie repräsentiert sind. Die höchsten Werte bekommen in diesem Fall `hikingDown` (0.16), `strolling` (0.39) und `walkingDownstairs` (0.45). Durch einen Abgleich mit den Ergebnissen des DPA-Algorithmus kann `walkingDownstairs` als Aktivität ausgeschlossen werden, da sie im Ort `Wald` nicht möglich ist. Daher wird `strolling`,

mit dem nächsthöheren Konfidenzwert, als wahrscheinlichste Aktivität ausgewählt und damit richtig erkannt.

Das Interessante an diesem Ansatz ist, dass ontologisches Schließen zunächst in einer Offline-Phase genutzt wird (DPA-Algorithmus), um in Kombination mit statistischen Modellen primitive Aktivitäten abzuleiten. Basierend auf diesen primitiven Aktivitäten werden dann, durch ontologisches Schließen in der Online-Phase, höhere Aktivitäten erkannt. Aktivitäten sind also in einer partonomischen Hierarchie strukturiert. Es gibt jedoch keinen Mechanismus, statistische Modelle auch für höhere Aktivitäten einzubeziehen. Außerdem wird kein Lösungskonzept für eine inkrementelle Interpretation angeboten; das Integrieren von Methoden zur Verarbeitung zeitlicher Constraints und weiterer probabilistischer Verfahren werden als künftige Ziele erwähnt.

2.3.2 Abduktion

Ein anderer, ebenfalls logikbasierter Ansatz der die Abduktion in den Vordergrund stellt wurde von Cohn, Magee, Galata, Hogg und Hazarika [41] beschrieben. Das bedeutet, es wird nach komplexen Konzepten gesucht, deren Instanziierung die vorhandenen Evidenzen „erklärt“. Demnach besteht das Problem der Szeneninterpretation aus Evidenzen Γ, die durch Sensoren detektiert werden, die interpretiert werden müssen durch eine Wissensbasis, welche typischerweise aus zwei Kategorien besteht: allgemeinem domänenunabhängigen Wissen Σ, und spezifischem domänenabhängigen Wissen ϕ. Das Problem ist nun, die Evidenz Γ mit dem gegebenen Wissen zu erklären, mit anderen Worten, vom logischen Standpunkt aus eine Erklärung Δ zu suchen, welche die folgende Formel erfüllt:

$$(\Sigma \cup \phi) \wedge \Delta \models \Gamma . \qquad (2\text{-}5)$$

Neben der kompositionellen Hierarchie für primitive und zusammengesetzte Objekte, definiert in Σ und ϕ, deren Blätter physikalischen Objekten entsprechen, gibt es Wissen für qualitatives räumlich-zeitliches Schließen (QRZS), welches ebenfalls in Σ und ϕ definiert wird und u.a. auf dem *Region Connection Calculus* (RCC) basierte (s. [39]). Auf diese Weise kann gewährleistet werden, dass die Ausgabe des Systems hinsichtlich des QRZS logisch konsistent ist. Die höhere Interpretationskomponente beruht auf drei wesentlichen Mechanismen:

- Durch qualitatives räumlich-zeitliches Schließen, unter Benutzung von Σ und ϕ, werden Inferenzen und Konsistenzchecks ausgeführt, basierend auf z.B. *kontinuierlichen Bewegungen*.

- Eine Abduktionsmaschine generiert Hypothesen, also mögliche (partielle) Erklärungen, basierend auf ϕ, Γ und Δ.

- Einer Komponente für die Handhabung unsicherer Informationen. Die Verhaltensmodelle in ϕ können probabilistisches oder anderes Metawissen enthalten, so dass bei Mehrdeutigkeit, im Falle von verschiedenen möglichen Erklärungen, die wahrscheinlichste gewählt wird.

Ansätze dieser Arbeit, sowie die Arbeiten von Neumann und Möller [82] wurden in das EU-Projekt *CogVis*[3] (Cognitive Vision Systems) integriert und anhand von Tischdeckszenen experimentell untersucht. Ziel des Projekts war die Entwicklung eines generischen, anwendungsunabhängigen Rahmenwerks für die höhere Bilddeutung.

Die Arbeit von Shanahan [98] gründet sich ebenfalls auf Interpretation durch Abduktion. Sie präsentierte eine formale Theorie der Roboterwahrnehmung als Form der Abduktion:

$$\Sigma \wedge \Delta \models \Gamma, \tag{2-6}$$

mit der Wissensbasis Σ, den Beobachtungen Γ und möglichen Erklärungen Δ (entsprechend der Formel 2-5), nur umfasst hier Σ das domänenabhängige und domänenunabhängige Wissen). Zielsetzung ist die Entwicklung eines allgemeinen Rahmenwerks, welches niedere Sensordaten in symbolische Repräsentationen der externen Welt transformiert und auch Aspekte wie Unvollständigkeit und Unsicherheit der Daten, top-down Informationsfluss, aktive Wahrnehmung und Sensorfusion integriert.

Unvollständigkeit und Unsicherheit. Unsicherheit entsteht durch Rauschen und Störungen der Sensoren und ist aufgrund physikalischer Charakteristiken unvermeidbar. Unvollständigkeit resultiert einerseits aus der Tatsache, dass ein Roboter immer nur ein kleines Fenster der realen Welt wahrnehmen kann und jeder Sensor seine Limitationen hat, andererseits kann nicht jeder Aspekt der realen Welt modelliert werden, sondern vorwiegend jene, die für die entsprechenden Ziele und Fähigkeiten des Roboters von Bedeutung sind. Durch sorgfältige Formulierung der Wissensbasis können diese Bedingungen berücksichtigt werden, z.B. indem

[3] Vertrag IST-2000-29375, Projekt CogVis.

bei Constraints bestimmte Bereiche zulässig sind und nicht die Existenz aller Objekte gefordert wird.

Top-down Informationsfluss. Die kognitive Psychologie hat seit langem erkannt, dass der Informationsfluss zwischen Wahrnehmung und Kognition in beide Richtungen verläuft (s. z.B. [36]), und es ist allgemein anerkannt, dass daher eine Kombination von top-down und bottom-up Verarbeitung in Bildanalysesystemen erforderlich ist. Ein Beispiel dafür zeigt Abbildung 4: Zunächst erkennen die meisten Menschen auf dem Bild nur ein paar schwarze Kleckse auf weißem Hintergrund. Wenn ihnen jedoch gesagt wird, dass das Bild einen Hund darstellt, dann erkennen sie plötzlich den Dalmatiner, der im Laub schnüffelt. In diesem Fall wird durch linguistisch erworbenes Wissen die Wahrnehmung eines Objekts ermöglicht; mit anderen Worten: höhere kognitive Prozesse beeinflussen niedere Wahrnehmungsprozesse.

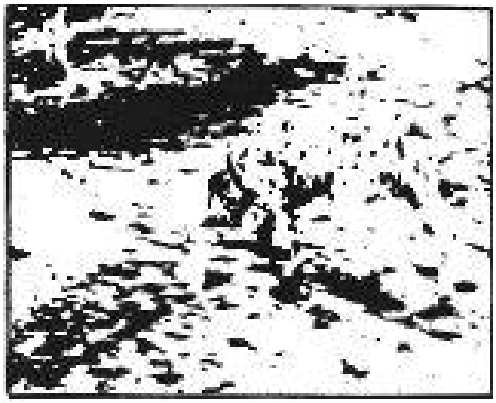


Abbildung 4: Beispiel für top-down Informationsfluss bei visueller Wahrnehmung.

Der top-down Informationsfluss wird in Form von *Erwartungen* realisiert. Angenommen, es wird eine Menge von möglichen Erklärungen Δs gefunden, welche die Beobachtung Γ (partiell) erklären, dann beinhalten diese Δs normalerweise auch immer Beobachtungen bzw. Evidenzen, die nicht in Γ enthalten sind, die also nicht auf Sensordaten basieren; das sind die Erwartungen. Diese Erwartungen können dann neu parametrisierte Bildverarbeitungsberechnungen oder den Einsatz weiterer Sensoren anstoßen, um die Erwartungen zu bestätigen oder zu verwerfen.

Aktive Wahrnehmung. Durch die oben beschriebene top-down Verarbeitung und die Generierung von Erwartungen können Aktionen aktiviert werden, um die Erwartungen zu bestätigen oder zurückzuweisen. Zum Beispiel wird die Kamera des Roboters neu positioniert, so dass Objekte beobachtet werden können, die vorher verdeckt waren. Das wird als aktive Wahrnehmung bezeichnet.

Sensorfusion. Unter Sensorfusion wird das Problem der Kombination von (möglicherweise in Konflikt stehenden) Daten unterschiedlicher Quellen verstanden. Die niederen Eingabedaten in Γ können aus verschiedenen Sensormodalitäten stammen oder vom selben Sensor, aber mit unterschiedlichen Verarbeitungsparametern oder zu einem anderen Zeitpunkt extrahiert. Um diesem Problem gerecht zu werden, wird versucht, die Wissensbasis Σ mit Formeln zu erweitern, die Rauschen und andere Abnormalitäten beinhalten. Auf diese Weise können die Erscheinungen erklärt werden. Ein Präferenzmaß priorisiert die Erklärungen Δ, die mit den wenigsten dieser Rausch- und Abnormalitätsformeln auskommen.

Wie bereits in der Arbeit von Cohn et al. [41] beschrieben, ist auch in dieser Arbeit eine Intention, möglichst umfangreiches Wissen, z.B. auch über Rauschen und andere Abnormalitäten, in der Wissensbasis Σ zu verankern und die Bildverarbeitung auf die Detektion einfacher geometrischer Formen und Regionen zu beschränken:

> *„In an ideal implementation of the theory, Σ would incorporate a full axiomatization of naive physics, either hand coded or acquired by a compatible learning method, such as inductive logic programming"* (Shanahan, 2005).

Eine Arbeit zum Interpretieren von Multimedia-Daten, die auf Abduktion und den sogenannten *Markov-Logik-Netzwerken* (MLN) [90] basiert, wurde von der Gruppe von Möller [55] vorgestellt.

Es ist seit langem ein Ziel in der Forschung der Künstlichen Intelligenz, Wahrscheinlichkeit und Prädikatenlogik erster Ordnung in einer einzigen Repräsentationsform zu kombinieren. Ein probabilistisches Modell eröffnet die Möglichkeit, effizient unsicheres Wissen zu behandeln, während die Prädikatenlogik erster Ordnung und Beschreibungslogiken vielfältiges Wissen in kompakter Weise repräsentieren können. Viele Anwendungen aus dem Bereich der Künstlichen Intelligenz erfordern beides.

Eine Repräsentationsform, die dieses Ziel verfolgt, sind MLN. Eine Wissensbasis, repräsentiert mit Prädikatenlogik erster Ordnung oder mit Beschreibungslogiken, kann als Menge von harten Constraints ansehen werden: Wenn ein Modell auch nur eine Formel verletzt, ist seine Wahrscheinlichkeit null. Die wesentliche Idee der MLN ist es, diese Constraints aufzuweichen: Wenn ein Modell eine Formel der Wissensbasis verletzt, dann ist es weniger wahrscheinlich, aber nicht unmöglich. Je weniger Formeln das Modell verletzt, desto wahrscheinlicher ist es. Jeder Formel wird ein Gewicht zugewiesen, welches die Strenge des Constraints repräsentiert: je größer das Gewicht, umso größer ist die Differenz der Wahrscheinlichkeit zwischen einem Modell, das die Formel erfüllt, und einem, das sie nicht erfüllt.

Wie bereits in Abschnitt 1.2.5 beschrieben, ist eine Anwendung zum Interpretieren von Multimedia-Daten die Suche nach Informationen im Internet oder in Datenbanken, beschrieben auf hohem Abstraktionsniveau. Bilder bestehen aus primitiven Objekten, welche die Grundlage für die konzeptuelle Wissensbasis bilden; diese können sowohl visueller als auch textueller Natur sein. Das Erkennen der primitiven Objekte ist eine niedere Bildverarbeitungsaufgabe und steht hier nicht im Fokus.

Wie schon in den obigen Arbeiten zur Abduktion beschrieben, ist auch in diesem Ansatz die Grundformel:

$$\Sigma \wedge \Delta \models \Gamma, \tag{2-7}$$

mit der Wissensbasis Σ, Beobachtungen Γ und gesuchten Erklärungen Δ. Um die gesuchten Erklärungen zu berechnen, wird auch in dieser Arbeit RacerPro verwendet und um Abduktionsmechanismen erweitert. Auf Basis dieser neuen Abduktionsdienste können nun entsprechende Hypothesen generiert werden, die die Beobachtungen erklären. Auch hier tritt dabei das Problem auf, dass es in der Regel mehrere mögliche Hypothesen gibt. Um ein probabilistisches Präferenzmaß zu realisieren, welches diese Hypothesen bewertet, wird das Konzept der MLN auf Beschreibungslogiken spezialisiert, um es auf diese Weise mit ABoxen und TBoxen benutzen und damit in RacerPro integrieren zu können.

2.3.3 Fazit der logikbasierten Ansätze

Die Untersuchungen der logikbasierten Ansätze zur Szeneninterpretation haben zu wenigstens zwei wichtigen Erkenntnissen geführt:

- Beschreibungslogiken bieten die Möglichkeit, taxonomische und partonomische Konzepthierarchien in objektzentrierter Weise zu repräsentieren. Weiterhin bieten sie nützliche Inferenzmöglichkeiten, wie z.B. Vererbung und Klassifikation. Andererseits ist es schwierig, Constraints zwischen Objekten und Ereignissen zu beschreiben, welche in der Regel maßgebend für die Definition und Erkennung von komplexen Konzepten sind. Außerdem fehlen entscheidende Funktionalitäten für ein generisches Rahmenwerk, welches eine flexible, schrittweise Interpretation ermöglicht, wie sie oftmals in komplexen Anwendungen gefordert wird.

- Vom logischen Standpunkt aus gesehen ist die Szeneninterpretation von Natur aus mehrdeutig. Evidenzen, die von der niederen Bildverarbeitung oder durch diverse Sensoren geliefert werden, können oft zu verschiedenen alternativen Instanziierungen führen. Unsichere und unvollständige Daten der Bildverar-

beitung bzw. Sensoren können dabei in komplexen Anwendungen nicht vermieden werden und verschärfen das Problem der Mehrdeutigkeit zusätzlich. Die Mehrdeutigkeit ist überdies nicht auf primitive Evidenzen beschränkt; in partonomischen Hierarchien kann die Ambiguität auch in höheren Ebenen auftreten. Außerdem verstärkt eine schrittweise Interpretation – wie sie für Realzeitanwendungen gefordert wird – das Problem der Mehrdeutigkeit: im Anfangsstadium der sich entwickelnden Szene müssen Interpretationsentscheidungen auf der Grundlage von spärlichem Kontext getroffen werden.

2.4 Probabilistische Ansätze

Die Arbeiten zur logikbasierten Szeneninterpretation zeigen die Notwendigkeit eines Präferenzmaßes, welches bei Mehrdeutigkeiten als Entscheidungshilfe herangezogen werden kann. Menschen lösen derartige Konflikte offenbar dadurch, dass sie sämtliche zur Verfügung stehenden Informationen der Domäne und der aktuellen Situation auswerten, um dann die wahrscheinlichste oder plausibelste Alternative auszuwählen. Es liegt daher nahe, *probabilistische* Präferenzmaße für die Szeneninterpretation einzusetzen (s. [79]). Unabhängig von der durch die Logik motivierten Notwendigkeit probabilistischer Präferenzmaße, wurden probabilistische Ansätze für die Szeneninterpretation entwickelt. Dies begründet sich u.a. durch die Erfolge von probabilistischen Methoden in den Bereichen des maschinellen Sehens und der Robotik.

2.4.1 Bayes-Netze

In einer frühen Arbeit von Binford, Levitt und Mann [30] wurden Bayes-Netze für die höhere Bildanalyse verwendet, um kompositionelle Hierarchien von Objekten zu repräsentieren, analog zu den in Abschnitt 2.3.1 beschriebenen Aggregaten. Diese Hierarchien formen einen gerichteten, azyklischen Graphen, dessen Knoten Teile und dessen Kanten Relationen repräsentieren. Die Kantenrichtung kann hierbei als „verursachen“ interpretiert werden, in dem Sinne, dass ein Elternteil seine Kinderteile „verursacht“. Umgekehrt stützen die Teile die Existenz des Ganzen (s. Abbildung 5). Eine derartige Repräsentation ist intuitiv sinnvoll und mit der Bayes-Theorie als Grundlage mathematisch konsistent. Jedoch impliziert sie bei gegebenem Elternteil die *statistische Unabhängigkeit* der Teile, das bedeutet, dass sich die Teile nicht gegenseitig beeinflussen. Mit anderen Worten, die Wahrscheinlichkeit bzw. die Eigenschaften eines Teils (hinsichtlich Existenz, Ort oder anderer Eigenschaften), wirkt sich, bei gegebenen Aggregateigenschaften, nicht

auf die Wahrscheinlichkeit bzw. die Wahrscheinlichkeitsverteilung eines anderen Teils aus. Diese Unabhängigkeit ist aber für viele Applikationen nicht gegeben.

Betrachtet man beispielsweise das Aggregat `Gedeck` in Abbildung 5 und nimmt an, die Existenz von `Gedeck` sei gegeben, ebenso seine Position in Form eines umgebenden Rechtecks, dann können die Positionen der Teile durch Wahrscheinlichkeitsverteilungen beschrieben werden. Nun beeinflusst aber die Position des Tellers die Positionen von Messer und Gabel, ebenso bedingen sich die Positionen von Untertasse und Tasse. Die Teile sind bei einem gegebenen Gedeck demzufolge nicht statistisch unabhängig voneinander. Gleiches gilt für die Modellierung von Ereignissen durch Teilereignisse: Hier sind, bei gegebenen Zeitpunkten des Gesamtereignisses, die Teile oftmals durch zeitliche Constraints miteinander verknüpft, die sich gegenseitig beeinflussen.

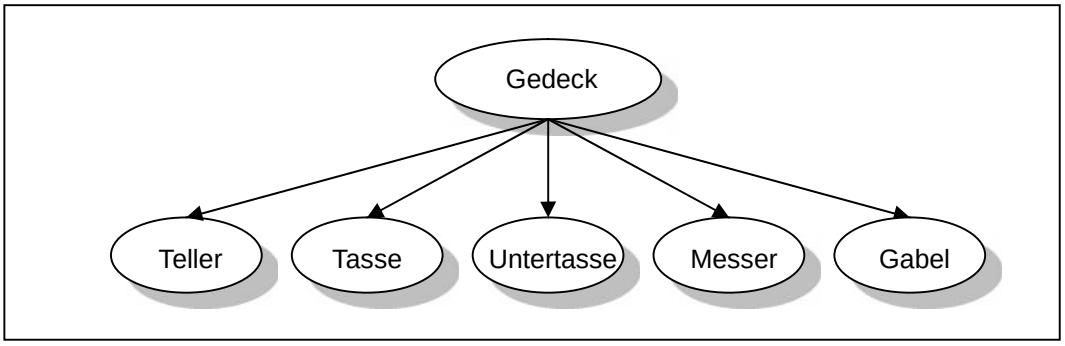


Abbildung 5: Repräsentation des Aggregats `Gedeck` als Bayes-Netz.

In seiner Dissertation von 1993 beschreibt Rimey [91] die Verwendung von Bayes-Netzen und Entscheidungstheorie für die höhere, selektive Bildanalyse. Motiviert wurde der Einsatz höherer Bilddeutung zur Realisierung von aktiven Interpretationssystemen, die ihre Sensoren kontrollieren (gleichbedeutend mit der bereits in Abschnitt 2.3.2 angesprochenen aktiven Wahrnehmung); so könnte beispielweise ein Roboter die Position seiner Kamera ändern, um ein (teilweise) verdecktes Objekt zu erkennen.

Rimey benutzt Bayes-Netze zur kompositionellen und taxonomischen Repräsentation von Objekthierarchien in der von Binford et al. [30] beschriebenen Weise. Sie dienen ausschließlich zur Berechnung der Wahrscheinlichkeiten der Existenz der Objekte. Eine wichtige Anforderung für ein Bildanalysesystem ist jedoch das Modellieren und Schließen von räumlichen Relationen und Positionen von Objekten in einer Szene. Deshalb führt er zusätzlich ein sogenanntes *Expected-Area-Net* (EAN) ein, welches die gleiche Struktur hat, wie das partonomische Netz (das Problem der statistischen Unabhängigkeit gilt also auch für das EAN). Eine Verbindung von Knoten A und B im EAN spezifiziert eine räumliche Relation Zwi-

schen den Objekten A und B, und durch eine Wahrscheinlichkeitsverteilung wird der Ort von B relativ zur Größe und dem Ort von A modelliert (geliefert durch einen Wissensingenieur). Darüber hinaus gibt es noch ein *Task-Net*, in welchem kodiert ist, welche Teile der Szene jeweils für die Lösung einer Aufgabe wichtig sind, und ein *Composite-Net* zur Verknüpfung und Koordination der übrigen Netze.

Der Bildanalyseprozess wird durch spezifizierte Aktionen (Tasks) getriggert. Dabei kann es sich beispielsweise um das Detektieren und Lokalisieren bestimmter Objekte handeln oder das Erkunden bestimmter Eigenschaften von bereits lokalisierten Objekten. Die Aktionsfunktionen bekommen als Eingabe unter anderem die aktuelle Kameraposition, Informationen über die zu erwartende Region, in der sich das Objekt befindet, und die Rohbilddaten. Bildverarbeitungsprozesse berechnen dann Evidenzen aus den gelieferten Rohbilddaten. Die Aktionsfunktionen enthalten jeweils einen Satz von Regeln, welche die Evidenzen spezifischen Stellen im Composite-Net zuordnen, wo sie dann durch entsprechende Propagationen verarbeitet werden.

2.4.2 Hierarchische Bayes-Netze

Um die Ausdruckskraft herkömmlicher Bayes-Netze zu erhöhen und eine bessere Integration der Aggregatstruktur zu ermöglichen, welche für viele Anwendungen essentiell ist, entwickelten Gyftodimos und Flach [61] *hierarchische Bayes-Netze* (HBN). Sie stellen eine Generalisierung von Bayes-Netzen dar, bei der ein Knoten des Netzes ein Aggregat repräsentieren kann. Innerhalb eines solchen Knotens können Verbindungen zwischen den Teilen des Aggregats bestehen und probabilistische Abhängigkeiten zwischen ihnen ausgedrückt werden. Zwischen den Knoten repräsentieren HBN bedingte und unbedingte probabilistische Abhängigkeiten in der gleichen Weise wie Bayes-Netze. HBN bestehen aus zwei Komponenten:

- Die *strukturelle Komponente* beschreibt die `hat-Teil`-Relationen und die probabilistischen Abhängigkeiten zwischen den Teilen.
- Die *probabilistische Komponente* beschreibt die probabilistischen Abhängigkeiten im Sinne des Bayes-Netzes.

Abbildung 6 demonstriert die Struktur eines HBN anhand des Beispiels *Golfspielen*, dessen Variable die Wahrscheinlichkeit ausdrückt, dass ein bestimmter Tag für eine Person geeignet ist, ihrem Hobby nachzugehen. Die Entscheidung wird beeinflusst durch Wetterverhältnisse und die Stimmung der Person, welche von geschäftlichen Belangen abhängt. Abbildung 6(a) zeigt den strukturellen Teil des herkömmlichen Bayes-Netzes. Nun werden die Zufallsvariablen `Vorhersage,`

`Wind` und `Temperatur` zum Aggregat `Wetter` zusammengefasst, `Aktienkurs` und `Aktien` zu `Markt`, `Markt` und `Meeting` zu `Geschäft` und schließlich `Wetter`, `Golfspielen` und `Geschäft` zu `t`. Abbildung 6(b) zeigt das resultierende HBN; die gestrichelten Linien stellen die Aggregatstruktur dar.

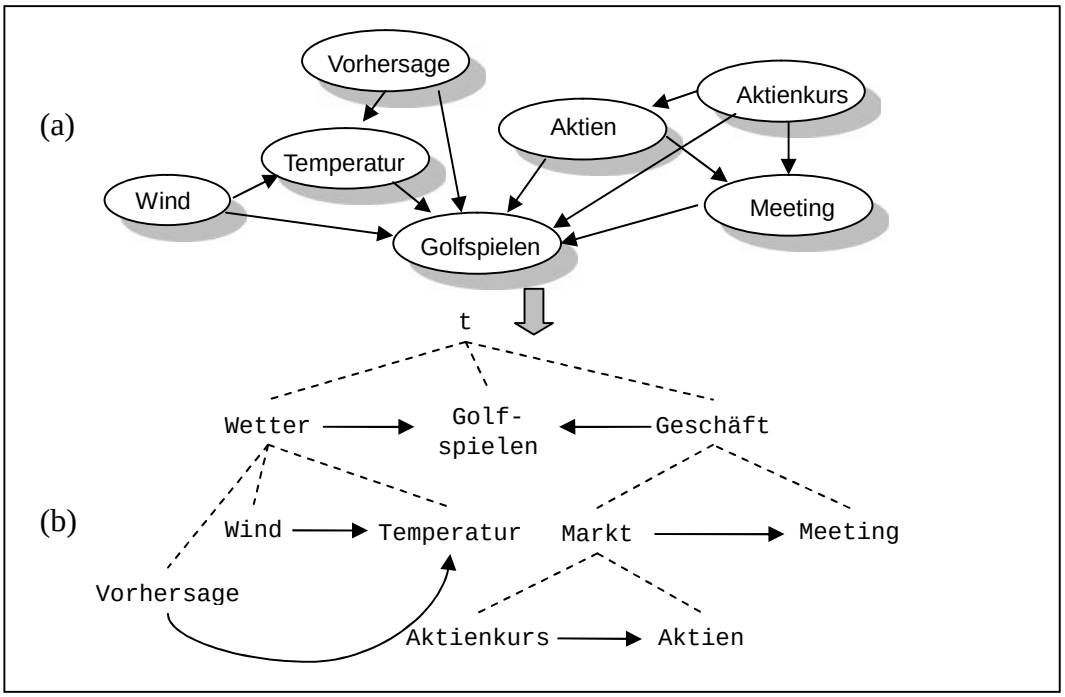


Abbildung 6: (a) Bayes-Netz. (b) Hierarchisches Bayes-Netz.

2.4.3 Bayes'sche Kompositionelle Hierarchien

Ähnlich zu den hierarchischen Bayes-Netzen, entwickelte Neumann *Bayes'sche Kompositionelle Hierarchien* (BCHs) [79], die speziell für die Anforderungen der Szeneninterpretation angepasst sind. Eine BCH ist ein probabilistisches Modell einer partonomischen Aggregathierarchie, welche die kompositionelle Struktur von Objekten oder Ereignissen repräsentiert. Die Wurzel besteht demzufolge aus der gesamten Szene und die Blätter repräsentieren primitive Objekte oder Ereignisse, die nicht weiter zerlegt werden können und die als Evidenzen von der Mittelschicht geliefert werden (s. Abbildung 1). Die schematische Darstellung einer BCH ist in Abbildung 7 illustriert.

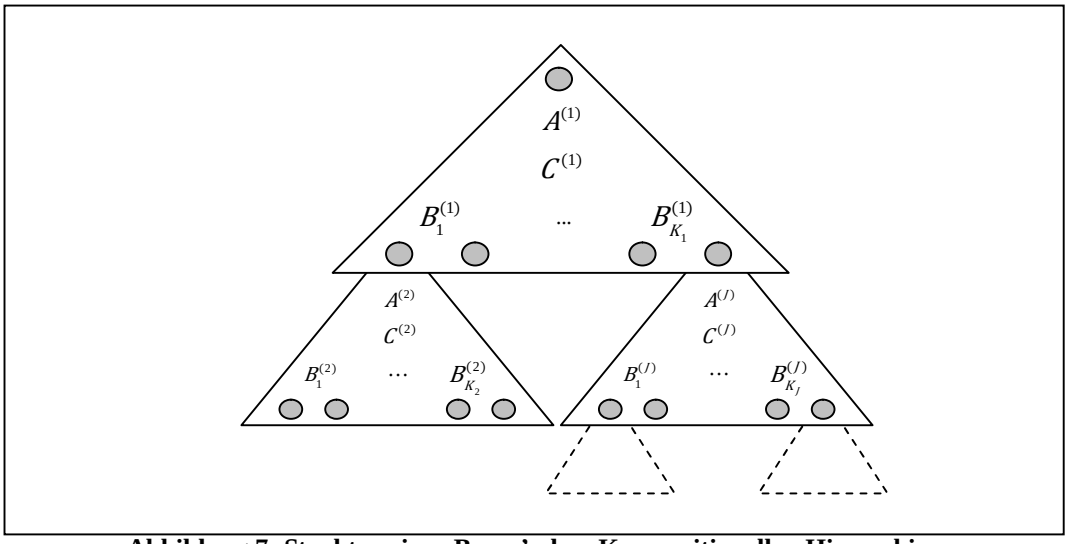


Abbildung 7: Struktur einer Bayes'schen Kompositionellen Hierarchie.

Jedes Aggregat – dargestellt durch ein Dreieck – wird beschrieben durch eine gemeinsame Wahrscheinlichkeitsverteilung $P(A,B_1,...,B_k,C)$. Dabei steht A für den Aggregatkopf, der eine externe Beschreibung für die nächst höhere Ebene bereitstellt, $B_1,...,B_k$ beschreibt die Teile, und C steht für zusätzliche interne Parameter, welche Constraints zwischen den Teilen ausdrücken, z.B. Distanzen, Größen oder zeitliche Constraints. Die Hierarchie ist in der Weise konstruiert, dass die externen Beschreibungen $A^{(2)}$ bis $A^{(J)}$ der Aggregatköpfe, die von Details der Teile abstrahieren, als Beschreibungen der Teile der nächst höheren Ebene verwendet werden, d.h. $B_1^{(1)} = A^{(2)}$ usw. Auf BCHs und deren Arbeitsweise wird im weiteren Verlauf dieser Arbeit noch detaillierter eingegangen (s. Abschnitt 5.4).

2.4.4 Stochastische Grammatiken

Eine alternative Methode zur höheren Bilddeutung, die ebenfalls häufig in der Literatur diskutiert wird, ist die Verwendung von *Grammatiken* für die höhere Bildanalyse. Ein interessanter Ansatz, der Grammatiken mit probabilistische Methoden kombiniert, wurde von Mumford and Zhu [77] präsentiert. In dieser Arbeit übernehmen grammatikalische Formalismen die Aufgabe der hierarchischen Wissensrepräsentation, und probabilistische Parsing-Algorithmen werden für den Interpretationsprozess verwendet.

Wissensrepräsentation mit Grammatiken und Graphen. Zur Repräsentation hierarchischer Objektstrukturen werden *Und-Oder-Graphen* verwendet **(**s. Abbildung 8**)**. Sie enthalten drei verschiedene Arten von Knoten:

- Und-Knoten repräsentieren die Komposition von Objekten und ihren Teilen. Sie korrespondieren beispielsweise zu folgenden Grammatikregeln: $A \rightarrow BCD$, $H \rightarrow NO$.

- Die horizontalen Verbindungen repräsentieren Relationen und Constraints zwischen den Teilen. Bei der Verarbeitung statischer Bilder sind dies z.B. räumliche oder Identitäts-Constraints.

- Oder-Knoten repräsentieren alternative Substrukturen. Sie korrespondieren zu Produktionsregeln wie: $B \rightarrow E \mid F$, $C \rightarrow G \mid H \mid I$.

- Blattknoten (Terminalknoten) repräsentieren die primitiven Bildteile.

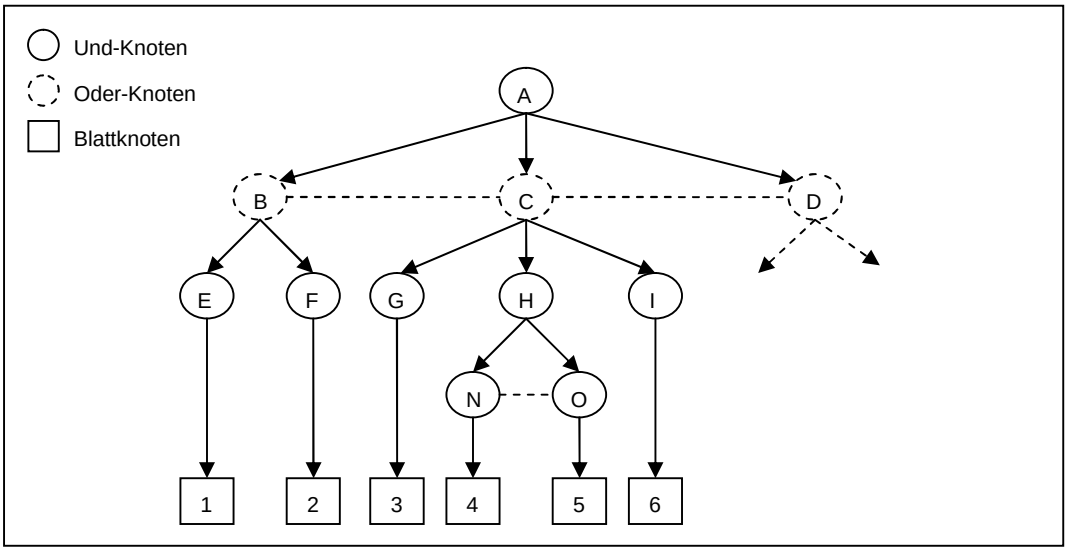


Abbildung 8: Und-Oder-Graph.

Auf diese Weise können alle Objektkategorien der Szene durch einen großen Und-Oder-Graphen repräsentiert werden. Auch rekursive Definitionen und gemeinsame Teile sind zulässig. Ein *Analysegraph* stellt eine Interpretation eines spezifischen Bildes dar. Er wird aus dem Und-Oder-Graphen abgeleitet, indem Alternativen der Oder-Knoten derart gewählt werden, dass die erzeugten Blattknoten mit den Instanzen der primitiven Bildteile übereinstimmen. Die primitiven Bildteile erben dabei die Relationen der Vorfahrenknoten, aus denen sie erzeugt wurden. Ein Analysegraph erzeugt so einen planaren Graphen, bestehend aus Blattknoten,

die mit primitiven Bildteilen assoziiert sind und Relationen zwischen ihnen. Diese Graphen werden *Konfigurationen* genannt. Eine Konfiguration repräsentiert also eine mögliche Interpretation des Bildes. Die *Sprache* einer Grammatik ist die Menge aller möglichen gültigen Konfigurationen, die durch die Grammatik produziert werden können.

Bei einer *stochastischen kontextfreien Grammatik* ist jede Produktionsregel mit einer bestimmten Wahrscheinlichkeit assoziiert. Sei $A \in V_N$ eine Menge von alternativen Produktionsregeln:

$$A \to \beta_1 | \beta_2 | ... | \beta_{n(A)}, \qquad \gamma_i : A \to \beta_i . \tag{2-8}$$

Dann wird jeder Produktionsregel eine Wahrscheinlichkeit $p(\gamma_i) = p(A \to \beta_i)$ zugeordnet, so dass gilt:

$$\sum_{i=1}^{n(A)} p(A \to \beta_i) = 1 . \tag{2-9}$$

In der Arbeit von Mumford and Zhu wird das Konzept der stochastischen Grammatiken auf Und-Oder-Graphen erweitert und führt zu einer *stochastischen kontextsensitiven Grammatik* (durch die Integration von Relationen und Constraints) für Bilder (für nähere Details s. [77]).

Parsing-Algorithmus. Der Interpretationsprozess hat hier die Aufgabe, Analysegraphen aus Eingabebildern zu berechnen, welche verschiedene Interpretationen repräsentieren. Dazu werden zwei Listen verwaltet: eine *offene Liste* und eine *geschlossene Liste*. Ein bottom-up Prozess generiert Einträge für die offene Liste auf zwei Arten:

- durch das Erzeugen von primitiven Evidenzen (Terminalsymbolen),
- durch das Erzeugen von Hypothesen (Nichtterminalsymbolen), wobei deren Teile partiell den existierenden Einträgen der offenen und geschlossen Liste zugeordnet, und deren Constraints getestet werden.

Der top-down Prozess validiert die Hypothesen durch die Berechnung der A-posteriori-Wahrscheinlichkeiten. Durch einen Greedy-Algorithmus wird diejenige Hypothese akzeptiert, welche die A-posteriori-Wahrscheinlichkeit maximiert. Eine akzeptierte Hypothese wird in die geschlossene Liste verschoben. Durch abwechselnde bottom-up und top-down Schritte wird eine Interpretation generiert.

2.4.5 Fazit der probabilistischen Ansätze

Die reale Welt ist komplex, und ein Szeneninterpretationssystem, welches den Anspruch erhebt, für Anwendungen der realen Welt geeignet zu sein, muss in der Lage sein

- mit unsicheren und unvollständigen Eingabedaten umzugehen,
- ein Präferenzmaß für die inhärente Mehrdeutigkeit bereitzustellen,
- Vorhersagen über den weiteren Verlauf der Szene zu treffen.

In der jüngeren Vergangenheit wurden u.a. probabilistische Methoden verwendet, um Systeme zu entwickeln, die unsicheres Wissen verarbeiten können. Erfolge probabilistischer Verfahren aus den Bereichen Computersehen und Robotik legen nahe, probabilistische Methoden auch in der höheren Bilddeutung einzusetzen. Die beschriebenen probabilistischen Ansätze belegen diesen Trend.

Eine besondere Bedeutung bei den probabilistischen Methoden nehmen Bayes-Netze ein. Mit Hilfe von Bayes-Netzen kann unsicheres Wissen in Form von gerichteten azyklischen Graphen dargestellt werden. Ein Bayes-Netz dient dazu, die gemeinsame Wahrscheinlichkeitsverteilung *aller* beteiligten Variablen unter Ausnutzung bekannten bedingter Unabhängigkeiten möglichst kompakt zu repräsentieren. Sie ermöglichen das Schließen unter Unsicherheit, sowohl für die Vorhersage als auch für die Diagnose, mit dem gleichen Inferenzprozess, der auf dem Bayes'schen Theorem basiert. Weiterhin können Effekte von Veränderungen – z.B. durch die Verfügbarkeit neuer Evidenz – effizient durch das Netz propagiert werden. Bei einer Vielzahl existierender Anwendungen gibt es drei primäre Gründe für den Einsatz von Bayes-Netzen (s.[34], S. 71):

- Die Inferenzalgorithmen sind effizient und ermöglichen Echtzeitanwendungen.
- Wissen kann verständlich und modular modelliert werden.
- Bayes-Netzes können durch Lernverfahren generiert werden. Das ist besonders wichtig für Bereiche, in denen nicht genügend Informationen durch Experten bereitgestellt werden können.

Diese Eigenschaften zeichnen Bayes-Netze prinzipiell auch als interessante Kandidaten für den Einsatz in der Szeneninterpretation aus. Jedoch unterstützen Bayes-Netze nicht in angemessener Weise die kompositionelle Aggregatstruktur, welche für die Szeneninterpretation wesentlich ist. Daher wurden Hierarchische Bayes-Netze (HBNs) [61] und Bayes'sche Kompositionelle Hierarchien (BCHs)

[79] entwickelt, welche auf Bayes-Netzen basieren und an die speziellen Anforderungen der Szeneninterpretation angepasst sind.

Stochastische Grammatiken sind ein mächtiges Werkzeug für die Realisierung von Systemen zur höheren Bilddeutung, wie die Arbeit von Mumford und Zhu [77] zeigt. Jedoch können die partonomischen und taxonomischen Konzepthierarchien nicht in der transparenten und objektzentrierten Weise repräsentiert werden, wie das mit Beschreibungslogiken möglich ist. Vielmehr werden die Modelle durch Grammatikregeln beschrieben, und diese können für eine komplexe Wissensbasis schnell unübersichtlich werden. Darüber hinaus gilt auch für Grammatikregeln, dass sich Constraints nicht in natürlicher Weise ausdrücken lassen, sondern durch ergänzende Mechanismen realisiert werden müssen, wie z.B. durch zusätzliche Eigenschaften der Und-Oder-Graphen (s. [77]).

2.5 Regelbasierte Ansätze

Regelbasierte Systeme wurden schon in den 70er und 80er Jahren für die Entwicklung von Expertensystemen eingesetzt. Sie repräsentieren Domänenwissen in einer sehr natürlichen Form mit Hilfe von *Wenn-dann-Regeln* (s. Abschnitt 4.3).

Ein regelbasierter Ansatz für komplexe Ereignisverarbeitung wurde kürzlich von Anicic et al. [25] vorgestellt. Dort wird zwischen *atomaren* und *komplexen* Ereignissen unterschieden. Atomare Ereignis sind punktuell und bilden die Evidenzen, welche als Eingabedaten an das System geliefert werden. Komplexe Ereignisse haben eine zeitliche Ausdehnung und werden mit Hilfe sog. *Ereignismuster* durch atomare Ereignisse definiert. Komplexe Ereignisse können dabei über beliebig viele Ebenen durch andere komplexe Ereignisse definiert werden. Die Aufgabe des Systems ist es, aus einem Eingabestrom von atomaren Ereignissen dynamisch komplexe Ereignisse zu erkennen.

Die Ereignismuster werden durch eine speziell entwickelte Sprache namens *ETALIS* definiert (für nähere Details, siehe [25]). Beispielsweise könnte ein Ereignismuster P folgende Struktur besitzen:

$$P ::= (Price(X, Y_1)\ \texttt{SEQ}\ Price(X, Y_2))\texttt{.7 WHERE}\ Y_2\ \texttt{>}\ Y_1 \cdot \texttt{1.5} \qquad (2\text{-}10)$$

Ziel dieses Ereignismusters ist es, die Preiserhöhung eines Artikels X um 50% in einem Zeitraum von sieben Tagen zu erkennen. Die atomaren Ereignisse $Price(X, Y_1)$ und $Price(X, Y_2)$ müssen dazu nacheinander (ausgedrückt durch das Schlüsselwort `SEQ`) innerhalb von sieben Tagen stattfinden.

Weiterhin gibt es *Ereignisregeln*, die auf der Basis von Ereignismustern definiert werden. Eine entsprechende Ereignisregel könnte für dieses Beispiel folgendermaßen lauten:

$$RemarkableIncrease(X) \leftarrow \quad (2\text{-}11)$$
$$(Price(X, Y_1)\ \texttt{SEQ}\ Price(X, Y_2))\texttt{.7 WHERE}\ Y_2\ \texttt{>}\ Y_1 \cdot \texttt{1.5}$$

Der Erkenntnisprozess verläuft *zielgetrieben*, d.h. durch Rückwärtsverkettung (engl.: *backward chaining*): ausgehend von einem herzuleitenden Fakt – welches ein atomares oder komplexes Ereignis repräsentiert – werden Regeln ermittelt, deren Aktionsteil den Fakt erzeugen würde (s. Abschnitt 4.3). Dies ist die Arbeitsweise der Programmiersprache Prolog, welche für den Prototypen des Systems verwendet wird.

Es kann vorkommen, dass es mehrere Regeln gibt, die einen herzuleitenden Fakt erzeugen. Angenommen es gibt eine Regel:

$$E \leftarrow A\ \texttt{SEQ}\ B, \quad (2\text{-}12)$$

und vier Fakten (Evidenzen), die in folgender Reihenfolge geliefert werden: A_1, A_2, B_1, B_2. Um die Mehrdeutigkeiten aufzulösen, werden drei Strategien diskutiert, in welcher Weise die Fakten „konsumiert“ werden:

- „Neuestes“ – dabei wird den neuesten Fakten die höchste Priorität gegeben. Hier würden die Paare (A_2, B_1) und (A_1, B_2) zum Feuern der Regel führen.
- „Chronologisch“ – dabei werden die Fakten in ihrer chronologischen Reihenfolge abgearbeitet. Hier würden die Paare (A_1, B_1) und (A_2, B_2) zum Feuern der Regel führen.
- „Uneingeschränkt“ – dabei werden alle möglichen Regelausführungen uneingeschränkt zugelassen. Hier würden die Paare (A_1, B_1), (A_2, B_1), (A_1, B_2) und (A_2, B_2) zum Feuern der Regel führen.

Nur die ersten beiden Strategien werden für die Ereigniserkennung als relevant angesehen. Auf die Behandlung unvollständiger und unsicherer Evidenzen wird nicht eingegangen.

2.5.1 Fazit regelbasierter Ansätze

Regelbasierte Systeme eigenen sich gut für die dynamische Verarbeitung von Eingabeströmen. Neue Evidenzen, die durch Fakten repräsentiert werden, können dem Arbeitsspeicher inkrementell hinzugefügt werden. Dadurch wird das Feuern von Regeln getriggert. Weiterhin können sie sowohl datengetrieben als auch zielgetrieben arbeiten (s. Abschnitt 4.3).

Ein bekannter Nachteil regelbasierter Systeme für die höhere Bilddeutung oder Ereigniserkennung ist, dass die Regeln einerseits das konzeptuelle Wissen repräsentieren und andererseits auch den Erkennungsprozess steuern. Die Kontrollstrukturen sind also *nicht* von der Wissensbasis getrennt, wie es in einem generischen Ansatz für die Szeneninterpretation gefordert wird (s. Abschnitt 1.3). Dadurch ist die Flexibilität eingeschränkt und die Übertragung auf eine andere Domäne aufwändig. In dem hier untersuchten Ansatz wäre es notwendig, alle Ereignismuster und Ereignisregeln neu zu erstellen (wobei die bestehenden Interpretationsmechanismen berücksichtigt werden müssten). Weiterhin gibt es keine gegebenen Mechanismen für Konsistenzprüfungen der Wissensbasis und für die Verarbeitung von unsicheren und unvollständigen Informationen (s. [73], [92]).

3 Konzeptioneller Ansatz

Ziel dieser Arbeit ist es, ein generisches Rahmenwerk für ein wissensbasiertes Szeneninterpretationssystem zu entwickeln. Dazu wird in Abschnitt 3.1 die wissenschaftliche Problemstellung erarbeitet, basierend auf den Zielen und Anforderungen aus Abschnitt 1.3 und unter Berücksichtigung der gewonnenen Erkenntnisse der bestehenden Ansätze aus Kapitel 2. In Abschnitt 3.2 wird der grundlegende Ansatz erläutert.

3.1 Problemstellung

Betrachten wir nun erneut die Anforderungen an ein generisches Szeneninterpretationssystems aus Abschnitt 1.3, um unter Berücksichtigung der Erkenntnisse der bestehenden Ansätze Methoden zu identifizieren, welche den Anforderungen (zumindest teilweise) gerecht werden. Daraus kann dann die grundlegende Problemstellung abgeleitet werden:

- Verwendung standardisierter Wissensrepräsentationsmethoden,
- Statische Verarbeitung von Bildern und dynamische Verarbeitung von Videosequenzen,
- Echtzeitfähigkeit,
- Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses,
- Verarbeitung unsicherer und unvollständiger Eingabedaten.

Verwendung standardisierter Wissensrepräsentationsmethoden. Standardisierte Sprachen, wie OWL DL (siehe Abschnitt 4.1.2), die auf Beschreibungslogiken basieren, sind geeignet, die Szeneninterpretation zu unterstützen (s. [82]):

- Taxonomische und partonomische Konzepthierarchien können in objektzentrierter Weise repräsentiert werden. Außerdem bieten DL-Systeme eine Reihe von Services an, welche die Erstellung und Wartung einer Wissensbasis unterstützen, z.B. Konsistenzüberprüfungen. Zusätzlich ist es jedoch notwendig, Constraints zwischen Konzepten auszudrücken. Dies ist mit Beschreibungslogiken nur bedingt möglich.

- Der Interpretationsprozess kann ebenfalls durch diverse Services eines DL-Systems unterstützt werden, z.B. die Spezialisierung einer Instanz oder der Test, ob ein Individuum Instanz eines Konzepts ist. Generell kann ein DL-System die Konsistenz der ABox hinsichtlich der TBox überprüfen, demzufolge entspricht eine konsistente ABox (partiell) einem Modell der TBox und damit einer (partiellen) Szeneninterpretation. Diese Eigenschaften von Beschreibungslogiken können für die Generierung konsistenter Szeneninterpretationen genutzt werden (s. Kapitel 4);

- Wie bereits erwähnt, ist die Szeneninterpretation von Natur aus mehrdeutig. Dies äußert sich in der Verwendung von Beschreibungslogiken dadurch, dass es mehrere Modelle gibt, für die die ABox konsistent ist. Von einem DL-System gibt es in diesem Fall keine Unterstützung für die Auswahl eines bestimmten Modells bzw. für eine Strategie zur Wahl einer bestimmten Alternative zur Generierung eines Teilmodells.

Grammatiken, regelbasierte Systeme und Bayes-Netz-basierte Methoden können ebenfalls mit Erfolg in der Szeneninterpretation eingesetzt werden, wie die vorgestellten Arbeiten aus Kapitel 2 gezeigt haben. Allerdings ist die Repräsentation der Konzepthierarchien nicht in gleichem Maße transparent und objektzentriert, wie dies bei Beschreibungslogiken der Fall ist. Weiterhin sind Methoden zur Konsistenzprüfung der Wissensbasis nicht Bestandteil dieser Repräsentationsmethoden. Es ist ein häufig beschriebener Nachteil von regelbasierten Systemen, dass es schwierig ist, eine umfangreiche Regelbasis zu entwerfen, die konsistent ist und alle spezifizierten Aufgaben erfüllt (s. [71], S. 231). Das Problem der Mehrdeutigkeit besteht bei Grammatiken und regelbasierten Systemen in gleicher Weise wie bei DL-Systemen und muss durch zusätzliche Mechanismen, z.B. probabilistische Ansätze, gelöst werden.

Echtzeitfähigkeit, statische und dynamische Verarbeitung. Betrachten wir zunächst die Verwendung von Beschreibungslogiken: Bei der Verarbeitung statischer Bilder können *alle* primitiven Objekte (also diejenigen, die nicht aus anderen Teilen zusammengesetzt sind) als Evidenzen in die ABox eingetragen werden. Dann können speziell entwickelte Inferenzmechanismen eines DL-System zur Generierung einer Szeneninterpretation genutzt werden. Schwieriger stellt sich das Problem bei der dynamischen Verarbeitung von Bildsequenzen dar: Anwendungen zur Interpretation von Bildsequenzen fordern in der Regel auch die echtzeitfähige und inkrementelle Verarbeitung der Eingabedaten (s. Abschnitt 1.2). Für die Verwen-

dung von Beschreibungslogiken bedeutet dies, dass Evidenzen Schritt für Schritt in die ABox eingetragen werden und ggf. auch wieder entfernt werden müssen (abhängig vom Interpretationsprozess). Bestehende DL-Systeme unterstützen die inkrementelle Modellkonstruktion jedoch nicht. Eine ähnliche Situation ergibt sich bei der Verwendung von Grammatiken: vereinfacht ausgedrückt, gilt es hier, eine Folge von Produktionsregeln zu bestimmen, so dass die generierten Terminalsymbole den beobachteten Evidenzen zugeordnet werden können, unter Berücksichtigung von typischerweise räumlichen und zeitlichen (bei Videosequenzen) Constraints. Die dynamische und inkrementelle Verarbeitung von Bildsequenzen bedeutet, dass Schritt für Schritt neue Evidenzen geliefert werden, für die dann möglicherweise (oft) vollständig neue Folgen von Produktionsregeln bzw. Und-Oder-Graphen berechnet werden müssen (wenn man die Arbeit von Mumford und Zhu [77] zu Grunde legt). Regelbasierte Systeme dagegen sind generell für die inkrementelle Verarbeitung von Daten geeignet. Die Regeln einer *Regelbasis* können sukzessive auf eine sich dynamisch verändernde Menge von Fakten – die *Faktenbasis* – angewandt werden. Auf die Grundlagen von regelbasierten Systemen wird in Abschnitt 4.3 eingegangen. Probabilistische Verfahren, die auf Bayes-Netzen basieren, sind durch ihre effizienten Inferenz-Algorithmen grundsätzlich für echtzeitfähige Anwendungen geeignet. Die in [79] vorgestellten Bayes'schen Kompositionellen Hierarchien sind darüber hinaus speziell an die für die Szeneninterpretation essentielle Aggregatstruktur angepasst und bieten weitere Performanzvorteile.

Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses. Es ist eine grundsätzliche Eigenschaft moderner standardisierter Wissensrepräsentationsmethoden, dass die Wissensbasis und der Inferenzprozess voneinander getrennt sind. Bei Beschreibungslogiken werden Konzepte der Wissensbasis deklarativ in Form von terminologischem Wissen der TBox beschrieben. Dieses Wissen wird dann auf Fakten der ABox angewandt. Hier ist die Wissensbasis vom Interpretationsprozess strikt getrennt. Bei regelbasierten Systemen wird die Wissensbasis in Form einer deklarativen Regelbasis und einer *Faktenbasis* – die Menge der Fakten, die bei der Szeneninterpretation Evidenzen entsprechen – repräsentiert. Eine Inferenzmaschine führt dann die Regeln aus. Regeln werden also zur Repräsentation von konzeptuellen Wissen *und* zur Steuerung des Interpretationsprozesses verwendet; ein bekannter Nachteil regelbasierter Systeme, besonders bei manuell erstellten Regeln. Ähnliches gilt für Grammatiken: Hier besteht die Wissensbasis aus Produktionsregeln und Fakten, die den Evidenzen entsprechen, repräsentiert durch Nichtterminalsymbole.

Verarbeitung unsicherer und unvollständiger Eingabedaten. Probabilistische Ansätze sind prädestiniert für die Verarbeitung unsicherer und unvollständiger Eingabedaten. Logikbasierte Ansätze, Grammatiken und regelbasierte Systeme bieten hier keine grundsätzlichen Mechanismen, um unvollständige oder unsichere Eingabedaten zu verarbeiten. Aus diesem Grund werden sie häufig mit probabilisti-

schen Methoden kombiniert (s. Kapitel 2). In Tabelle 1 sind die Erkenntnisse der in diesem Abschnitt angestellten Überlegungen zusammenfassend dargestellt.

Basierend auf den Erkenntnissen der bestehenden Ansätze aus Kapitel 2 und den Anforderungen an ein generisches Interpretationssystem kann nun die zentrale Leitfrage dieser Arbeit formuliert werden:

Wie muss ein generisches Rahmenwerk für ein wissensbasiertes Szeneninterpretationssystem für komplexe Anwendungen aufgebaut sein, welches auf standardisierten Wissensrepräsentationsmethoden basiert und sowohl statische Bilder, als auch dynamische Bildsequenzen in Echtzeit verarbeiten kann, und welches in der Lage ist, auch unsichere und unvollständige Eingabedaten auszuwerten?

	Transparente Repräsentation von Konzepthierarchien	Dynamische Verarbeitung, Echtzeitfähigkeit	Trennung von Wissensbasis und Interpretationsprozess	Verarbeitung unsicherer und unvollständiger Eingabedaten
Beschreibungslogiken	+	o	++	o
Grammatiken	+	o	o	+
Regelbasierte Systeme	o	++	o	+
Probabilistische Ansätze (basierend auf Bayes-Netzen)	+	+	++	++

Tabelle 1: Wissensrepräsentationsmethoden und Anforderungen eines generischen Rahmenwerks für die Szeneinterpretation.

3.2 Grundlegender Ansatz

Die Zusammenfassung von Wissensrepräsentationsmethoden und Anforderungen eines generischen Rahmenwerks für die Szeneinterpretation (s. Tabelle 1) lässt vermuten, dass es bisher keine Wissensrepräsentationstechnik für sich alleine genommen gibt, die alle Anforderungen in vollem Umfang befriedigend erfüllt. Daher werden in der Regel verschiedene Techniken miteinander kombiniert (s. Kapi-

tel 2), um den Anforderungen der wissensbasierten Bilddeutung gerecht zu werden. Auch der in dieser Arbeit vorgestellte Ansatz eines generischen Rahmenwerks für die Szeneinterpretation basiert auf einer Kombination verschiedener Techniken.

Zwei Komponenten sind essentiell für ein wissensbasiertes Interpretationssystem: die Wissensbasis und der Interpretationsprozess. Ein wichtiger Aspekt dieser Arbeit liegt in der Generizität des Interpretationssystems, d.h. der Anwendbarkeit für unterschiedliche Domänen. Die Voraussetzung dafür ist eine leicht verständliche Repräsentationsform und klare Strukturierung der Wissensbasis. Weiterhin ist die Verfügbarkeit von Software-Werkzeugen wünschenswert, welche die Erstellung, Modifikation und Wartung der Wissensbasis unterstützen. Beschreibungslogiken bieten hierfür die besten Voraussetzungen. Demgegenüber stehen die Nachteile, dass Beschreibungslogiken eine flexible, schrittweise Interpretation nicht in natürlicher Weise unterstützen und, dass die Zuordnung von quantitativen Sensordaten der niederen Bildverarbeitung zu qualitativen symbolischen Daten außerhalb der Logik liegt.

Die Fähigkeit zur inkrementellen, echtzeitfähigen Verarbeitung dynamischer Bildsequenzen betrifft in erster Linie den Interpretationsprozess. Regelbasierte Systeme bieten sich hier an, da sie effizient auf einer sich dynamisch verändernden Faktenbasis arbeiten können, und leistungsstarke Regelmaschinen zur Verfügung stehen. Darüber hinaus lässt sich ein regelbasierter Prozess gut parallelisieren und mit probabilistischen Methoden kombinieren, wie sich im weiteren Verlauf dieser Arbeit zeigen wird. Beides ist wichtig, um dem beschriebenen Problem der Mehrdeutigkeit der Szeneninterpretation zu begegnen. Bei komplexen Anwendungen der realen Welt, welche im Fokus dieser Arbeit stehen, ist es unvermeidlich, dass Eingabedaten unsicher und unvollständig sind. Das ist ein weiterer Grund für eine Architektur des Systems, welche eine modulare Integration probabilistischer Methoden ermöglicht. Demgegenüber stehen die Nachteile der aufwändigen Erzeugung einer Regelbasis, die bisher nur durch kostspielige und fehleranfällige domänenspezifische Programmierung zu erreichen war. Außerdem ist es schwierig, die logische Konsistenz bei manuell erzeugten Regelbasen zu gewährleisten.

Eine zentrale Idee dieser Arbeit ist es daher zu untersuchen, inwieweit eine auf Beschreibungslogiken basierte Wissensbasis *automatisch* in eine konsistente Regelbasis übersetzt werden kann, um auf diese Weise die Vorteile beider Techniken zu kombinieren. Des Weiteren werden Mechanismen untersucht und entwickelt, die eine echtzeitfähige bottom-up und top-down Verarbeitung und Parallelisierung ermöglichen, welche durch ein Präferenzmaß gesteuert wird.

4 Grundlagen

In diesem Kapitel werden die Grundlagen der verwendeten Techniken für die Entwicklung eines generischen Rahmenwerks zur wissensbasierten Szeneninterpretation beschrieben. In Abschnitt 4.1 werden zunächst die Wissensrepräsentation und die beschreibungslogischen Grundlagen behandelt, die zu der zentralen Beschreibungssprache OWL (Web Ontology Language) führen. Dann werden in Abschnitt 4.2 Ontologien behandelt. In Abschnitt 4.3 wird auf die Grundlagen regelbasierter Systeme eingegangen.

4.1 Wissensrepräsentation

Die Repräsentation von Informationen zur Entwicklung intelligenter Systeme spielt eine wesentliche Rolle in der Künstlichen Intelligenz. Wissensrepräsentation im Kontext der Szeneninterpretation dient dazu, Aspekte der realen Welt zu repräsentieren, wie z.B. Aktionen, Ereignisse, physikalische Objekte, räumliche, zeitliche und andere Beziehungen etc. Dabei gibt es einerseits *explizite* Repräsentationen, bei denen das Wissen symbolsprachlich orientiert ist. Beispiele hierfür sind die Aussagenlogik, Prädikatenlogik, Beschreibungslogik, semantische Netze, Skripte, Frames und Konzeptgraphen (für eine detaillierte Beschreibung, s. [73], [92]). Hier liegt das Wissen in *deklarativer* Natur vor und ist von der Inferenz getrennt (Frames können jedoch auch *prozedurale* Informationen enthalten). Demgegenüber stehen *implizite* Ansätze, bei denen das Wissen über Objekte und deren Verarbeitung vermischt vorliegt. Hierzu gehören beispielsweise *prozedurale* Ansätze, bei denen Objekte und Prozeduren direkt als Programmcode kodiert werden, und *konnektionistische* Ansätze, bei denen Wissen *subsymbolisch*, d.h. durch die Verknüpfung einfacher Einheiten repräsentiert wird, wie beispielsweise bei neuronalen Netzen. Hier ist das Wissen implizit in der Struktur und den Gewichten des Netzes kodiert (s. [73]).

Das Hauptinteresse der Logik ist auf die Entwicklung formaler Repräsentationssprachen ausgerichtet, deren Schlussregeln korrekt und vollständig sind. Spezielle Repräsentationssysteme, wie z.B. semantische Netzwerke und Beschreibungslogi-

ken wurden eingeführt, um die Anordnung einer *Kategoriehierarchie* und *Vererbung* zu unterstützen. Vererbung ist eine wichtige Form der Inferenz und erlaubt, dass die Eigenschaften von Objekten aus ihrer Zugehörigkeit zu Kategorien hergeleitet werden. Außerdem ermöglicht die Vererbung das Speichern von Informationen auf der Ebene der größtmöglichen Abstraktion. Dadurch wird die Größe der Wissensbasis verringert und das Risiko von Inkonsistenzen minimiert.

4.1.1 Beschreibungslogiken

Eine Familie von Sprachen, die für die in dieser Arbeit identifizierten Anforderungen geeignet zu sein scheint, sind die sogenannten *Beschreibungslogiken* oder auch *Terminologischen Logiken*. Die meisten Beschreibungslogiken sind eine Untermenge der Prädikatenlogik erster Stufe, im Gegensatz zu dieser aber entscheidbar. Wie bereits erwähnt, sind semantische Netzwerke und Beschreibungslogiken u.a. aus dem Bedürfnis entstanden, Kategoriehierarchien und Vererbung zu unterstützen. Beschreibungslogiken haben aber im Gegensatz zu semantischen Netzwerken und Frames eine wohldefinierte, logikbasierte Semantik. Die Bezeichnung „Beschreibungslogik“ ist auf die Tatsache zurückzuführen, dass Begriffe der Anwendungsdomäne durch objektzentrierte Formalismen beschrieben werden, bei denen die Definition von *Konzepten* im Vordergrund steht. Diese entsprechen logisch gesehen einstelligen Prädikaten. Des Weiteren gibt es *Rollen* und *Attribute*, die Beziehungen zu anderen Konzepten beschreiben. Rollen entsprechen logisch gesehen zweistelligen Prädikaten und Attribute partiellen Funktionen.

Eine Wissensbasis, die durch eine Beschreibungslogik definiert wird, besteht aus zwei Komponenten: der TBox und der ABox. Die TBox enthält das terminologische Wissen, d.h. Wissen über die Konzepte der Domäne in Form von TBox-Axiomen. Sei TBox T eine Menge von TBox-Axiomen, dann wird T von einer Interpretation I genau dann erfüllt, wenn I alle Axiome in T erfüllt. Dann ist I ein Modell von T.

Die ABox hingegen enthält das Wissen über *Instanzen* dieser Konzepte, sowie deren Beziehung zueinander. Sie repräsentiert den Zustand der modellierten Welt in Form von Konzept- und Rollenzusicherungen über *Individuen* (Instanzen). Eine Konzeptzusicherung $C(a)$ bedeutet, dass die Interpretation des Individuums a zur Interpretation des Konzepts C gehört. Eine Rollenzusicherung $r(a,b)$ gibt an, dass Individuum a zu Individuum b über Rolle r in Beziehung steht. Die Semantik wird wieder über die Interpretation $I=(\Delta^I,\cdot^I)$ definiert (s. Tabelle 3).

In einem DL-System können neben den atomaren Konzepten und Rollen mit Hilfe der jeweiligen Beschreibungssprache komplexe Konzepte und Rollen definiert

werden. Ein DL-System enthält jedoch nicht nur Wissen in Form von TBox und ABox, sondern bietet auch Inferenzdienste an, um über dieses Wissen zu schließen. Typische Anfragen sind beispielsweise, ob eine Wissensbasis erfüllbar ist oder ob ein Konzept genereller ist als ein anderes, d.h. ob das erste Konzept das zweite *subsumiert*. Wichtige Probleme einer ABox sind herauszufinden, ob eine Menge von Einträgen *konsistent* ist, d.h. ob sie ein Modell hat oder ob sie zur Folge hat, dass ein bestimmtes Individuum eine Instanz eines gegebenen Konzepts ist. In einer Applikation ist ein DL-System in der Regel in eine größere Umgebung eingebettet. Andere Komponenten stellen Anfragen an die Wissensbasis oder modifizieren sie, indem sie Konzepte, Rollen und Instanzen einfügen oder löschen (s. Abbildung 9**)**.

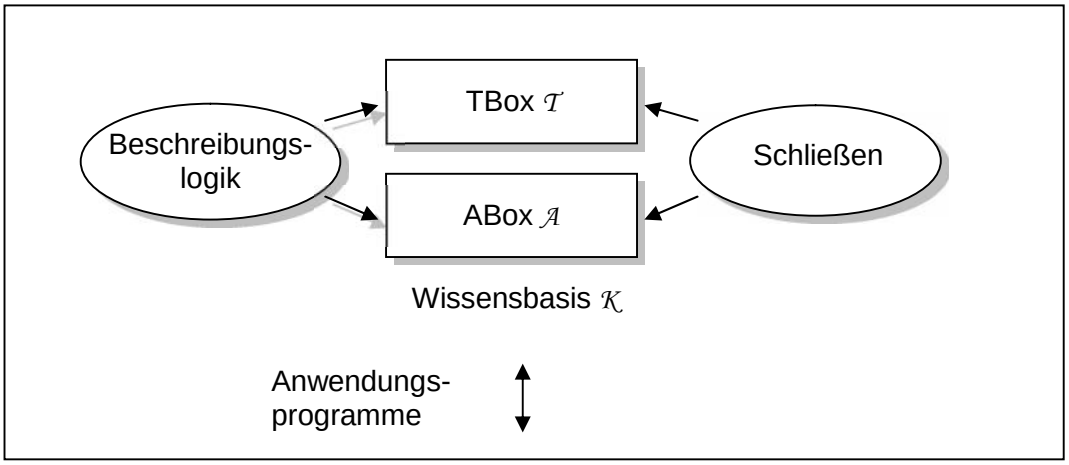


Abbildung 9: Architektur eines beschreibungslogischen wissensbasierten Systems (nach [27]).

Die Beschreibungslogiken $\mathcal{SHIF}(\mathcal{D})$ und $\mathcal{SHOIN}(\mathcal{D})$ basieren auf der Beschreibungslogik $\mathcal{ALC}$ (für weiteres Details, s. [27]) und bilden die Grundlage für OWL. Sie werden im Folgenden definiert.

4.1.1.1 Die Beschreibungslogik $\mathcal{SHIF}(\mathcal{D})$

Definition 4-1: $\mathcal{SHIF}(\mathcal{D})$

Die Buchstaben von $\mathcal{SHIF}(\mathcal{D})$ *haben folgende Bedeutung (s. auch Tabelle 2):*

- $\mathcal{S}$ *steht für* $\mathcal{ALC}$ *plus Rollentransitivität,*
- $\mathcal{H}$ *steht für Rolleninklusion,*
- $\mathcal{I}$ *steht für inverse Rollen,*

- $\mathcal{F}$ *steht für funktionale Rollen,*
- $\mathcal{D}$ *steht für Datentypen.*

$\mathcal{S}$, $\mathcal{H}$, I und $\mathcal{F}$ erklären sich aus den Definitionen in Tabelle 2. Ein Datentyp (auch konkrete Domäne genannt) $\mathcal{D}$ besteht aus eine Menge $\Delta^{\mathcal{D}}$, der Domäne, und einer Menge $pred(\mathcal{D})$, den Prädikatnamen von $\mathcal{D}$. Jeder Name $P \in pred(\mathcal{D})$ ist mit einem n-stelligen Prädikat $P^{\mathcal{D}} \subseteq (\Delta^{\mathcal{D}})^n$ assoziiert.

Beispiel 4-1: Konkrete Domänen

- Eine Domäne $\mathcal{N}$ hat als Menge $\Delta^{\mathcal{N}} = \mathbb{N}$ und die binären Prädikatnamen $<,\leq,\geq,>$ und die unären Prädikatnamen $<_n,\leq_n,\geq_n,>$ mit $n \in \mathbb{N}$, assoziiert mit Prädikaten auf $\mathbb{N}$ mit der üblichen Interpretation.
- Domänen können auch komplexer sein, beispielweise sei IC das Allen-Intervall-Kalkül [24]. Dann besteht Δ^{IC} aus Zeitintervallen und die Prädikate werden aus den Allen-Relationen (bevor, während, ...) gebildet.

4.1.1.2 Die Beschreibungslogik $\mathcal{SHOIN}(\mathcal{D})$

Definition 4-2: $\mathcal{SHOIN}(\mathcal{D})$

Aufbauend auf den schon in $\mathcal{SHIF}(\mathcal{D})$ *enthaltenen Sprachelementen gilt für die Sprache* $\mathcal{SHOIN}(\mathcal{D})$ *zusätzlich:*

- $\mathcal{O}$ *steht für abgeschlossene Klassen,*
- $\mathcal{N}$ *steht für Zahlenrestriktionen.*

Abgeschlossene Klassen bestehen aus einer Menge von individuellen Namen (auch Nominale genannt). Damit lässt sich beispielsweise Ausdrücken, dass die fünf ständigen Vertreter des Sicherheitsrats der Vereinten Nationen {Frankreich, Russland, USA, China, UK} sind.

Syntax	Semantik	Beschreibung	Symbol
A	$A^I \subseteq \Delta^I$	Konzeptname	$\mathcal{AL}$
$\top$	Δ^I	universelles Konzept	$\mathcal{AL}$
$\bot$	$\varnothing$	leeres Konzept	$\mathcal{AL}$
$C \sqsubseteq D$	$C^I \subseteq D^I$	allgemeine Konzeptinklusion	$\mathcal{ALC}$
$C \sqcap D$	$C^I \cap D^I$	Schnittmenge	$\mathcal{AL}$
$C \sqcup D$	$C^I \cup D^I$	Vereinigung	$\mathcal{U}$
$\neg C$	$\Delta^I \setminus C^I$	allgemeine Negation	$\mathcal{C}$
$\forall R.C$	$\{a \in \Delta^I \mid \forall b:(a,b) \in R^I \rightarrow b \in C^I\}$	Wertebeschränkung	$\mathcal{AL}$
$\exists R.C$	$\{a \in \Delta^I \mid \exists b:(a,b) \in R^I \wedge b \in C^I\}$	Existenzquantifizierung	$\mathcal{E}$
$\geq_n R$	$\{a \in \Delta^I \mid \mid \{b \mid (a,b) \in R^I\} \mid \geq n\}$	unqualifizierte Anzahlrestriktion	$\mathcal{N}$
$\leq_n R$	$\{a \in \Delta^I \mid \mid \{b \mid (a,b) \in R^I\} \mid \leq n\}$		
$\geq_n R.C$	$\{a \in \Delta^I \mid \mid \{b \mid (a,b) \in R^I \wedge b \in C^I\} \mid \geq n\}$	qualifizierte Anzahlrestriktion	$\mathcal{Q}$
$\leq_n R.C$	$\{a \in \Delta^I \mid \mid \{b \mid (a,b) \in R^I \wedge b \in C^I\} \mid \leq n\}$		
$\{a\}$	$\{a^I\} \subseteq \Delta^I$	abgeschlossene Klasse	$\mathcal{O}$
R	$R^I \subseteq \Delta^I \times \Delta^I$	Rollenname	$\mathcal{AL}$
R^-	$\{(b,a) \in \Delta^I \times \Delta^I \mid (a,b) \in R^I\}$	inverse Rolle	$\mathcal{I}$
$Trans(R)$	$(a,b) \in R^I \wedge (b,c) \in R^I \rightarrow (a,c) \in R^I$	transitive Rolle	$+$
$Func(R)$	$\{(a,b),(a,c)\} \subseteq R^I \rightarrow b = c$	funktionale Rolle	$\mathcal{F}$
$R \sqsubseteq S$	$R^I \subseteq S^I$	Rolleninklusion	$\mathcal{H}$

Tabelle 2: Syntax und Semantik der Konstruktoren für komplexe Konzept- und Rollenbeschreibungen.

Syntax	Semantik	Beschreibung
$C(a)$	$a^I \in C^I$	Konzeptzugehörigkeit
$r(a,b)$	$(a^I, b^I) \in r^I$	Rollenzugehörigkeit
$a = b$	$a^I = b^I$	Gleichheit
$a \neq b$	$a^I \neq b^I$	Ungleichheit

Tabelle 3: Syntax und Semantik von ABox-Zusicherungen.

4.1.2 Die Beschreibungssprache OWL

Die Beschreibungssprachen OWL (Web Ontology Language) ist eine Spezifikation des *World Wide Web Konsortiums*[4] (W3C), um Ontologien anhand einer formalen Beschreibungssprache zu definieren. Ontologien sind formale Darstellungen einer Menge von Begrifflichkeiten und der zwischen ihnen bestehenden Beziehungen in einer bestimmten Domäne. Sie dienen dazu, Wissensbasen beschreibungslogisch zu repräsentieren, um sie beispielsweise für Anwendungsprogramme und Dienste des Internets zur Verfügung zu stellen. In Abschnitt 4.2 wird detaillierter auf Ontologien eingegangen.

Die Sprache OWL gibt es in drei Varianten für unterschiedliche Anforderungen. Mit der bisherigen Vorarbeit können nun zumindest zwei dieser Varianten präzise definiert werden:

- *OWL Lite*. Die „Light-Version“ dient vor allem zur Erstellung einfacher Taxonomien und leicht axiomatisierter Ontologien. Sie entspricht im Wesentlichen der Beschreibungslogik $\mathcal{SHIF}(\mathcal{D})$. OWL Lite ist entscheidbar und Teilsprache von OWL DL.

[4] Das World Wide Web Consortium ist ein Gremium zur Standardisierung von Techniken, die das World Wide Web betreffen.

- *OWL DL*. DL steht für „Description Logic“. OWL DL entspricht der Beschreibungssprache $\mathcal{SHOIN}(\mathcal{D})$ und ist Teilsprache von OWL Full. OWL DL ist bei hoher Ausdrucksstärke noch entscheidbar und ermöglicht automatisches Schließen und automatische Konsistenztests. Daher ist diese Variante für viele Anwendungen interessant.

- *OWL Full*. OWL Full ist sehr ausdrucksstark, jedoch im Allgemeinen unentscheidbar. Diese Sprachvariante ist für Anwendungsfälle gedacht, bei denen hohe Ausdrucksstärke wichtiger ist als die Garantie der Entscheidbarkeit und automatisches Schließen. Eine Hauptursache für die Unentscheidbarkeit ist die nicht vorhandene Typentrennung zwischen Individuen, Klassen und Rollen. Dadurch eröffnet sich die Möglichkeit, Aussagen über Klassen und Rollen zu definieren, in denen diese Klassen oder Rollen syntaktisch als Individuen auftreten. Man spricht in diesem Fall von *Metamodellierung* (s. [63]).

Für die Entwicklung eines generischen Rahmenwerks für ein wissensbasiertes Szeneninterpretationssystem ist eine hohe Ausdruckstärke der zu Grunde liegenden Beschreibungslogik ebenso wichtig wie Entscheidbarkeit, automatisches Schließen und automatische Konsistenzprüfungen. Daher fiel die Wahl, der in dieser Arbeit verwendeten Beschreibungssprache, auf OWL DL.

Im nächsten Abschnitt wird nun der zentrale Begriff der „Ontologie“ im Detail erläutert. Außerdem wird die Sprache OWL im Kontext des *Schichtenmodells des Semantischen Webs* betrachtet, aus dem sich ihre Syntax ableitet.

4.2 Ontologien

Der wohl bekannteste Definitionsversuch stammt von Tom Gruber [56], der eine Ontologie als „explizite formale Spezifikation einer Konzeptualisierung“ bezeichnet. Nach Guarino und Giarette [60] ist eine andere Möglichkeit, Ontologie als Synonym für *ontologische Theorie* zu verwenden, welche der TBox entspricht. Daraus ergibt sich die Definition einer Ontologie als „Repräsentation eines konzeptuellen Systems durch eine logische Theorie“. Diese Definition hat eine gewisse Eleganz, da sie den Begriff der Ontologie im Wesentlichen mit der TBox gleichsetzt und ihn damit deutlich gegenüber dem Begriff der Wissensbasis abgrenzt, die sowohl die TBox, als auch die ABox umfasst.

Guarino [78] schlägt eine prinzipielle Unterscheidung von vier Arten von Ontologien vor, abhängig von ihrem Generalisierungsgrad (s. Abbildung 10).

- *Top-Level-Ontologien* beschreiben sehr generelle Konzepte, wie Raum, Zeit, Objekte, Aktionen, usw., die unabhängig von einer bestimmten Domäne sind. Es ist daher sinnvoll, vereinheitlichte Top-Level-Ontologien für große Nutzergemeinschaften zu definieren.

- *Domänen-Ontologien* und *Aufgaben-Ontologien* beschreiben spezielle Domänen (z.B. Medizin, Automobile, Flughäfen) bzw. spezielle Aufgaben oder Aktivitäten (z.B. Diagnostizieren, Verkaufen) durch Spezialisierung der in der Top-Level-Ontologie eingeführten Konzepte.

- *Applikations-Ontologien* beschreiben Konzepte und Aufgaben einer konkreten Anwendung, die durch Spezialisierung der Domänen-Ontologien und Aufgaben-Ontologien gebildet werden.

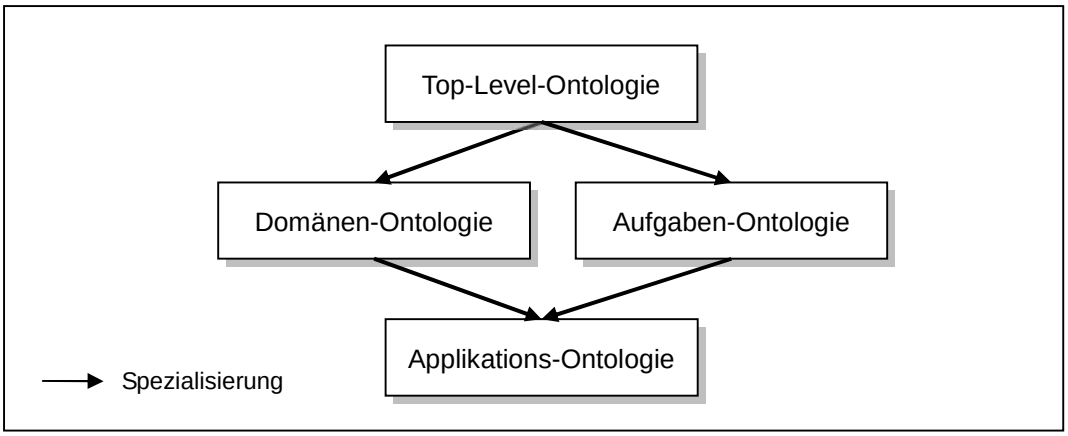


Abbildung 10: Arten von Ontologien, abhängig von ihrem Generalisierungsgrad.

Die Unterteilung der Ontologie in eine Top-Level-Ontologie und eine Domänen-Ontologie wird in dieser Arbeit umgesetzt (s. Abschnitt 5.1). Da die Anwendung als Szeneninterpretationssystem festgelegt ist (und damit auch die Aufgaben), entfällt eine weitere Unterteilung.

4.2.1 Semantisches Web

Ontologien haben in den vergangenen Jahren vor allem durch die Idee des *semantischen Webs* an Popularität gewonnen. Das semantische Web ist eine Weiterentwicklung des *World Wide Web*, mit dem Ziel, einen höheren Automatisierungsgrad bei der Verarbeitung von Informationen des Internets zu erreichen. Die Idee des

semantischen Webs ist es, Informationen in einer Form zur Verfügung zu stellen, die von Maschinen verarbeitet werden kann. Ein zentrales Element ist die Repräsentation von Metadaten – also Daten über Daten – in Form von Anmerkungen oder elektronischen Karteikarten (s. [58], S. 34).

Die Voraussetzung, Informationen zwischen verschiedenen Anwendungen und Plattformen auszutauschen und mit Hilfe der Logik zueinander in Beziehung zu setzen – auch *Interoperabilität* genannt – ist die Schaffung offener Standards mit formal klaren Definitionen, welche flexibel und erweiterbar sind (s. [63]). Genau zu diesem Zweck hat das World Wide Web Konsortium (W3C) die grundlegenden Standards XML, RDFS und OWL definiert. Nachdem die Sprache OWL bereits aus logikbasierter Sicht eingeführt wurde (s. Abschnitt 4.1.2), wird sie im Folgenden im Kontext dieser Standards und des Schichtenmodells für das Semantische Web betrachtet.

4.2.1.1 RDF und RDFS

RDF (*Resource Description Framework*) ist eine formale Sprache für die Beschreibung strukturierter Informationen und wird als grundlegendes Darstellungsformat des Semantischen Webs angesehen. Ein RDF-Dokument beschreibt einen gerichteten Graphen (*RDF-Graphen*), bei dem sowohl Knoten als auch Kanten mit eindeutigen Bezeichnern beschriftet sind. RDF-Graphen werden durch eine Menge von *RDF-Tripeln* der Form „Subjekt-Prädikat-Objekt“ angegeben. Diese Graphen repräsentieren „Dinge“ (in RDF *Ressourcen* genannt) und deren Beziehung zueinander. Um eindeutige Namensräume zu schaffen und Mehrdeutigkeiten zu vermeiden, werden in RDF grundsätzlich URIs (*Uniform Resource Identifier*) zur Bezeichnung aller Ressourcen verwendet. Konkrete Datenwerte werden in RDF durch *Literale* dargestellt. Das sind reservierte Bezeichner für RDF-Ressourcen eines bestimmten Datentyps (s. Abbildung 11**)**.

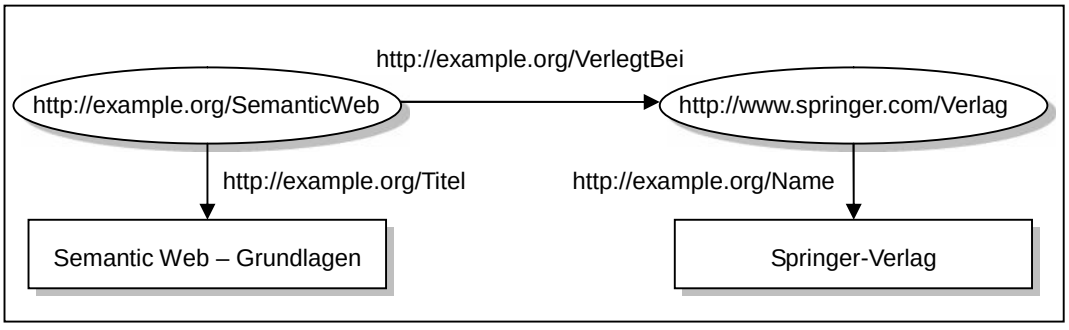


Abbildung 11: Ein RDF-Graph mit Literalen zur Beschreibung von Datenwerten (nach [63]).

Mit dem Prädikat `rdf:type` können Subjekten (*Individuen*) *Klassenbezeichner* zugewiesen werden. Damit können in RDF Aussagen über Individuen und deren Beziehungen (engl.: *Properties*) und einfache Klassenzugehörigkeiten ausgedrückt werden. Es ist jedoch nicht möglich, terminologisches Wissen zu modellieren, also Aussagen über Klassen auszudrücken, beispielsweise, dass jede Universität eine Institution ist. Um das zu ermöglichen benötigt man RDFS.

RDFS steht für *Resource Description Framework Schema*, wobei Schema für *Schemawissen* steht und gleichbedeutend mit terminologischem Wissen ist. RDFS stellt universelle Ausdrucksmittel zur Verfügung, die es ermöglichen, Aussagen über semantische Beziehungen der Termini eines beliebigen nutzerdefinierten Vokabulars zu treffen (s. [63], S. 67). Beispiele dafür sind die Prädikate `rdfs:Class` und `rdfs:subClassOf`, mit denen Klassen definiert, Unterklassenbeziehungen ausgedrückt und damit Klassenhierarchien modelliert werden können – eine wichtiges Mittel zur Wissensmodellierung. Weitere wichtige Prädikate sind `rdfs:subPropertyOf`, zum Modellieren von Rollenhierarchien und `rdfs:domain`, zum Einschränken von Definitionsbereichen und `rdfs:range`, zum Einschränken von Wertebereichen (für eine Zusammenfassung aller RDFS-Sprachkonstrukte, s. [63], S. 85 - 86). Damit ist RDFS eine Wissensrepräsentationssprache oder auch Ontologiesprache für einfache Ontologien.

Die *modelltheoretische Semantik* für RDFS (d.h. die durch eine Interpretation definierte Semantik) ist recht komplex, und auf ihre Herleitung wird an dieser Stelle aus Gründen der Übersichtlichkeit verzichtet. Sie ist hier nicht zwingend Notwendig, da die Verfahren dieser Arbeit auf der Sprache OWL DL aufbauen und ihre Semantik bereits ausführlich behandelt wurde. Für weitere Details sei deshalb auf [63] verwiesen.

4.2.1.2 OWL als Erweiterung von RDFS

OWL-Dokumente beschreiben *OWL Ontologien*, basierend auf der RDF-Syntax (auch *OWL-RDF-Syntax* genannt). Eine OWL Ontologie besteht im Wesentlichen aus Klassen und Rollen (Beziehungen zwischen Klassen oder Klassen und Datentypen), wie eine RDFS-Ontologie auch. Allerdings können in OWL diese Klassen und Rollen in komplexere Beziehung zueinander gesetzt werden, beispielsweise durch die Prädikate `owl:someValuesFrom` und `owl:allValuesFrom`, für qantifizierte Aussagen der From „jede Prüfung muss mindestens einen Prüfer haben“. Die Ausdrucksstärke von OWL Lite und OWL DL wurde bereits in Abschnitt 4.1.1 ausführlich erläutert. Die Sprache OWL Full enthält ganz RDFS und ist daher unentscheidbar, da es in RDFS keine strenge Typentrennung zwischen Individuen, Klassen und Rollen gibt, wie sie in OWL DL vorgeschrieben ist. Somit können Klassen auch als Individuen auftreten und Aussagen über Klassen von

Klassen beschrieben werden. Abbildung 12 soll die Ausdrucksstärke und Zusammenhänge der verschiedenen Sprachen verdeutlichen.

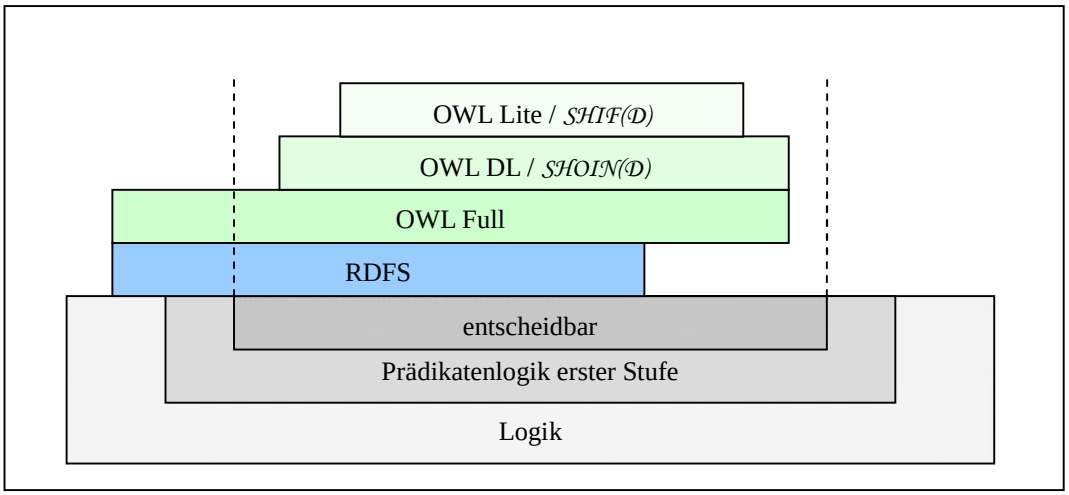


Abbildung 12: Ausdrucksstärke von RDFS, OWL Full, OWL DL und OWL Lite.

4.2.1.3 Schichtenmodell des Semantischen Webs

Die Prinzipien des Semantischen Webs sind in Schichten verschiedener Webtechnologien und Standards organisiert (s. [12], [58], [70]). Abbildung 13 zeigt eine schematische Darstellung. *Unicode*[5] und URIs gewährleisten die Verwendung internationaler Zeichen und – soweit möglich – eindeutiger Ressourcenidentifizierung. XML dient zur Definition von RDFS und OWL. Die wiederum dienen zur Beschreibung einfacher und komplexer Ontologien. Die Logikschicht wird in diesem Modell als Möglichkeit verstanden, Regeln auf Basis von Ontologien zu definieren, welche in der Beweisschicht ausgeführt und zusammen mit der Vertrauensschicht evaluiert werden können (s. [70]). Die Ergebnisse werden dann schließlich von der jeweiligen Anwendung weiterverarbeitet. Die Vertrauensschicht baut zusätzlich auf dem Konzept der *digitalen Unterschrift* auf. Darunter versteht man mit elektronischen Informationen verknüpfte Daten, die den Unterzeichner identifizieren und mit denen die Integrität der signierten Informationen geprüft werden kann.

[5] Internationaler Standard für Schriftzeichen und Textelemente aller bekannten Sprachkulturen und Schriftsysteme.

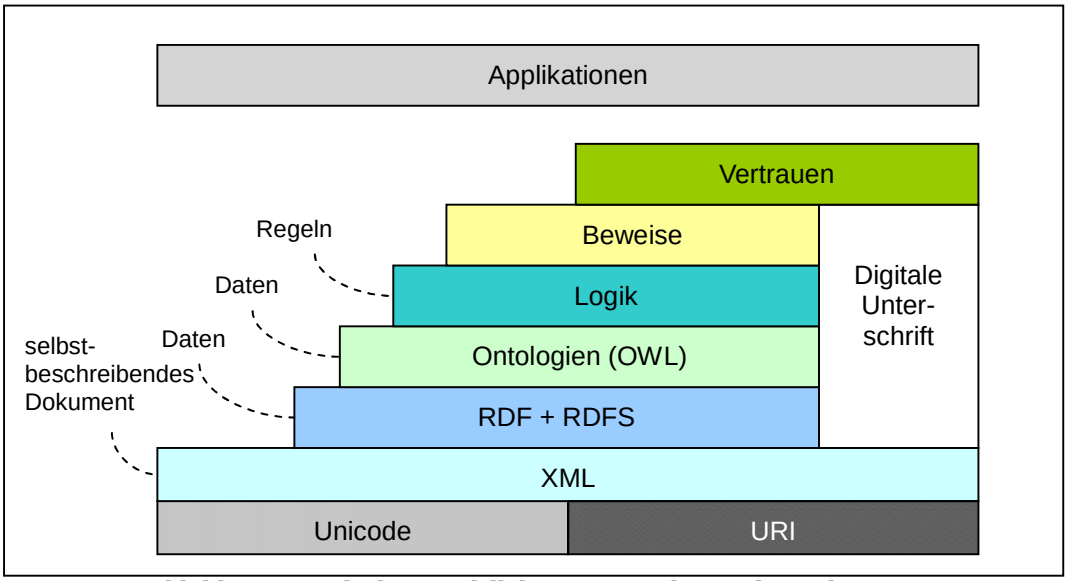


Abbildung 13: Schichtenmodell des Semantischen Webs nach W3C.

Laut Grütter [58] gibt es derzeit für alle Schichten bis einschließlich „Ontologien“ standardisierte Technologien und erste Anwendungen, während sich Technologien für die höheren Schichten in der Entwicklung befinden. Eine wichtige Institution ist die *Rule Markup Initiative* (RuleML), deren Ziel es u.a. ist, ein ganzheitliches Vorgehen bei der Entwicklung von Sprachen zur Definition von Regeln zu erreichen – z.B. RIF (*Rule Interchange Format*), siehe [66], [69]. Das in dieser Arbeit vorgestellte Interpretationssystem kann als eine Anwendung dieser höheren Schichten betrachtet werden, auch wenn es sich hierbei nicht um eine Anwendung des Semantischen Webs handelt.

4.2.2 OWL DL Ontologie

Nun sind die Grundlagen für die Definition einer OWL DL Ontologie (im weiteren Verlauf wird abkürzend auch der Begriff „OWL Ontologie“ verwendet) bereitgestellt.

Definition 4-3: OWL DL Ontologie

Eine OWL DL Ontologie ist eine Ontologie, formuliert in der Beschreibungssprache OWL DL in einem OWL-Dokument, welches auf der RDF-Syntax basiert.

OWL-Dokumente sind für den Menschen schwer lesbar, wie XML-basierte Dokumente im Allgemeinen. Es stehen jedoch einige Software-Werkzeuge zur Ver-

fügung, um OWL Ontologien zu modellieren. In dieser Arbeit wird der weit verbreitete Ontologie-Editor *Protégé*[6] der Stanford University verwendet.

Beispiel 4-2: OWL DL Ontologie

Abbildung 14 zeigt einen Screenshot von Protégé mit den Klassen `Pizza`, `PizzaBelag` und `PizzaBoden`. Die Klasse `PizzaBelag` hat die Unterklassen `Käse`, `Mozarella` und `Salami`. Es gilt also $\texttt{Käse} \sqsubseteq \texttt{PizzaBelag}$, $\texttt{Mozarella} \sqsubseteq \texttt{PizzaBelag}$ und $\texttt{Salami} \sqsubseteq \texttt{PizzaBelag}$. Die Klasse `PizzaBoden` hat die Unterklassen `DickerBoden` und `DünnerBoden`. Die Klasse `Pizza` hat die Rolle `hatBelag` mit der Existenzquantifizierung, dem Definitionsbereich `Pizza` und dem Wertebereich `PizzaBelag`, d.h., jede Pizza hat mindestens einen Pizzabelag. Die zweite Rolle ist `hatBoden`, mit einer qualifizierten Anzahlrestriktion von eins, dem Definitionsbereich `Pizza` und dem Wertebereich `PizzaBoden`, d.h., jede Pizza hat genau einen Boden. Durch die qualifizierte Anzahlrestriktion handelt es sich hier also um die Sprache $\mathcal{SHOIQ}(\mathcal{D})$, die aber ebenfalls zu OWL DL gerechnet wird (s. [63], S. 172).

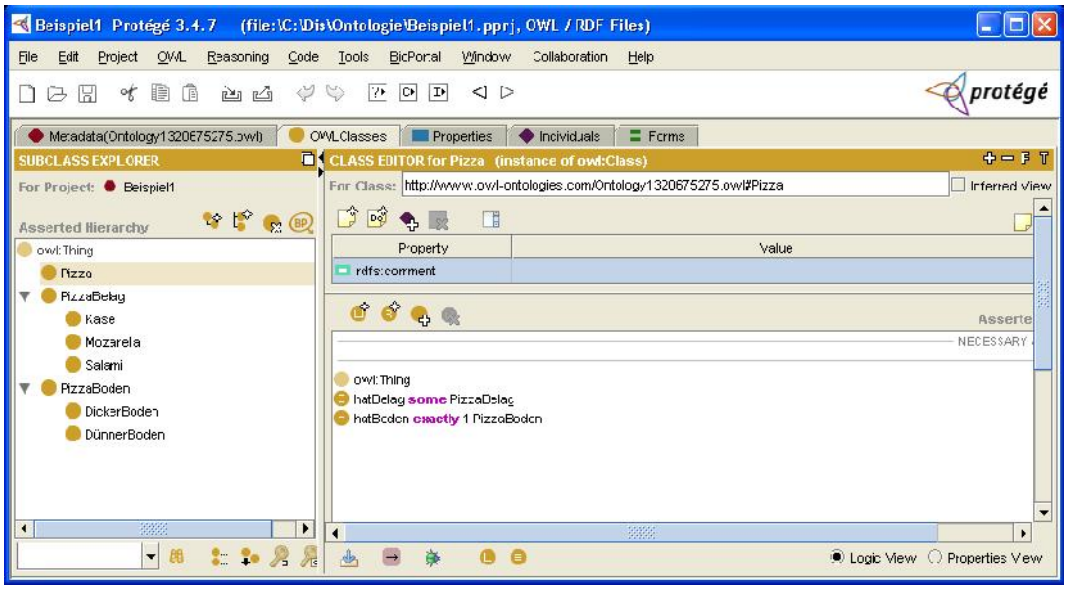


Abbildung 14: Ontologie-Editor Protégé.

[6] http://protege.stanford.edu/

4.2.3 SWRL

Die *Semantic Web Rule Language* (SWRL) ist eine Sprache zur Definition von Regeln, die OWL mit der *Rule Markup Language* (s. [19], abgekürzt ebenfalls RuleML) kombiniert. Sie dient dazu, eine OWL Ontologie durch das Hinzufügen von Regeln zu erweitern. SWRL-Regeln sind ein wichtiger Bestandteil dieser Arbeit und werden deshalb an dieser Stelle ausführlicher behandelt.

Definition 4-4: Syntax von SWRL

Die Syntax von SWRL-Regeln wird in OWL DL in Form Horn-ähnlicher Regeln (s. [95]) definiert. SWRL-Atome definieren sich zunächst wie folgt:

$$Atome \leftarrow C(i) \mid D(v) \mid R(i,j) \mid U(i,v) \mid builtIn(p,v_1,...,v_n) \mid i = j \mid i \neq j\,, \qquad (4\text{-}1)$$

mit:

- C *= Klasse,*
- D *= Datentyp,*
- R *= Objektrolle,*
- U *= Datentyprolle,*
- i,j *= Objektvariablennamen oder Objektinviduennamen,*
- $v_1,...,v_n$ *= Datentypvariablennamen oder Datentypindividuennamen,*
- p *= Name für Built-in-Funktionen.*

Nun kann die Syntax einer SWRL-Regel definiert werden:

$$a \leftarrow b_1,...,b_n\,, \qquad (4\text{-}2)$$

wobei a *- der Kopf (oder die Konklusion) – ein Atom ist und alle* b_s *Atome sind und den Rumpf (oder die Prämisse) bilden. Eine SWRL-Wissensbasis* $K = (\Sigma, P)$ *besteht aus einer* $\mathcal{SHOIN}(\mathcal{D})$*-Wissensbasis* Σ *und einer endlichen Menge von SWRL-Regeln* P*.*

Definition 4-5: Semantik von SWRL

Sei $I = (\Delta^I, \Delta^D, \cdot^I, \cdot^D)$ *eine Interpretation mit*

- Δ^I = *Objektinterpretationsdomäne,*
- Δ^D = *Datentypinterpretationsdomäne,*
- $\cdot^I$ = *Objektinterpretationsfunktion,*
- $\cdot^D$ = *Datentypinterpretationsfunktion,*

wobei $\Delta^I \cap \Delta^D = \varnothing$ *, so dass gilt:* $V_{IX} \to P(\Delta^I)$ *und* $V_{DX} \to P(\Delta^D)$*, mit*

- V_{IX} = *Objektvariablen,*
- V_{DX} = *Datentypvariablen,*
- P = *Potenzmengenoperator.*

Eine Prämisse ist erfüllt, wenn jedes Atom der Prämisse erfüllt ist. Eine Konklusion ist erfüllt, wenn jedes Atom der Konklusion erfüllt ist. Tabelle 4 zeigt die Syntax und Semantik von SWRL-Atomen.

SWRL-Atome	
Syntax	**Semantik**
$C(i)$	$i^I \in C^I$
$R(i,j)$	$(i^I, j^I) \in R^I$
$U(i,v)$	$(i^I, v^D) \in U^I$
$D(v)$	$v^D \in D^D$
$builtIn(p, v_1, ..., v_n)$	$(v_1^D, ..., v_n^D) \in p^D$
$i = j$	$i^I = j^I$
$i \neq j$	$i^I \neq j^I$

Tabelle 4: Syntax und Semantik von SWRL-Atomen.

Beispiel 4-3: SWRL-Regeln

Es gibt SWRL-Regeln, die auch in reinem OWL DL ausgedrückt werden können; in diesen Fällen sind die Regeln lediglich „syntaktischer Zucker“, z.B. lässt sich die Regel (ein ? bezeichnet Objektvariablennamen oder Datentypvariablennamen)

```
Computer(?c) ^
hasCPU(?c, ?cpu) ^
hasSpeed(?cpu, ?sp) ^
HighSpeed(?sp)
->
FastComputer(?c)
```
(4-3)

in OWL DL folgendermaßen formulieren:

$$\texttt{Computer} \sqcap \exists \texttt{hasCPU}.\exists \texttt{hasSpeed}.\texttt{HighSpeed} \sqsubseteq \texttt{FastComputer} \quad (4\text{-}4)$$

Es gibt jedoch auch SWRL-Regeln, die sich nicht in OWL DL ausdrücken lassen, z.B.:

```
hasParent(?newphew, ?parent) ^
hasBrother(?parent, ?uncle)
->
hasUncle(?newphew, ?uncle)
```
(4-5)

Der Grund dafür ist, dass in der Konklusion zwei verschiedene Variablen vorkommen. Grundsätzlich ist eine Übersetzung nach OWL DL nicht möglich, wenn Prämisse und Konklusion mehr als eine gemeinsame Variable haben (s. [68]).

Entscheidbarkeit. Im Allgemeinen ist eine SWRL-Wissensbasis nicht mehr entscheidbar. Entscheidbarkeit ist nur dann gegeben, wenn die Regeln *DL-sicher* sind. Ein intuitives Verständnis von DL-sicheren Regeln ist, dass alle in der Konklusion auftretenden Variablen auch in der Prämisse vorkommen und bei Regelanwendung konkrete Werte erhalten (s. [66]). Da sich die hier verwendeten SWRL-Regeln bzw. die durch den Übersetzungsprozess automatisch generierten Regeln für das regelbasierte Szeneninterpretationssystem nur auf Fakten beziehen, ist dies der Fall (s. auch Abschnitt 5.2.2).

4.3 Regelbasierte Systeme

Ein *regelbasiertes System* ist ein wissensbasiertes System, in dem regelbasiertes Schließen stattfindet. Die Programmierung dieser Systeme ist nicht prozedural, sondern deklarativ, d.h. sie basiert auf Fakten und Regeln. Regelbasierte Systeme wurden in den 70er und 80er Jahren populär, vor allem durch die Entwicklung von Expertensystemen (s. [52], [73]). MYCIN [43] beispielsweise war ein medizinisches Expertensystem für die Diagnose von spinaler Meningitis und bakteriellen Infektionen des Blutes, PROSPECTOR [43] ermittelte, basierend auf den geologischen Daten eines Gebietes, den wahrscheinlichsten Ort und Typ von Erzvorkommen.

4.3.1 Aufbau eines regelbasierten Systems

Regelbasierte Systeme bestehen aus

- einer Menge von Fakten, der *Faktenbasis,*
- einer Menge von Regeln, der *Regelbasis,*
- einem Kontrollsystem mit Regelinterpreter (auch *Inferenzmaschine* genannt).

Die Regeln liegen in folgender Form vor:

WENN < *Bedingungsteil* > *DANN* < *Aktionsteil* > . (4-6)

Der Bedingungsteil wird auch Prämisse oder LHS (engl.: *left hand side*), der Aktionsteil auch Konklusion oder RHS (engl.: *right hand side*) genannt. Die Inferenzmaschine besteht aus einem *Musterabgleicher*, einer *Agenda* und einer *Ausführungsmaschine* (s. Abbildung 15).

Der Arbeitsspeicher (Faktenbasis) enthält die Fakten, mit denen das System arbeitet. Die Aufgabe des Musterabgleichers ist es, diejenigen Regeln herauszufinden, die auf den aktuellen Inhalt des Arbeitsspeichers angewandt werden können, d.h. bei denen die Prämisse zu *wahr* ausgewertet wird. Ist das für eine Regel der Fall, wird die sie *aktiviert*. Die Liste der aktivierten Regeln, also derjenigen, die potenziell feuern könnten, bilden die Agenda (auch Konfliktmenge genannt). Sie hat die Aufgabe zu entscheiden, welche Regel die höchste Priorität bekommt und zuerst *feuern* (ausgeführt werden) soll. Dazu wird eine Konfliktstrategie benutzt; sie kann sehr einfach sein und z.B. der zuerst aktivierten Regel die höchste Priorität geben. Oftmals bieten Regelmaschinen einfache Strategien zur Auswahl an. Eine Strategie kann aber auch vom Systementwickler reimplementiert werden und be-

liebig komplex sein. Wenn schließlich entschieden wurde, welche Regel feuern soll, wird ihre Konklusion von der Ausführungsmaschine ausgeführt.

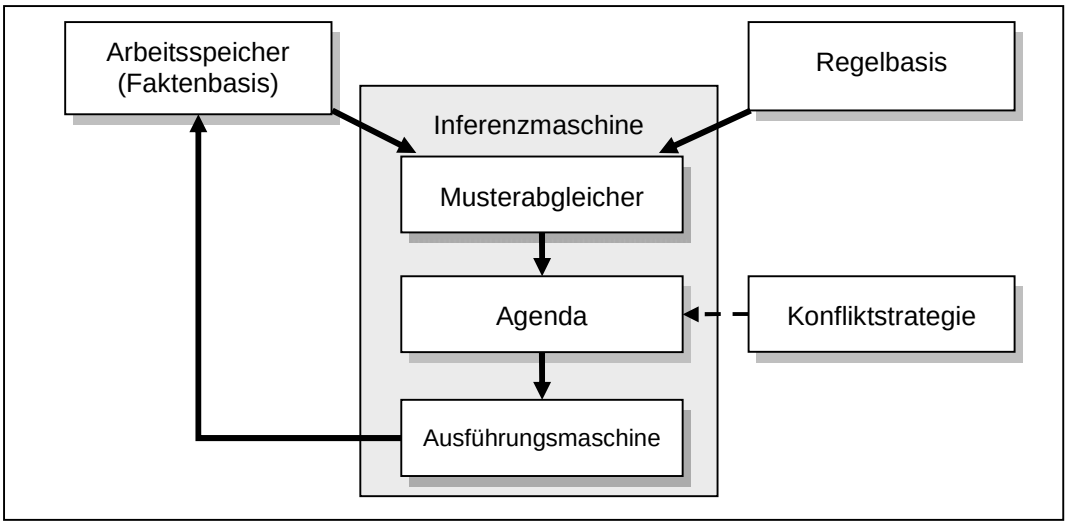


Abbildung 15: Architektur eines regelbasierten Systems.

In klassischen regelbasierten Systemen besteht die Konklusion ausschließlich aus dem Hinzufügen, Löschen oder Modifizieren von Fakten im Arbeitsspeicher. Moderne Regelmaschinen erlauben das Einbinden beliebiger Funktionen einer anderen Programmiersprache in der Konklusion. Wurde eine Regel ausgeführt, ist es erforderlich, die Agenda neu zu berechnen, denn durch das Hinzufügen, Löschen oder Modifizieren von Fakten können neue Regeln aktiviert oder bereits aktivierte deaktiviert werden. Dieser Vorgang wird solange ausgeführt, bis keine Regel mehr feuern kann. Für das Aktualisieren der Agenda wurden effiziente Algorithmen entwickelt, wie z.B. der *Rete-Algorithmus* (s. [52], S. 136 - 146). Es gibt zwei grundsätzliche Verfahren bei der Anwendung von Regeln:

- Vorwärtsverkettung (engl.: *forward chaining*) oder auch datengetrieben: es wird versucht, auf Grundlage von Fakten eine Diagnose zu stellen. Wenn die Prämisse der Regel, basierend auf den aktuellen Fakten des Arbeitsspeichers, erfüllt ist, wird der Aktionsteil ausgeführt (unter Berücksichtigung der Konfliktstrategie).

- Rückwärtsverkettung (engl.: *backward chaining*) oder auch zielgetrieben: es wird versucht, ein Hypothese zu beweisen. Ausgehend von einem herzuleitenden Fakt werden Regeln ermittelt, deren Aktionsteil den Fakt erzeugen würde. Im nächsten Schritt werden die in dem Bedingungsteil einer solchen Regel auftretenden Fakten zu den herzuleitenden Fakten (dies entspricht der Vorgehensweise eines Prolog-Interpreters).

4.3.2 Varianten regelbasierter Systems

In [71] wird zwischen drei grundsätzlichen Varianten regelbasierter System unterschieden (die ihrerseits wieder in verschiedenen Varianten auftreten können):

- Regelbasierte Systeme mit Aussagenlogik,
- Regelbasierte Systeme mit Attributlogik,
- Regelbasierte Systeme mit Prädikatenlogik erster Ordnung.

Regelbasierte Systeme mit Aussagenlogik. Die allgemeine Form von aussagenlogischen Regeln ist:

$$p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow h\,, \tag{4-7}$$

wobei $p_1 \wedge p_2 \wedge ... \wedge p_n$ und h Literale der Aussagenlogik (s. [92], [95]) beschreiben. Um dynamische Veränderungen in der Faktenbasis zu modellieren, werden zwei zusätzliche Basisoperationen benötigt:

- $retract(q)$ – löscht den Fakt q aus der Faktenbasis,
- $assert(q)$ – trägt den Fakt q in die Faktenbasis ein.

Damit sieht das Schema der Regel dann folgendermaßen aus:

$$p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow retract(d_1, d_2, ...d_i), assert(h_1, h_2, ..., h_j)\,. \tag{4-8}$$

Regeln dieser Art werden auch *Produktionsregeln* genannt. Im Allgemeinen ist hier der Inferenzprozess nicht mehr zwangsläufig *monoton*. Als monoton wird ein Inferenzprozess bezeichnet, wenn ableitete neue Fakten nicht im Widerspruch zu der vorherigen Faktenmenge stehen können.

Regelbasierte Systeme mit Attributlogik. Regelbasierte Systeme, die auf der Aussagenlogik basieren, sind für komplexe Anwendungen nicht ausdrucksstark genug. Viele verfügbare Regelmaschinen basieren auf der *Attributlogik*, so auch die in dieser Arbeit verwendete Java-basierte Regelmaschine Jess (*Java Expert System Shell*, s. [9], [52]) oder das regelbasierte Werkzeug für Geschäftsprozesse *Drools* (s. [4]).

Die Grundidee ist, dass Attributen atomare (einzelne) oder eine Menge von Werten zugewiesen werden können. Sei U eine endliche, nichtleere Menge von Ob-

jekten (das Universum) und A eine Menge von Attributen. Jedes Attribut $A_i \in A$ ist eine (partielle) Funktion der Form

$$A_i : U \to D_i , \tag{4-9}$$

wobei D_i (Domänen) zulässige Werte von A_i beschreiben. Ein generalisiertes (partielles) Attribut hat die Form

$$A_i : U \to 2^{D_i} , \tag{4-10}$$

wobei 2^{D_i} die Potenzmenge von D_i darstellt. In der sogenannten *Attributlogik* (engl.: *Set Attributive Logic*) kann für A_i gelten: $A_i = d$, $A_i = t$ oder $A_i \in t$, wobei $d \in D$ ein einzelner Wert und $t = \{d_1, d_2, ..., d_k\}$, $t \subseteq D$ eine Menge (ein Set) dieser Werte ist. $A_i = t$ bedeutet, dass dem Attribut A_i alle Werte aus t zugewiesen werden, bei $A_i \in t$ sind es nur einige Werte. In der erweiterten Form können die Regeln einer Attributlogik folgende Komponenten enthalten (s. [71]):

- einen eindeutigen Regelbezeichner (ID),
- Spezifikation eines Kontextes, in dem eine Regel angewandt werden kann,
- Vorbedingungen der Regel, welche durch logische Formeln spezifizieren, wann die Regel erfüllt ist,
- dynamische Operationen *retract* und *assert*,
- Konklusion,
- Kontrollstrukturen, wie *if ...then...else* .Die generische Form einer Regel dieser Art sieht wie folgt aus:

$$\begin{aligned} &rule : \psi \wedge \\ &\quad A_1 \in t_1 \wedge A_2 \in t_2 \wedge ... \wedge A_n \in t_n \\ &\quad \to \\ &\quad retract(B_1 = b_1, B_2 = b_2, ..., B_i = b_i) \\ &\quad assert(C_1 = c_1, C_2 = c_2, ..., C_j = c_j) \\ &\quad H_1 = h_1, H_2 = h_2, ..., H_k = h_k \\ &\quad (Kontrollstrukturen), \end{aligned} \tag{4-11}$$

wobei

- ψ den Kontext spezifiziert, in dem die Regel angewandt werden soll,
- $A_1 \in t \wedge ... \wedge A_n \in t_n$ die Vorbedingung ist,
- $B_1 = b_1,...,B_i = b_i$ spezifiziert, welche Fakten aus dem Arbeitsspeicher gelöscht werden,
- $C_1 = c_1,...,C_j = c_j$ spezifiziert, welche Fakten dem Arbeitsspeicher hinzugefügt werden,
- $H_1 = h_1,...,H_k = h_k$ spezifiziert alle Arten von Aktionen, die in der Konklusion ausgeführt werden können (beispielsweise auch Funktionsaufrufe einer anderen Programmiersprache).

mit den Attributen $B_1,...,B_i,C_1,...,C_j,H_1,...,H_k \in A$.

Regelbasierte Systeme mit Prädikatenlogik erster Ordnung. Die allgemeine Form von Regeln der Prädikatenlogik erster Ordnung ist:

$$q_1 \wedge q_2 \wedge ... \wedge q_n \rightarrow h, \tag{4-12}$$

wobei $q_1 \wedge q_2 \wedge ... \wedge q_n$ und h Literale der Prädikatenlogik erster Ordnung sind. Da Regeln dieser Art in der hier vorgestellten Arbeit nicht verwendet werden, wird auf eine ausführliche Darstellung verzichtet (für weitere Details, siehe [71]).

5 Realzeit-Szeneninterpretation mit ontologiebasierten Regeln

In diesem Kapitel wird gezeigt, wie die in Kapitel 4 vorgestellten Mechanismen zur Entwicklung eines generischen Rahmenwerks für die wissensbasierte Szeneninterpretation genutzt werden können. Die Techniken werden anhand von Beispielen aus der Domäne der Flughafenvorfeldaktivitäten verdeutlicht, welche Gegenstand des in der Einleitung erwähnten EU-Projekts Co-Friend war. Dazu wird in Abschnitt 5.1 zunächst auf die Repräsentation von Aktionsmodellen in einer OWL Ontologie mit SWRL-Erweiterung eingegangen. Diese Aktionsmodelle repräsentieren die Handlungsabläufe, die das Interpretationssystem erkennen soll. In Abschnitt 5.2 wird dargestellt, wie aus einer derartigen Ontologie automatisch Regeln für das regelbasierte Szeneninterpretationssystem SCENIOR (*SCENe Interpretation with Ontology-based Rules*) generiert werden. Abschnitt 5.3 beschreibt den Interpretationsprozess, bei dem das System diese Regeln auf Evidenzen anwendet, die von der Mittelschicht geliefert werden, um schließlich eine Interpretation der Szene abzuliefern. Weiterhin werden Betrachtungen zur Komplexität und zum Determinismus des Interpretationsprozesses angestellt. In Abschnitt 5.4 wird verdeutlicht, wie dieser Interpretationsprozess durch ein probabilistisches Präferenzmodell gesteuert wird.

5.1 Modellierung von Aktionen

Wie bereits in Abschnitt 1.1 erwähnt, ist der Begriff „Szeneninterpretation“ sowohl für die Analyse statischer Bilder, als auch für dynamische Bildsequenzen zu verstehen, bei denen die Aufgabe darin besteht, Handlungsabläufe (auch *Aktionen* oder *Ereignisse* genannt) zu erkennen. Da sich jedoch diese Arbeit schwerpunktmäßig mit der Interpretation von Videosequenzen befasst und auch die damit zusammenhängende Verarbeitung zeitlicher Constraints beinhaltet, werden die Mechanismen anhand von zu erkennenden Aktionen erläutert. Die vorgestellten Techniken können aber auch, zumindest teilweise oder mit entsprechenden Modifikationen, für die statische Bildanalyse verwendet werden (s. auch Abschnitt 6.4).

5.1.1 Aggregatrepräsentation

In [92] werden der *Situationskalkül* und der *Ereigniskalkül* beschrieben. Im Situationskalkül werden Aktionen als Übergänge zwischen Situationen definiert. Der Ereigniskalkül verwendet Zeitpunkte statt Situationen, um Handlungsabläufe zu beschreiben. Sicherlich wären diese Kalküle geeignet, Ereignisse in Videosequenzen zu modellieren. Der generische Anspruch des hier vorgestellten Ansatzes begründet jedoch die Verwendung der abstrakten Struktur des Aggregats zur Repräsentation von Bildobjekten oder Ereignissen, das bereits in Abschnitt 2.3.1 vorgestellt wurde:

$$\begin{aligned}&\texttt{Aggregate\_Concept} \equiv \texttt{Parent\_Concept} \sqcap \\ &\quad \exists_{=1}\texttt{has-part}_1.\texttt{Part\_Concept}_1 \sqcap \ldots \sqcap \\ &\quad \exists_{=1}\texttt{has-part}_k.\texttt{Part\_Concept}_k \sqcap \\ &\quad \texttt{conceptual constraints}\end{aligned} \tag{5-1}$$

Zwar würde OWL DL mehrere Elternkonzepte, also Mehrfachvererbung, zulassen, die beabsichtigte Transformation in die Java-basierte Regelsprache Jess erlaubt hingegen nur Einfachvererbung. Die Anzahlrestriktion der `has-part`-Relationen ist immer eins; nur durch die Verwendung verschiedener Rollennamen für verschiedene Teile kann in den SWRL-Regeln gezielt auf bestimmte Teile zugegriffen werden, um Constraints zu formulieren. Die linke und rechte Seite sind äquivalent, d.h., aus der Existenz des Aggregatkonzepts kann auf die Existenz der Teile und erfüllter Constraints geschlossen werden, und umgekehrt.

Beispiel 5-1: Das Aggregat Refuelling

Abbildung 16 zeigt die kompositionelle Struktur des Aggregats `Refuelling`, welches einen Betankungsvorgang des Flugzeugs repräsentiert. Der erste Teil, `Tanker-Positioning`, beschreibt das als punktuell modellierte *primitive Ereignis*, dass das Tankfahrzeug in die Tankzone einfährt. Die Tankzone ist ein vordefinierter Bereich, in dem sich das Tankfahrzeug während des Betankens aufhält. Angelehnt an die Arbeiten von Fusier et al. [53] werden vordefinierte Zonen für die Lokalisierung von Objekten verwendet. In Abbildung 17 sind die Zonen des Flughafenvorfelds dargestellt. Der *primitive Zustand* `Tanker-Positioned` beschreibt, dass sich der Tanker in der Tankzone positioniert hat. Zustände sind nicht punktuell, sondern über ein bestimmtes Zeitintervall ausgedehnt. *Primitive* Ereignisse und Zustände werden (im Gegensatz zu *zusammengesetzten* Ereignissen und Zuständen) von der Mittelschicht geliefert (s. Abbildung 1). Sie können direkt durch die niedere Bildverarbeitungsschicht berechnet werden.

Der primitive Zustand `Fuelling` stellt das eigentliche Betanken dar; er muss während des Zeitintervalls von `Tanker-Positioned` stattfinden. Das primitive Ereignis `Tanker-Leaves-Tanking-Zone` beschreibt schließlich, dass das Tankfahrzeug die Tankzone verlässt.

Die meisten primitiven Zustände und Ereignisse werden aus Objekttrajektorien generiert, die das Flugzeugvorfeld durch stationäre Weitwinkelkameras erfassen. Es gibt jedoch auch bestimmte Ereignisse – sogenannte `Special-Vision-Tasks` (SVTs) – die robuster durch dynamische PTZ-Kameras (engl.: *pan-tilt-zoom*) erfasst werden können. PTZ-Kameras können durch Schwenken, Neigen und Zoomen Objekte und Vorgänge gezielter erfassen, z.B. das Anschließen des Tankstutzens an das Flugzeug (`Fuelling`) oder die Bewegungsrichtung des Gepäcks auf dem Förderband beim Entladen bzw. Beladen. Aufgrund der Tatsache, dass bis zum Ende des Projekts nur Eingabedaten ohne SVTs für Experimente zur Verfügung standen (s. Abschnitt 7.1.2), werden sie in den weiteren Betrachtungen nicht mehr aufgeführt.

Auf die Begriffe „Ereignis“, „Zustand“ und ihre ontologische Einordnung wird in Abschnitt 5.1.2 noch detaillierter eingegangen.

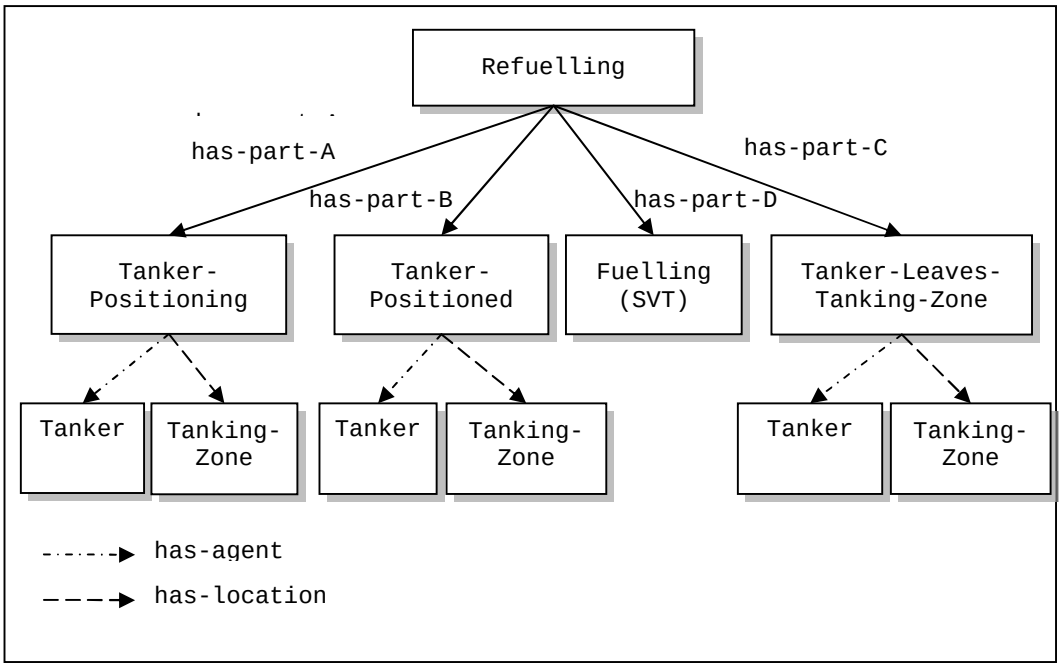


Abbildung 16: Kompositionelle Struktur des Aggregats `Refuelling`.

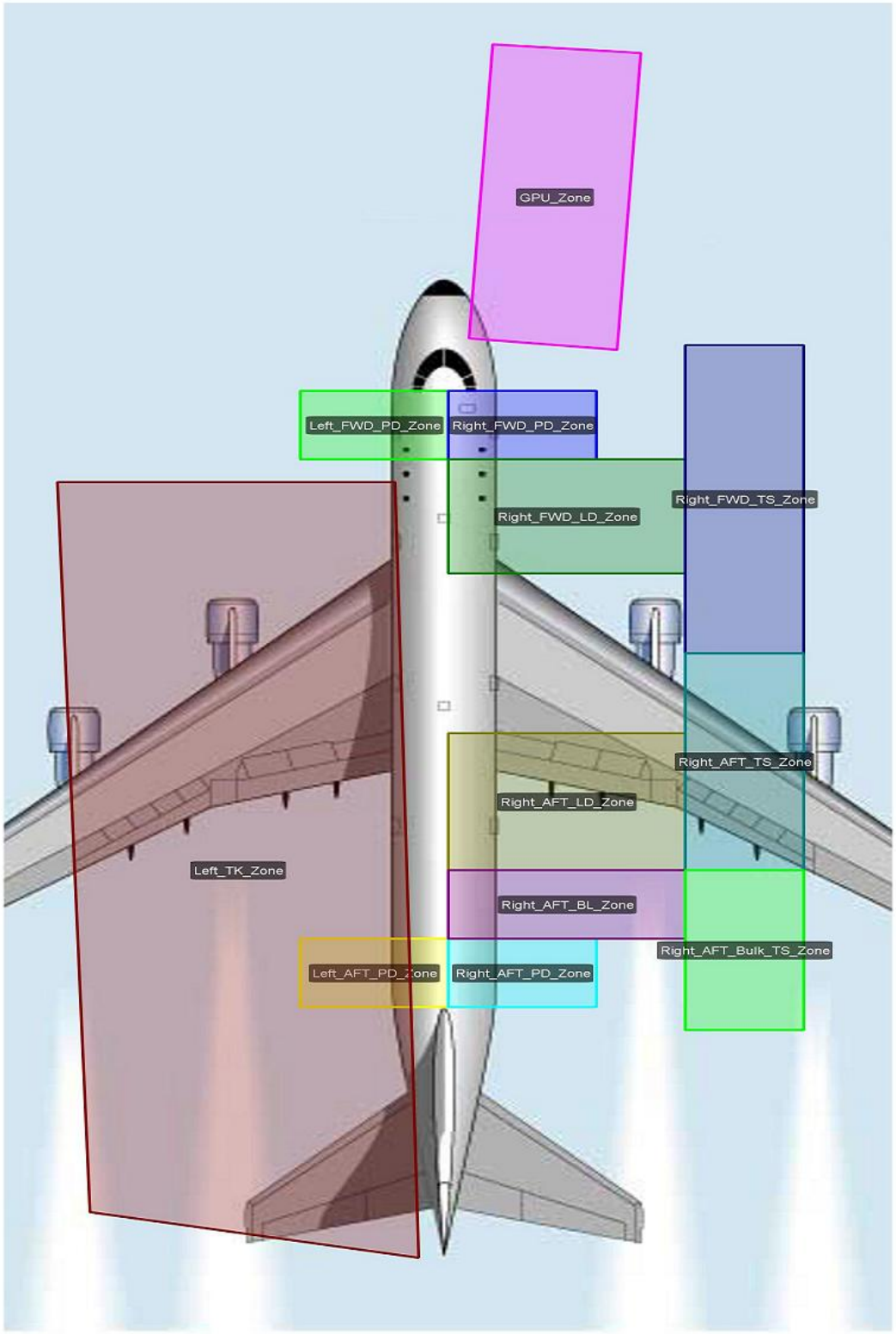


Abbildung 17: Vordefinierte Zonen für die Lokalisation von Objekten (aus [40]).

In der Schreibweise von OWL stellt sich der taxonomische und kompositionelle Teil des Aggregats `Refuelling` folgendermaßen dar (ohne `Fuelling`):

```
Refuelling⊑Composite-Event⊓                                    (5-2)
           has-part-A exactly 1 Tanker-Positioning⊓
           has-part-B exactly 1 Tanker-Positioned⊓
           has-part-C exactly 1 Tanker-Leaves-Tanking-Zone
```

Primitive Ereignisse oder Zustände sind nicht aus weiteren Ereignissen oder Zuständen (sogenannten *konzeptuellen Objekten*) zusammengesetzt, können aber *physikalische Objekte*, wie z.B. Zonen oder Fahrzeuge enthalten. In diesem Beispiel besitzen die drei Teile jeweils noch ein Fahrzeug (auch *Agent* genannt) `Tanker` und eine Zone `Tanking-Zone`.

Konzeptuelle Constraints. Zwischen den Teilen gelten konzeptuelle Constraints; in der Domäne der Flughafenvorfeldaktivitäten sind dies zeitliche Constraints und *Identitäts-Constraints*. Die zeitlichen Constraints im vorliegenden Beispiel legen fest, dass das Ereignis `Tanker-Positioning` vor dem Zustand `Tanker-Positioned` und der wiederum vor dem Ereignis `Tanker-Leaves-Tanking-Zone` stattfinden muss. Die Identitäts-Constraints stellen sicher, dass es sich bei allen drei Teilen um dasselbe Fahrzeug und um dieselbe Zone handelt. Diese konzeptuellen Constraints lassen sich in OWL DL nicht formulieren, da Rollen *zweistelligen* Prädikaten entsprechen (s. Abschnitt 4.1.1), d.h., Rollen können nur Beziehungen zwischen dem Aggregatkonzept und jeweils einem seiner Teile ausdrücken, nicht jedoch *zwischen* den Teilen. Nun könnte versucht werden, dieses Problem durch geschickte *Reifikation*, d.h. durch „Vergegenständlichung“ zu lösen. Dabei wird eine Beziehung in einem Konzept vergegenständlicht, beispielweise ließe sich ein Konzept `A-vor-B` mit den Teilen `A` und `B` definieren. Für zwei Individuen (Aktionen) x und y mit $A(x)$ und $B(y)$ wäre dann die Interpretation, dass x zeitlich vor y stattgefunden hat. Bei komplizierten Zusammenhängen zeitlicher Constraints und in Verbindung mit Identitäts-Constraints, wie sie in dieser Domäne vorkommen, würde dies jedoch zu einer Vielzahl von (für Menschen) schwer verständlichen Konzepten führen. Das widerspricht aber dem Grundgedanken einer Ontologie, welche auch dem Austausch von Wissen und als Schnittstelle zu Experten der jeweiligen Domäne dienen soll. Reifikation wird jedoch für die Modellierung räumlicher Relationen benutzt: Wenn ein Individuum Element der Klasse `Tanker-Positioned` ist, dann impliziert dies die räumliche Beziehung, dass sich der Tanker innerhalb der Tankzone aufhält, d.h., die räumliche Relation `inside(Tanker,Tanking-Zone)` ist in der Klasse vergegenständlicht worden.

Die konzeptuellen Constraints werden durch SWRL-Regeln (s. Abschnitt 4.2.3) repräsentiert. Wie bereits erwähnt, kann die Verwendung von SWRL-Regeln zur

Unentscheidbarkeit der Wissensbasis führen, sofern sie nicht DL-sicher sind, d.h. nur auf Fakten der ABox angewandt werden. Einerseits ist dies der Fall, denn die Regeln des regelbasierten Systems (welche basierend auf den SWRL-Regeln generiert werden, s. Abschnitt 5.2), werden nur auf Evidenzen angewandt, die durch niedere Bildverarbeitungskomponenten und die Mittelschicht geliefert werden. Andererseits existiert (zum Zeitpunkt der Entwicklung dieser Arbeit) nach meinem Kenntnisstand kein DL-Reasoner, der SWRL-Regeln für Konsistenzüberprüfungen der Wissensbasis evaluiert – dafür wäre die Entscheidbarkeit ausschlaggebend.

SWRL wird durch den Protégé-Editor unterstützt, so dass ein gewisses Maß an Konsistenz gewährleistet ist. Es ist beispielsweise nicht möglich, eine Regel mit Klassen zu definieren, welche nicht in der Wissensbasis enthalten sind, oder Variablen in der Konklusion zu verwenden, die nicht in der Prämisse eingeführt wurden. Die SWRL-Regel zur Beschreibung der konzeptuellen Constraints für das Aggregat `Refuelling` sieht folgendermaßen aus:

```
(1)   Refuelling(?ref) ^                                                    (5-3)
(2)   has-part-A(?ref, ?tanker-pos) ^
(3)   has-part-B(?ref, ?tanker-posed) ^
(4)   has-part-C(?ref, ?tanker-leaves) ^
(5)   has-start-time(?ref, ?ref-st) ^
(6)   has-finish-time(?ref, ?ref-ft) ^

(7)   has-agent(?tanker-pos, ?a1) ^
(8)   has-location(?tanker-pos, ?l1) ^
(9)   has-start-time(?tanker-pos, ?tanker-pos-st) ^
(10)  has-finish-time(?tanker-pos, ?tanker-pos-ft) ^

(11)  has-agent(?tanker-posed, ?a2) ^
(12)  has-location(?tanker-posed, ?l2) ^
(13)  has-start-time(?tanker-posed, ?tanker-posed-st) ^
(14)  has-finish-time(?tanker-posed, ?tanker-posed-ft) ^


(15)  has-agent(?tanker-leaves, ?a3) ^
(16)  has-location(?tanker-leaves, ?l3) ^
(17)  has-start-time(?tanker-leaves, ?tanker-leaves-st) ^
(18)  has-finish-time(?tanker-leaves, ?tanker-leaves-ft)

      ->
(19)  equal(?a1, ?a2) ^
(20)  equal(?a2, ?a3) ^
(21)  equal(?l1, ?l2) ^
(22)  equal(?l2, ?l3) ^
(23)  equal(?ref-st, ?tanker-pos-st) ^
(24)  equal(?ref-ft, ?tanker-leaves-ft) ^
(25)  min-before(?tanker-pos-ft, ?tanker-posed-st, 0) ^
(26)  min-before(?tanker-posed-ft, ?tanker-posed-st, -3600000) ^
(27)  min-before(?tanker-posed-ft, ?tanker-leaves-st, 0)
```

Dabei werden die SWRL-Regeln nicht logisch interpretiert, sondern dienen lediglich der Zuweisung von Variablennamen in der Prämisse (Zeile 1 - 18) und der Formulierung von Constraints in der Konklusion (Zeile 19 - 27). Über die `has-part`-Relation wird auf die Teile (Zeile 2 - 4) und auf die Start- und Endzeit (Zeile 5, 6) des Aggregatkonzepts zugegriffen. Dann wird jeweils auf den Agenten (hier das Fahrzeug), die Zone und die Start- und Endzeit der Teile zugegriffen (Zeile 7 - 18). Jedes konzeptuelle Objekt hat eine Start- und Endzeit (aus Gründen der Vereinheitlichung und einfacherer Implementierung haben auch Ereignisse eine Start- und Endzeit, welche immer identisch sind). Die Zeilen 19 und 20 in der Konklusion drücken aus, dass es sich bei allen Teilen um dasselbe Fahrzeug handeln muss, analog wird in Zeile 21 und 22 ausgedrückt, dass die Zone dieselbe sein muss. In den Zeilen 25 bis 27 sind die zeitlichen Constraints dargestellt: Zeile 25 formuliert, dass `Tanker-Positioning` vor `Tanker-Positioned` stattfinden muss, Zeile 27 entsprechend, dass sich `Tanker-Positioned` vor `Tanker-Leaves-Tanking-Zone` ereignen muss. Zeile 26 besagt, dass sich das Tankfahrzeug bei einem korrekten Ablauf nicht länger als 60 Minuten (Angabe in Millisekunden) in der Tankzone aufhält.

Die `min-before`-Funktion ist eine selbstdefinierte Built-in-Funktion (s. Abschnitt 4.2.3) zur kompakten Darstellung einer Ungleichung der Form:

$$T_i - T_k \leq c_{ik}\,, \tag{5-4}$$

mit den Zeitpunkten T_i, T_k und der Konstante c_{ik}. Diese Ungleichungen bilden die Grundlage für die hier verwendete *konvexe Zeitpunktalgebra*, die in Abschnitt 5.2.3 erörtert wird. Zwar können zeitliche Constraints auch mit der verfügbaren *SWRLTemporalOntology* [14] modelliert werden, sie basiert jedoch auf *Allens Intervallalgebra* [24] und eignet sich daher nur zur Beschreibung qualitativer zeitlicher Beziehungen. Die in dieser Arbeit im Fokus stehenden Domänen erfordern dagegen die Möglichkeit der quantitativen Verarbeitung zeitlicher Constraints. Eine Ontologie für die konvexe Zeitpunktalgebra ist meines Wissens nicht verfügbar. Bei der Transformation der Ontologie in die Regelbasis wird aus den zeitlichen Constraints der SWRL-Regeln ein zeitliches Constraint-Netz generiert (s. Abschnitt 5.2.3).

SWRL-Regeln sind sicherlich für einen Experten der jeweiligen Domäne nicht leicht verständlich, sofern er nicht über Programmierkenntnisse verfügt. Dies ist ein Makel, der akzeptiert werden muss, denn zum Formulieren der konzeptuellen Constraints sind SWRL-Regeln gut geeignet. Im Sinne der Benutzerfreundlichkeit wären hier Lösungen denkbar, die den Fachexperten bei der Erstellung unterstützen, z.B. Benutzungsoberflächen und Werkzeuge, die dann die Regeln automatisch generieren.

5.1.2 Top-Level-Ontologie

Die Top-Level-Ontologie beschreibt den dömenenunabhängigen Teil der Ontologie (s. Abschnitt 4.2), der generelle Konzepte zur Modellierung von Handlungsabläufen bereitstellt (s. Abbildung 18).

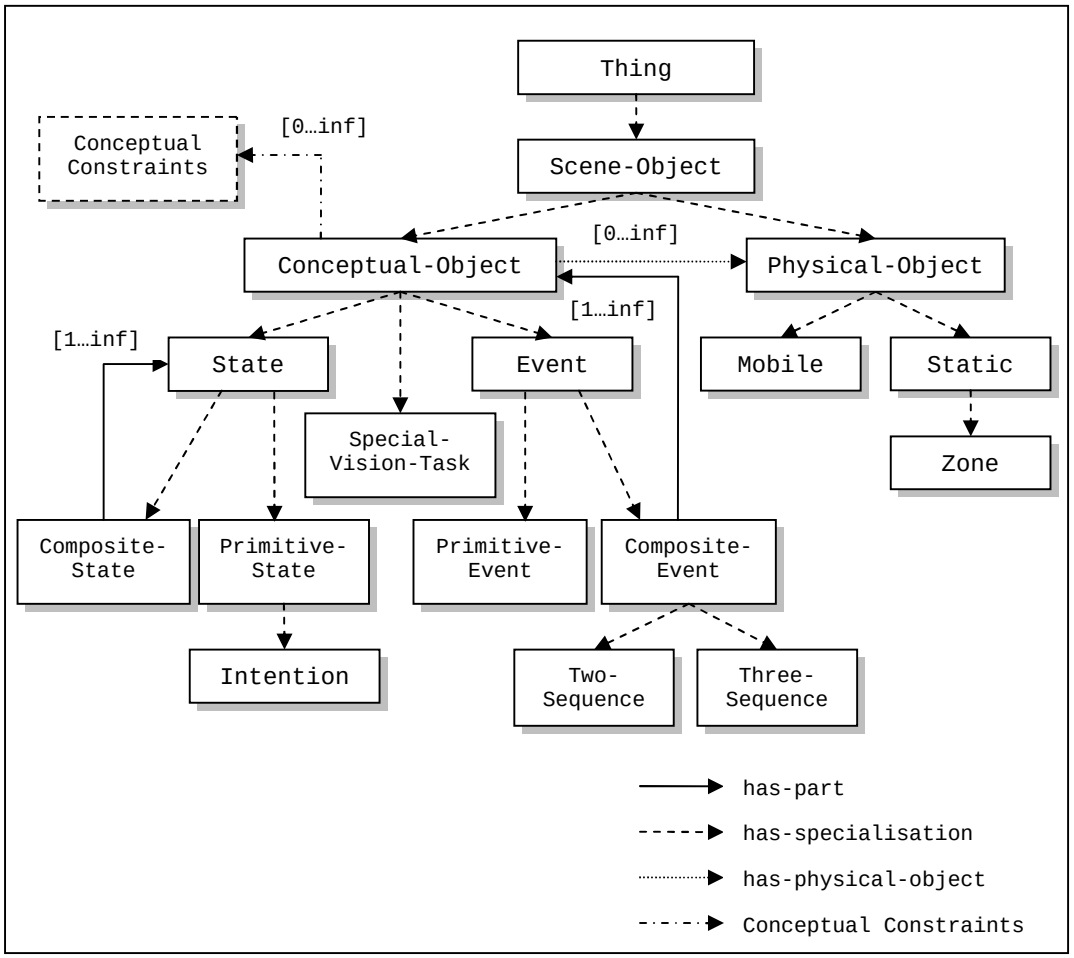


Abbildung 18: Top-Level-Ontologie für die Szeneninterpretation.

Die Wurzelklasse jeder Ontologie ist `Thing`. Davon leitet sich die Klasse `Scene-Object` ab. Sie repräsentiert alle Konzepte, die für die Szeneninterpretation benötigt werden, und ist unterteilt in die Konzepte `Conceptual-Object` und `Physical-Object`. Das Konzept `Conceptual-Object` repräsentiert Aktionen, die entweder Ereignisse `(Event)` oder Zustände `(State)` sein können. Ein Zustand ist eine räumlich-zeitliche Eigenschaft eines physikalischen Objekts (oder mehrerer), welche in einem Zeitintervall und allen Teilintervallen wahr ist. Diese Eigenschaft wird auch als *durativ* bezeichnet (s. [51]). Beispiel für Zustände sind `Ve-`

hicle-Inside-Zone oder Tanker-Positioned. Ein zusammengesetzter Zustand (Composite-State) ist eine Kombination mehrerer Zustände. Ein Beispiel hierfür ist der zusammengesetzte Zustand Vehicle-Stopped-Inside-Zone; er besteht aus den primitiven Zuständen Vehicle-Inside-Zone (viz) und Vehicle-Stopped (vs), wobei (vs) während (vis) stattfinden muss. Ein primitives Ereignis (Primitive-Event) ist zeitlich punktuell. Oftmals bezeichnet es den Übergang von einem Zustand in einen anderen, z.B. kann das Ereignis Vehicle-Enters-Zone als Übergang von Vehicle-Outside-Zone und Vehicle-Inside-Zone aufgefasst werden. Zusammengesetzte Ereignisse (Composite-Event) können Zustände und Ereignisse als Teile haben; sie sind daher in der Regel nicht punktuell. Ein Beispiel für ein zusammengesetztes Ereignis ist das oben beschriebene Aggregat Refuelling.

Primitive Zustände und Ereignisse haben keine konzeptuellen Objekte als Teile, sie können jedoch physikalische Objekte besitzen (die has-part-Relation beschränkt sich aber auf konzeptuelle Objekte, während für physikalische Objekte spezielle Rollen, wie z.B. has-agent oder has-location verwendet werden. Ihre Instanzen können direkt durch niedere Bildverarbeitungsprozesse und die Mittelschicht berechnet werden.

Die Ontologie enthält außerdem das eher ungewöhnliche Konzept Intention, welches hier nur der Vollständigkeit halber aufgeführt ist. Es bezieht sich auf den *mentalen Zustand* einer Person. Mentale Zustände können Ursachen von Handlungen sein, z.B. könnte der Zustand „Kopfschmerzen“ die Ursache dafür sein, dass eine Person eine Aspirintablette nimmt. Auch wenn mentale Zustände natürlich nicht direkt beobachtet werden können, so lassen sie sich evtl. aus Beobachtungen ableiten und für Vorhersagen oder zum Auflösen von Mehrdeutigkeiten nutzen. Mechanismen, die das Konzept der mentalen Zustände verwenden, wurden aufgrund von Zeitmangel und anderen Schwierigkeiten im Projekt Co-Friend nicht weiter verfolgt und auch in dieser Arbeit nicht näher untersucht.

Ein weiteres unübliches Konzept ist Zone, welches eine qualitative Lokalisierung durch ein konvexes Polygon in der Vorfeldebene beschreibt. Diese vordefinierten Zonen spielen eine wichtige Rolle in der Modellierung von Ereignissen und Zuständen, wie z.B. in Tanker-Positioning, Aircraft-Positioned, etc. Dieses Konzept ist insofern ungewöhnlich, da es für jede Zone nur *ein* Individuum gibt.

Die Konzepte Two-Sequence und Three-Sequence sind abstrakte, häufig auftretende Handlungsmuster, welche als Elternkonzepte für spezifischere Modelle dienen, so dass Vererbung ausgenutzt werden kann. Two-Sequence steht für ein Ereignis, welches aus zwei (nicht notwendigerweise unmittelbar) aufeinanderfolgenden Zuständen oder Ereignissen besteht, Three-Sequence entsprechend aus

drei. Das Aggregat `Two-Sequence` definiert sich wie folgt (`Three-Sequence` entsprechend):

```
Two-Sequence⊑Composite-Event⊓                                    (5-5)
             has-part-A exactly 1 Conceptual-Object⊓
             has-part-B exactly 1 Conceptual-Object
```

Durch eine SWRL-Regel werden die Start- und Endzeit von `Two-Sequence` definiert und gewährleistet, dass das erste Ereignis bzw. der erste Zustand endet, bevor das zweite Ereignis bzw. der zweite Zustand beginnt:

```
Two-Sequence(?ts) ^                                              (5-6)
has-part-A(?ts, ?co1) ^
has-part-B(?ts, ?co2) ^
has-start-time(?ts, ?ts-st) ^
has-finish-time(?ts, ?ts-ft) ^

has-start-time(?co1, ?co1-st) ^
has-finish-time(?co1, ?co1-ft) ^

has-start-time(?co2, ?co2-st) ^
has-finish-time(?co2, ?co2-ft)

->
equal(?ts-st, ?co1-st) ^
equal(?ts-ft, ?co2-ft) ^
min-before(?co1-ft, ?co2-st, 0)
```

Ein Aggregat `Aircraft-Arrival`, welches aus den Teilen `Aircraft-Positioning` und `Aircraft-Positioned` besteht, kann nun definiert werden als:

```
Aircraft-Arrival⊑Two-Sequence⊓                                   (5-7)
         has-part-A exactly 1 Aircraft-Positioning⊓
         has-part-B exactly 1 Aircraft-Positioned
```

In der SWRL-Regel müssen nun lediglich noch die Identitäts-Constraints formuliert werden:

```
Aircraft-Arrival(?aa) ^                                                    (5-8)
has-part-A(?aa, ?aircraft-pos) ^
has-part-B(?aa, ?aircraft-posed) ^

has-agent(?aircraft-pos, ?a1) ^
has-location(?aircraft-pos, ?l1) ^

has-agent(?aircraft-posed, ?a2) ^
has-location(?aircraft-posed, ?l2) ^

->
equal(?a1, ?a2) ^
equal(?l1, ?l2)
```

Weitere Konzepte abstrakter Handlungsmuster in der Domänen-Ontologie können auch Identitäts-Constraints beinhalten, wie z.B. das Konzept `Visit` (s. Abschnitt 5.1.3).

Für SWRL-Regeln gibt es ursprünglich keine Mechanismen, die Vererbung unterstützen. Die hier gezeigten Techniken zur Ausnutzung von Vererbung durch die Definition abstrakter Handlungsmuster wird erst durch den Transformationsprozess in das regelbasierte System ermöglicht (s. Abschnitt 5.2).

Es ist noch anzumerken, dass es in der Ontologie kein Konzept `View` gibt, welches das Erscheinungsbild eines Objekts repräsentiert. In der Arbeit von Hotz et al. [65] zur Erkennung von Strukturen in Häuserfassaden beispielsweise wurden Instanzen von 2D-Objekt-Erscheinungen als Eingabe der höheren Bildinterpretation verwendet. In der hier beschriebenen Arbeit hingegen basiert die Interpretation auf detektierten 3D-Objekten ohne spezielle Erscheinungsformen.

Weiterhin sei angemerkt, dass die hier vorgestellte Top-Level-Ontologie keinen Anspruch auf Vollständigkeit erhebt. Die Entwicklung von Ontologien – das hat auch das Projekt Co-Friend gezeigt – ist ein evolutionärer Prozess, ähnlich einem Softwareentwicklungsprozess. Durch das Einbeziehen weiterer Domänen werden Modifikationen der Top-Level-Ontologie nötig sein, welche die Qualität und Allgemeinheit der Ontologie verbessern werden.

5.1.3 Domänen-Ontologie

Die Domänen-Ontologie für die Flughafenvorfeld-Domäne besteht aus ca. 80 konzeptuellen Objekten und ca. 40 physikalischen Objekten, welche das Modell einer vollständigen Flugzeugabfertigung beschreiben, repräsentiert durch das

Wurzelkonzept `Turnaround`. Abbildung 20 zeigt eine mögliche kompositionelle Struktur dieses Aggregats (Einrückungen sind als `has-part`-Relationen zu verstehen). Es gibt mehrere Varianten einer Abfertigung, die sich in unterschiedlichen Modellen widerspiegeln: Beispielsweise wird ein Flugzeug nicht bei jeder Abfertigung betankt oder Be- *und* Entladen, z.B. wenn es der erste oder der letzte Flug des Tages ist. Generell kann davon ausgegangen werden, dass in vielen Domänen Variationen von Modellen benötigt werden, um komplexe statische Szenen oder Handlungsabläufe realistisch zu beschreiben. Ein allgemeiner Ansatz, um diesem Problem zu begegnen – die Unterteilung in Teilmodelle – wird in Abschnitt 5.2 vorgestellt. In Abbildung 19 sind einige wichtige Konzepte des Domänenmodells dargestellt.

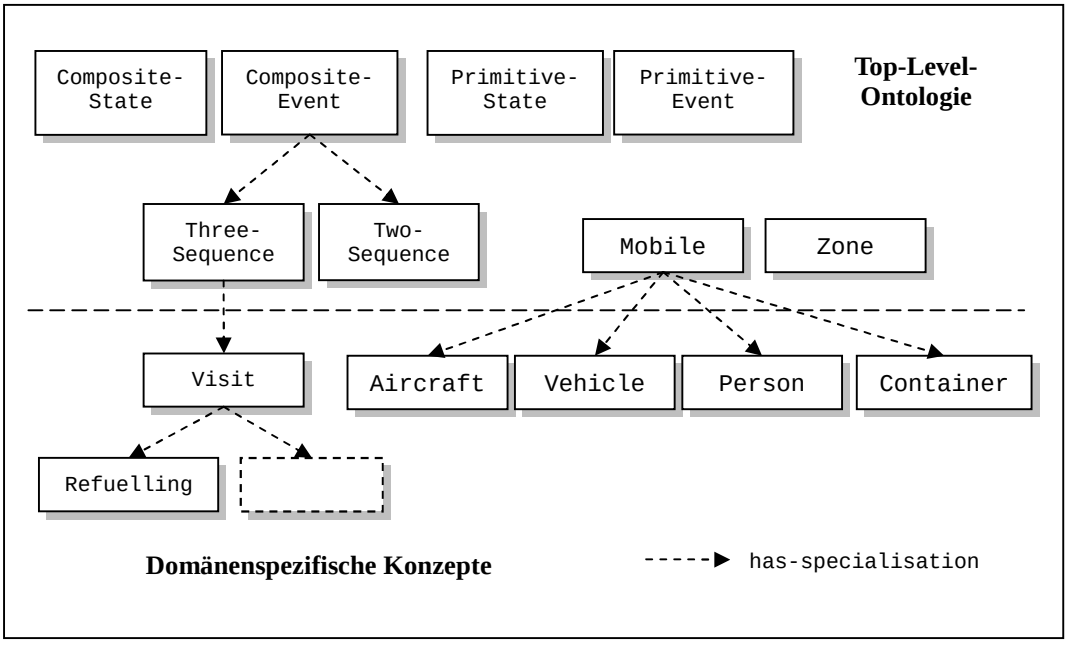


Abbildung 19: Domänenspezifische Konzepte für die Flughafenvorfeld-Domäne.

Das von `Three-Sequence` abgeleitete Konzept `Visit` beschreibt das in dieser Domäne häufig auftretende Handlungsmuster eines Fahrzeugs, welches in eine Zone einfährt, dort Teil eines bestimmten Vorgangs ist und anschließend die Zone wieder verlässt. Die Identitäts-Constraints werden in der SWRL-Regel zu `Visit` definiert, analog zur SWRL-Regel 5-8.

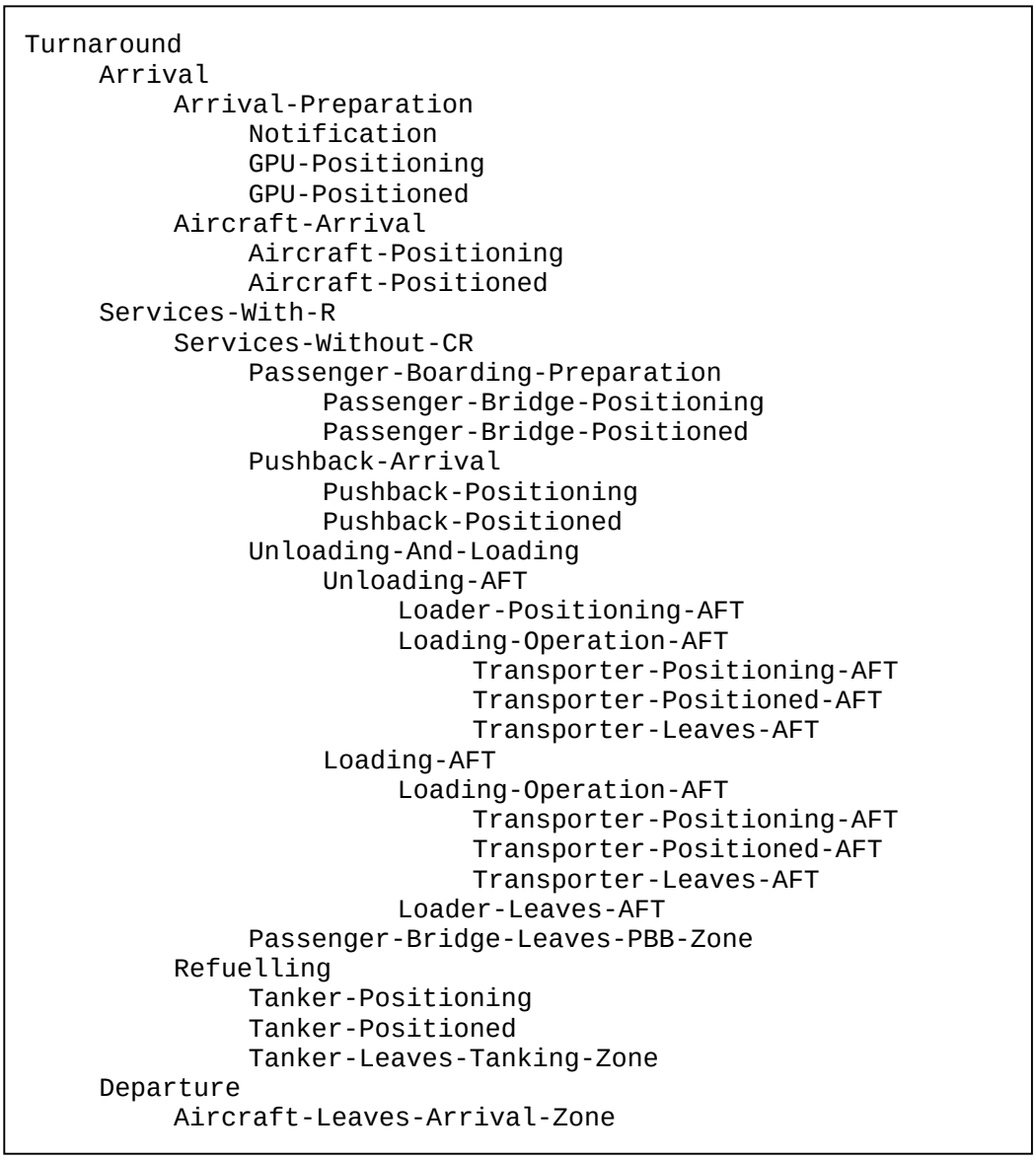


Abbildung 20: Eine mögliche kompositionelle Struktur des Domänenmodells `Turnaround`, welches eine Flugzeugabfertigung repräsentiert.

Das Aggregat `Refuelling` kann nun als Spezialisierung von `Visit` modelliert werden: $\texttt{Refuelling} \sqsubseteq \texttt{Visit}$. Damit muss in der SWRL-Regel nur noch die maximale Dauer definiert werden, und die ursprüngliche SWRL-Regel von `Refuelling` (s. Regel 5-3) vereinfacht sich zu:

```
Refuelling(?ref) ^                                                    (5-9)
has-part-B(?ref, ?tanker-pos) ^
has-start-time(?tanker-pos, ?tanker-pos-st) ^
has-finish-time(?tanker-pos, ?tanker-pos-ft)

->
min-before(?tanker-pos-ft, ?tanker-pos-st, -3600000)
```

Das Konzept `Mobile` hat die Spezialisierungen `Aircraft`, `Vehicle`, `Person` und `Container`. Im aktuellen Modell werden jedoch nur `Aircraft` und `Vehicle` verwendet, da sich die Erkennung einzelner Personen oder Container durch die niederen Bildverarbeitungskomponenten als zu problematisch herausstellte. Gleiches gilt für die Konzepte `Catering`, `Waste-Removel`, `Replace-Drinking-Water` und weitere Konzepte, welche ursprünglich Teil der Ontologie waren. `Vehicle` dient als Elternkonzept für spezifische Fahrzeuge, z.B. GPU (*Ground Power Unit*), Loader, Tanker, etc., `Zone` entsprechend für spezifische Zonen. Die vollständige Taxonomie ist in Anhang A.1 aufgeführt.

Einige Konzepte treten in mehr als einem Aggregat auf, z.B. ist das Konzept `Loading-Operation-AFT` sowohl Teil des Aggregats `Loading-AFT`, als auch des Aggregats `Unloading-AFT`. Dies zeigt das bereits angesprochene grundlegende Problem der Mehrdeutigkeit in der Szeneninterpretation. Zum Teil kann die Mehrdeutigkeit durch Constraints aus dem Kontext aufgelöst werden: Es könnte z.B. sein, dass bestimmte zeitliche Constraints eines Aggregats, welches in der kompositionellen Hierarchie höher als `Loading-AFT` und `Unloading-AFT` steht (beispielsweise `Service`), nur die Zuordnung einer Instanz von `Loading-Operation-AFT` zu `Loading-AFT` *oder* `Unloading-AFT` zulassen. Wenn eine Instanz dennoch mehrfach zugeordnet werden kann, werden beide alternativen Interpretationen parallel verfolgt. Diese Mechanismen werden im nächsten Abschnitt verdeutlicht.

Die hier beschriebenen zeitlichen Constraints stellen nur ein grobes Modell der zeitlichen Zusammenhänge eines Flugzeugabfertigungsvorgangs dar. Grundsätzlich sind feste Constraints oftmals ein Problem: Sind sie zu streng, schließen sie möglicherweise gültige Interpretationen aus, während zu weiche Constraints zu viele ungültige Interpretationen einschließen und damit im ungünstigsten Fall zu falschen Ergebnissen führen oder aber den Suchraum vergrößern. Daher gibt es zusätzlich ein probabilistisches Modell, repräsentiert durch Bayes'sche Kompositionelle Hierarchien (BCHs). Konzeptionell sollten diese probabilistischen Modelle ebenfalls in der Ontologie enthalten sein. Doch zurzeit gibt es nach meinem Kenntnisstand keine effiziente Möglichkeit, probabilistische Verteilungen mit OWL zu beschreiben. Daher sind die probabilistischen Modelle außerhalb der Ontologie repräsentiert und werden zur Laufzeit mit dem Interpretationssystem verknüpft.

5.2 Automatisches Generieren eines Interpretationssystems aus der Ontologie

In diesem Abschnitt wird dargestellt, wie aus einer Ontologie, welche die Modelle der zu erkennenden Handlungsabläufe repräsentiert, automatisch Regeln für das regelbasierte Szeneninterpretationssystem generiert werden. Dazu wird in Abschnitt 5.2.1 zunächst eine Systemübersicht des Interpretationssystems gegeben. Anschließend wird in Abschnitt 5.2.2 beschrieben, wie die Ontologie automatisch in eine Regelbasis für das Interpretationssystem übersetzt wird. In Abschnitt 5.2.2.2 werden *Teilmodelle* eingeführt, die eine effizientere Verarbeitung von Modellvariationen erlauben und in Abschnitt 5.2.2.3 *Hypothesengraphen*, welche die zu erkennenden Aktionen repräsentieren und die Grundlage für den Interpretationsprozess bilden. In Abschnitt 5.2.3 wird im Detail auf das *zeitliche Constraint-Netz* eingegangen. Schließlich wird in Abschnitt 5.2.4 untersucht, inwieweit die Konsistenz und Vollständigkeit der automatisch generierten Regelbasis gewährleistet werden kann.

5.2.1 Systemübersicht

Das Szeneninterpretationssystem SCENIOR ist eine Neuentwicklung, die im Rahmen des Co-Friend-Projekts und dieser Arbeit entstanden ist. Es ist in Java implementiert, und zusätzlich ist die Regelmaschine Jess (s. [9], [52]) integriert, welche die Verarbeitung der Regeln übernimmt. Jess ist eine der schnellsten verfügbaren Regelmaschinen. Da sie Java-basiert ist, lässt sie sich problemlos in eine Java-Umgebung einbetten und kann direkt Java-Objekte manipulieren und über sie schließen. In Abbildung 21 sind die Hauptkomponenten von SCENIOR abgebildet. Hier soll lediglich eine Systemübersicht gegeben werden, auf Details der Implementierung wird in Kapitel 6 eingegangen.

In der Initialisierungsphase des Systems lädt der Konverter (der ebenfalls eine Komponente von SCENIOR ist) die OWL Wissensbasis und übersetzt sie in *Jess-Regeln* und *Jess-Templates*. Das Resultat wird als *Konzeptuelle Jess Wissensbasis* (KJW) bezeichnet (s. Abschnitt 5.2.2). Wichtig ist hier die Unterteilung in *Teilmodelle*. Zwar könnten theoretisch alle alternativen Handlungsabläufe – z.B. eine Abfertigung *mit* Betankung und eine *ohne* – durch separate Modelle repräsentiert werden, doch das würde zu hoher Redundanz und ineffizienter Verarbeitung führen. Deshalb können sinnvolle Einheiten als optionale und mehrfach verwendbare Teilmodelle deklariert werden (s. Abschnitt 5.2.2.2). Einfach ausgedrückt sind diese Teilmodelle Repräsentationen für Teilhandlungen, die in gewisser Weise wichtig genug sind, als Ganzes erkannt zu werden. Die zeitlichen Constraints, die in den SWRL-Regeln definiert sind, werden bei der Konvertierung in zeitliche Constraint-Netze (ZCNs) übersetzt; dabei wird für jedes Teilmodell ein eigenes ZCN generiert (s. Abschnitt 5.2.3).

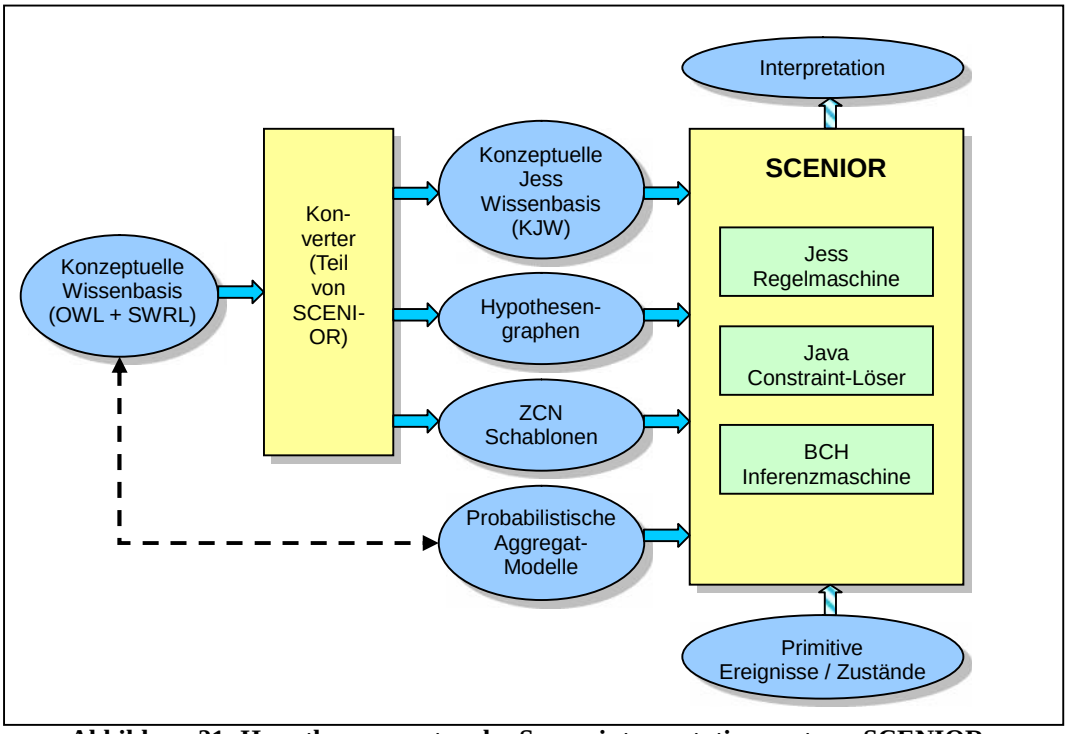


Abbildung 21: Hauptkomponenten des Szeneninterpretationssystems SCENIOR.

Für jedes Teilmodell wird ein separater *Jess-Thread* erzeugt und mit dem entsprechenden Hypothesengraph – der den zu erkennenden Handlungsablauf repräsentiert – und dem dazugehörigen ZCN ausgestattet. Ebenfalls wird für jeden Thread aus den probabilistischen Aggregatmodellen eine BCH generiert. Dann kann der Interpretationsprozess gestartet werden.

SCENIOR erwartet die Evidenzen einer sich entwickelnden Szene in Realzeit in Form von symbolischen Aktivitäts-Token, versehen mit quantitativen zeitlichen Informationen und eventuell weiteren nützlichen Eigenschaften. Bei der *Co-Friend-Plattform* (s. Abschnitt 7.1.2) werden diese Evidenzen Frame-basiert (ein Frame pro Sekunde) in Form von primitiven Ereignissen und Zuständen geliefert. Die punktuellen Ereignisse kommen nur in einem Frame vor, während die Zustände in allen Frames innerhalb des Zeitintervalls geliefert werden, in denen der Zustand gültig ist. Die Ereignisse und Zustände werden durch eine Tracking-Komponente der niederen Bildverarbeitungsschicht und einen Detektor für primitive Aktionen der Mittelschicht generiert. Diese Komponenten wurden von Partnern des Co-Friend-Projekts entwickelt (s. Abschnitt 7.1.2).

Der Interpretationsprozess startet mit den Threads, die initial für jedes Teilmodell erzeugt wurden. Die nun von der Mittelschicht gelieferten primitiven Ereignisse

und Zustände werden an alle Threads geliefert und in die Arbeitsspeicher der Regelmaschinen eingetragen. Dann werden in datengetriebener Weise die Regeln angewandt. Bei möglichen Mehrfachzuordnungen der Evidenzen zu den Hypothesengraphen entstehen durch Klonen neue Threads. Auf diese Weise werden parallel alternative Interpretationsmöglichkeiten verfolgt. Die zeitlichen Informationen (Zeitstempel) werden in die jeweiligen ZCNs eingespeist und vom Constraint-Löser propagiert (s. Abschnitt 5.2.3). Wenn es dabei zu einem Konflikt kommt, stirbt der entsprechende Thread. Außerdem werden die Zeitstempel an die BCHs geliefert. Sie errechnen ein Präferenzmaß für jeden Thread, so dass nur die vielversprechensten Interpretationen verfolgt werden, wenn die maximale Anzahl paralleler Threads erreicht ist. Auf diese Weise wird eine Strahlsuche (engl.: *Beam Search*) etabliert. Der Interpretationsprozess wird im Detail in Abschnitt 5.3 dargestellt.

5.2.2 Generieren von Regeln aus der Ontologie

Es wurde bereits in der Arbeit von Neumann und Möller [82] gezeigt, dass Szeneninterpretation als eine Suche im Raum der möglichen Interpretationen angesehen werden kann. Der Raum wird durch die taxonomischen und kompositionellen Relationen definiert, und die Suche in diesem Raum wird durch Constraints kontrolliert. Wie bereits in Abschnitt 2.3.1 erwähnt, wurden dabei vier Interpretationsschritte zur Navigation im Suchraum identifiziert, um eine Interpretation zu konstruieren:

- Aggregatinstanziierung (Aufwärtsschritt in der kompositionellen Hierarchie),
- Aggregatentfaltung (Abwärtsschritt in der kompositionellen Hierarchie),
- Instanzspezialisierung (Abwärtsschritt in der taxonomischen Hierarchie),
- Instanzverschmelzung (Verschmelzen von Instanzen, die separat erzeugt wurden).

In dem hier vorgestellten Rahmenwerk werden Regeln für die ersten drei Interpretationsschritte generiert, zusätzlich noch einige Unterstützungsregeln. Es wird sich zeigen, dass der Schritt der Instanzverschmelzung durch die Parallelverarbeitung und die Verwendung von Hypothesen entbehrlich wird.

5.2.2.1 Templates und Slots

Templates sind das zentrale Strukturelement der Regelsprache Jess. Sie ähneln Klassen einer objektorientierten Programmiersprache, mehr noch Relationen einer relationalen Datenbanken. Der Name eines Templates entspricht dem Namen der Relation einer Tabelle. Templates haben *Slots*, die den Attributen einer relationalen Datenbank entsprechen. Die Slots definieren also, welche Eigenschaften zu einem Objekt gespeichert werden können. Für jede Rolle eines Konzepts wird ein entsprechender Slot des korrespondierenden Templates erzeugt. Diese können wie objektorientierte Klassen hierarchisch definiert werden. Ein Template erbt auf diese Weise die Slots seines Eltern-Templates, wobei nur Einfachvererbung möglich ist. Mit dem Schlüsselwort `assert` können nun Fakten in den Arbeitsspeicher eingetragen werden. Für eine Instanz des Templates `Vehicle` mit dem Slot `name`, welches den Wert `vehicle_17` hat, stellt sich das folgendermaßen dar:

```
(assert (Vehicle (name vehicle_17)))                    (5-10)
```

Eine Instanz des zusammengesetzten Ereignisses `Aircraft-Arrival` (s. Formel 5-7) hat dementsprechend folgende Struktur:

```
(assert (Aircraft-Arrival                               (5-11)
    (name aa_14)
    (has-part-A aeaz_4)
    (has-part-B ap_7)))
```

Um bei der Übersetzung der OWL Wissensbasis die taxonomische Hierarchie zu erhalten, werden die Vererbungsmechanismen von Jess genutzt: zu jedem Konzept C der Wissensbasis wird ein Template C_j generiert, und für zwei Konzepte C, D mit $C \sqsubseteq D$ wird ein Template C_j und dessen Eltern-Template D_j erzeugt. Da in Jess nur Einfachvererbung möglich ist, darf auch in der OWL Wissensbasis nur Einfachvererbung verwendet werden. In OWL können auch Rollen hierarchisch definiert werden (s. Abschnitt 4.1.1.1). In Jess können Slots jedoch keine hierarchische Struktur haben. Dies wird simuliert, indem für ein Template nicht nur ein Slot der zugehörigen Rolle erzeugt wird, sonder auch Slots für alle generelleren Rollen entlang der Rollenhierarchie.

Es gibt bereits Ansätze zur Kombination von OWL, SWRL und Jess. Zu den bekanntesten gehören zum Einen der Ansatz von Erikson [48], implementiert als Protégé-Erweiterung *JessTab*, zum Anderen die *SWRLJessBridge* [13], die seit Version 3.4 fester Bestandteil von Protégé ist. Beide Ansätze sind nur interaktiv in Protégé zu verwenden und eignen sich nicht für eine vollautomatische inkrementelle Verarbeitung. Der von Erikson vorgestellte Ansatz beschreibt die Möglichkeit, Jess-Regeln auf Protégé-Instanzen anzuwenden. Dabei werden die Instanzen

der OWL Wissensbasis zwar automatisch in Jess-Fakten überführt, die Jess-Regeln müssen dagegen manuell erstellt werden. Der in [13] vorgestellte Ansatz übersetzt sowohl OWL Klassen und Instanzen in Templates und Fakten von Jess, als auch SWRL-Regeln in Jess-Regeln. Er ist daher verwandt mit dem hier vorgestellten Ansatz, unterscheidet sich jedoch in wesentlichen Aspekten. Bei dem SWRLJessBridge-Mechanismus werden alle Klassen in Templates übersetzt, die nur den Slot `name` besitzen; in ihm wird der Name der Instanz gespeichert. Die taxonomische Struktur der OWL Wissensbasis wird nicht durch hierarchische Jess-Templates abgebildet (die Struktur der Templates ist flach), sondern durch das Duplizieren von Fakten entlang der taxonomische Hierarchie. Weiterhin werden die Eigenschaften von Konzepten nicht in Template-spezifischen Slots gespeichert (als sogenannte *ungeordnete Fakten* – ungeordnet deshalb, da auf Slot-Inhalte gezielt durch die Bezeichnung des Slots zugegriffen werden kann und daher keine Reihenfolge beachtet werden muss), sondern als sogenannte *geordnete Fakten*, welche im Prinzip einfachen Listen entsprechen, für die nicht explizit ein Template definiert werden muss. Das Beispiel aus Programmausdruck 5-11 stellt sich im SWRLJessBridge-Mechanismus folgendermaßen dar (`aa` $\triangleq$ `aircraft_arrival`, `aeaz` $\triangleq$ `aircraft_enters_arrival_zone`, `ap` $\triangleq$ `aircraft_positioned`):

```
(assert (Aircraft-Arrival (name aa_14)))                    (5-12)
(assert (has-part-A aa_14 aeaz_4))
(assert (has-part-B aa_14 ap_7))
```

Nun gilt für `Aircraft-Arrival` folgende Klassenbeziehung:

```
Aircraft-Arrival⊑Composite-Event⊑Event                      (5-13)
                ⊑Conceptual-Object⊑Scene-Object
```

Diese Klassenbeziehung wird durch das Hinzufügen weiterer Fakten mit demselben Namen emuliert:

```
(assert (Composite-Event (name aa_14)))                     (5-14)
(assert (Event (name aa_14)))
(assert (Conceptual-Object (name aa_14)))
(assert (Scene-Object (name aa_14)))
```

Dadurch wird eine korrekte Verarbeitung von Regeln gewährleistet. Eine mögliche SRWL-Regel zur Instanziierung des Aggregats `Aircraft-Arrival` stellt sich folgendermaßen dar:

```
Conceptual-Object(?co) ^
has-part-A(?co, ?aeaz) ^
has-part-B(?co, ?ap)
->
Aircraft-Arrival(?co)
```
(5-15)

Die daraus automatisch generierte Jess-Regel des SWRLJessBridge-Mechanismus lautet:

```
(defrule aa-instantiation-rule
    (Conceptual-Object (name ?co))
    (has-part-A ?co ?aeaz)
    (has-part-B ?co ?ap)
     =>
    (assert (Aircraft-Arrival (name ?co)))
)
```
(5-16)

Mit SWRL-Regeln können keine neuen Instanzen erzeugt werden, deshalb muss sich eine Instanz von `Conceptual-Object` in der OWL Wissensbasis befinden, die dann zusätzlich zu einer Instanz von `Aircraft-Arrival` wird, wenn die entsprechenden Bedingungen erfüllt sind.

Wenngleich die Übersetzung der SWRL-Regeln in Jess-Regeln bei diesem Ansatz relativ unkompliziert und gradlinig ist, so hat sie den Nachteil, dass die objektzentrierte Darstellung der Aggregate in Jess verloren geht, da die Eigenschaften der Konzepte als geordnete Fakten gespeichert werden. Weiterhin entstehen durch das Duplizieren viele zusätzliche Fakten; beides kann zu Skalierungsproblemen bei komplexen Anwendungen führen. Deshalb wird in dieser Arbeit ein anderer Ansatz verfolgt, der in den nächsten Abschnitten vorgestellt wird.

5.2.2.2 Teilmodelle

Wie bereits in Abschnitt 5.1.3 erwähnt, können komplexe statische und dynamische Szenen realer Domänen oftmals nicht nur durch *ein* Modell beschrieben werden. Vielmehr ist es der Normalfall, dass verschiedene alternative Modelle nötig sind, um Variationen in den Szenen zu beschreiben. In der Domäne der Flughafenvorfeldaktivitäten wurden schon unterschiedliche Be- und Entladeszenarien genannt. Weitere Beispiele sind das Betanken oder Enteisen des Flugzeugs, welche nicht Bestandteil jeder Abfertigung sind. Die Modelle sind also nicht vollkommen verschieden, sondern unterscheiden sich nur in einigen Teilen. Es ist leicht vorstellbar, dass derartige Situationen auch in anderen Domänen auftreten.

Eine Möglichkeit wäre es nun, dass für jede Alternative ein vollständig separates Modell erstellt wird. Dieser Ansatz hat den offensichtlichen Nachteil hoher Redundanz, da nahezu identische alternative Interpretationen parallel verfolgt werden müssen. Das Problem der hohen Redundanz würde sich jedoch auch bei nur einem Modell ergeben, bedingt durch den Interpretationsprozess, bei dem durch Klonen der Threads eine Vielzahl von möglichen Interpretationen untersucht werden. Dies soll an einem einfachen Beispiel erläutert werden. Angenommen, das Modell in Abbildung 22 sei zusammenhängend, mit Wurzel `A` und verschiedenen primitiven Ereignissen bzw. Zuständen `{E,F,G,H,I,J,K,L}`. Zur Vereinfachung werden hier Constraints außer Acht gelassen, und es wird weiter angenommen, dass es jeweils eine passende primitive Evidenz `{e,f,g,h,i,j,k,l}` gibt. Werden alle möglichen Zuordnungen betrachtet, dann ergibt sich die Anzahl der Threads als Potenzmenge der Menge der Evidenzen, d.h. $2^8 = 256$ Threads. Wird das Modell jedoch in drei Teilmodelle M_1, M_2 und M_3 zerlegt, dann werden für die Modelle M_2 und M_3 jeweils $2^4 = 16$ Threads erzeugt, für beide zusammen also 32. Nur jeweils ein Thread für `B` und einer für `C` führen zu einer vollständigen Instanziierung. Für diese Threads werden nun sogenannte *höhere Evidenzen* `b` und `c` erzeugt und an den Thread mit Modell M_1 weitergereicht. Dort entstehen weitere $2^2 = 4$ Threads, insgesamt also nur 36 Threads gegenüber 256 bei *einem* Modell (in der Praxis wird die Differenz jedoch deutlich niedriger ausfallen, da durch die Constraints und Typunterschiede eine Vielzahl der Alternativen unterbunden wird).

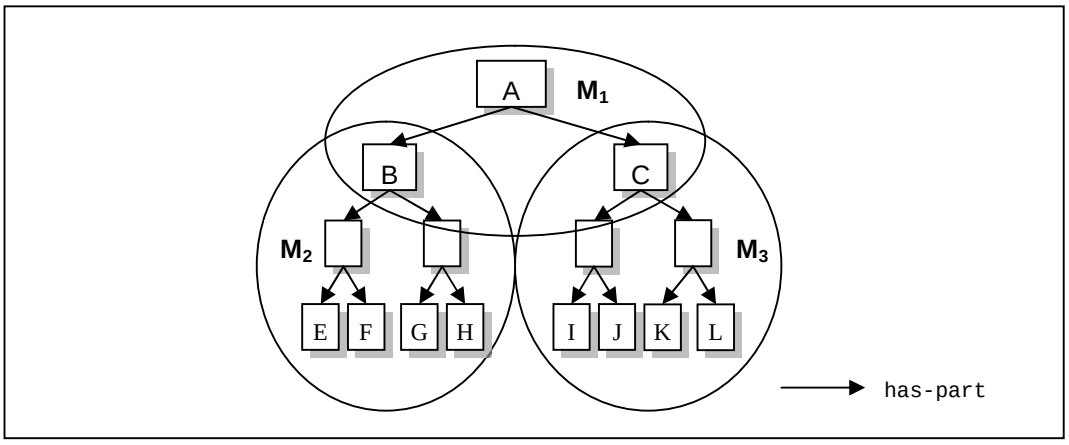


Abbildung 22: Reduzierung der Redundanz durch Zerlegung in Teilmodelle.

Daher wird hier der Ansatz verfolgt, einzelne oder alternative Modelle in verschiedene Teilmodelle zu zerlegen. Diese Vorgehensweise hat den weiteren Vorteil, dass Teilziele definiert werden können, die als eigene Interpretationsaufgabe angesehen werden können.

Teilmodelle repräsentieren demzufolge Teilziele, deren Instanzen als Zwischenergebnisse in Form von höheren Evidenzen als Eingabe für andere Aggregate dienen. Diese höheren Evidenzen werden im Interpretationsprozess prinzipiell wie primitive Evidenzen behandelt. In der OWL Wissensbasis werden Konzepte, welche die Wurzel dieser Teilmodelle bilden, mit der Eigenschaft `context-free` markiert. Für den Transformationsprozess ist dies das Zeichen, ein eigenständiges Teilmodell in einem separaten Thread zu erstellen. Alternative Teilmodelle können als Aggregate definiert werden, deren Wurzelkonzepte Unterklassen eines Konzepts sind, welches als `context-free` markiert ist. Wird das Wurzelkonzept R eines Teilmodells M_k instanziiert, dann wird diese Instanz als höhere Evidenz an alle Threads weitergereicht, deren Teilmodelle R oder eine Oberklasse davon als Blatt haben.

In Abbildung 23 ist dies an einem Beispiel illustriert. Die Konzepte `B` und `C` sind als `context-free` markiert. Weiterhin gilt $C_1 \sqsubseteq C$ und $C_2 \sqsubseteq C$. C_1 könnte beispielsweise einen `Service (G)` mit `Refuelling (F)` und C_2 einen `Service` ohne `Refuelling` repräsentieren. Wird nun C_1 oder C_2 erkannt (vollständig instanziiert), wird die jeweilige Instanz als höhere Evidenz an die Threads des Modells M_1 weitergereicht, welches einen `Turnaround` repräsentieren könnte. Dort können sowohl C_1 als auch C_2 an `C` zugewiesen werden.

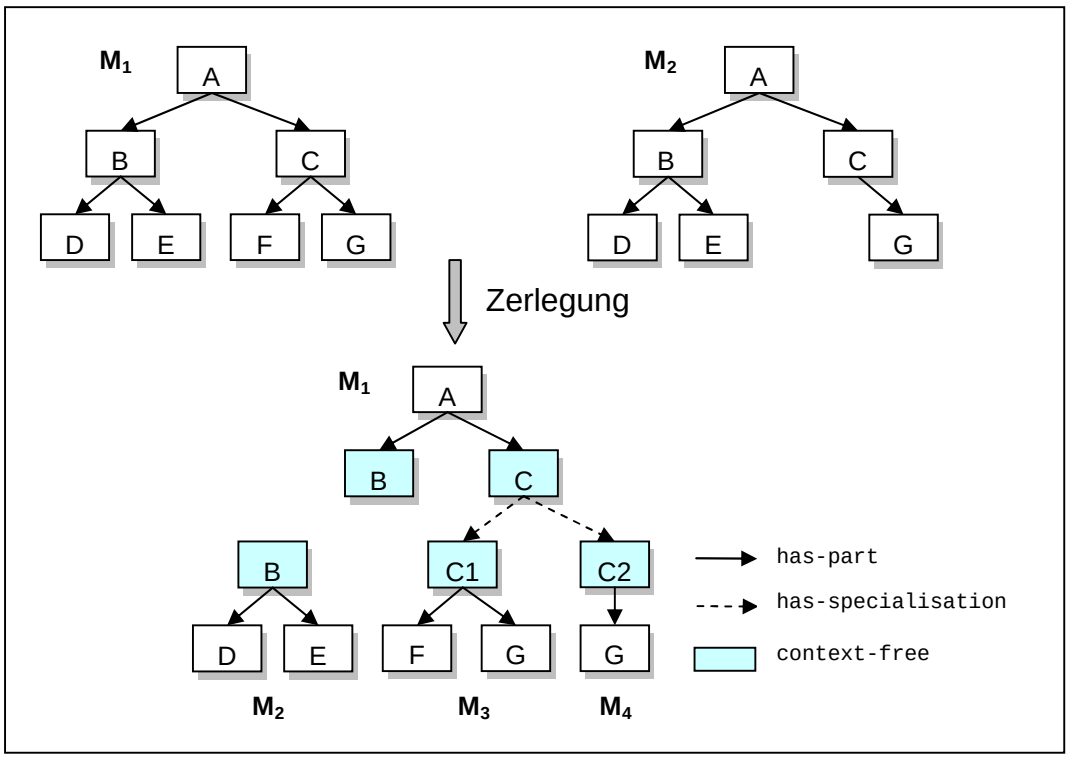


Abbildung 23: Zerlegung von alternativen Modellen in Teilmodelle.

5.2.2.3 Hypothesengraphen

Aggregat-Instanziierungsregeln beschreiben die Aufwärtsschritte in der kompositionellen Hierarchie und stellen sozusagen das Rückgrat der Szeneninterpretation dar. Der erste und naheliegende Ansatz für die Struktur einer automatisch generierten Aggregat-Instanziierungsregel stellt sich für das Beispiel `Refuelling` (s. auch Abbildung 16) folgendermaßen dar (aus Gründen der Übersichtlichkeit ist das Überprüfen der Constraints hier nur angedeutet):

```
(defrule refuelling-instantiation-rule                    (5-17)
    (Tanker-Positioning (name ?tp))
    (Tanker-Positioned (name ?tpd)
    (Tanker-Leaves-Tanking-Zone (name ?tlz))
    ;; überprüfe Constraints
    =>
    (assert (Refuelling
                (name new_ref)
                (has-part-A ?tp)
                (has-part-B ?tpd)
                (has-part-C ?tlz)))
)
```

Wenn also die Evidenzen für die Teile des Aggregats vorliegen und die Constraints erfüllt sind, wird das Aggregat instanziiert und als neuer Fakt dem Arbeitsspeicher hinzugefügt. Die Teile werden dabei der neuen `Refuelling`-Instanz zugeordnet. Wenngleich dieser Ansatz einleuchtend und gradlinig erscheint, bringt er doch verschiedene Probleme mit sich, die im Folgenden erläutert werden.

Problem 1: Parallele Verfolgung und Bewertung möglicher Interpretationen. Zunächst könnte hier die provokative Frage gestellt werden, warum überhaupt der nicht unerhebliche Aufwand der Verwaltung paralleler Threads betrieben wird und nicht einfach nur *eine* Regelmaschine benutzt wird. Als wesentliche Aspekte sind hier die Anforderung der inkrementellen Verarbeitung und die Echtzeitfähigkeit zu nennen. Das Interpretationssystem muss in der Lage sein, zu jedem beliebigen Zeitpunkt Zwischenergebnisse der Interpretation abliefern zu können. Weiterhin ist in bestimmten Domänen, z.B. bei den Flughafenvorfeldaktivitäten, eine Vorhersage des weiteren Verlaufs der Szene von Interesse, beispielsweise hinsichtlich der zeitlichen Entwicklung. Dies setzt voraus, alternative Interpretationsmöglichkeiten separat zu verfolgen und mit einem Präferenzmaß zu bewerten. Mit nur einer Regelmaschine ist dies kaum zu realisieren, da sich alle primitiven Evidenzen und alle daraus abgeleiteten höheren Evidenzen in einem Arbeitsspeicher befinden würden, und die alternativen Interpretationsmöglichkeiten nur schwer identifiziert werden könnten.

Ziel ist daher die parallele Verfolgung möglicher Interpretationen in klar voneinander getrennten Threads. In der Regel können jedoch aufgrund der Kombinatorik nicht alle logisch konsistenten Möglichkeiten verfolgt werden, so dass unwahrscheinliche Interpretationen frühzeitig verworfen werden müssen. Zu diesem Zweck wird ebenfalls das Präferenzmaß herangezogen, welches jede mögliche Alternative bewertet und diejenigen mit einer niedrigen Bewertung aussortiert.

Regeln im Allgemeinen und von der Art, wie sie in Regel 5-17 dargestellt ist, haben einen lokalen Charakter, d.h., sie können nur die aggregatspezifischen Constraints in Betracht ziehen, nicht aber Constraints, welche höher in der kompositionellen Hierarchie definiert sind. Das abstrakte Beispiel in Abbildung 24 verdeutlicht diesen Zusammenhang. Das Identitäts-Constraint des Aggregats `A` legt fest, dass die Evidenz für `E` des Aggregats `B` mit der Evidenz für `E` des Aggregats `C` identisch sein muss. Regeln nach Schema 5-17 würden jedoch `B` und `C` auch dann instanziieren (vorausgesetzt, alle nötigen Evidenzen wären vorhanden und alle übrigen Constraints erfüllt), wenn die Evidenzen e_1 und e_2 verschieden wären. Erst die Instanziierungsregel des Aggregats `A` würde – zu einem möglicherweise viel späteren Zeitpunkt – feststellen, dass das Constraint verletzt ist. Wünschenswert wäre hier ein Mechanismus, der diese Alternative von vornherein ausschließt.

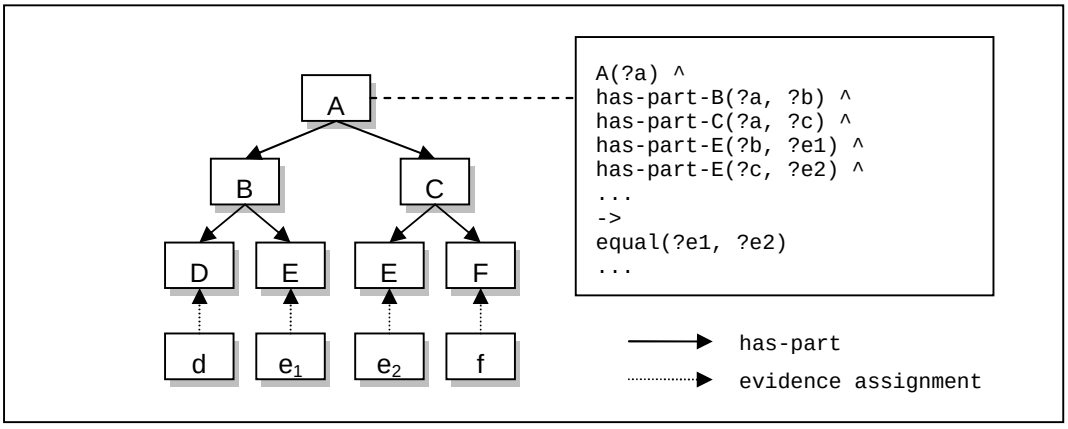


Abbildung 24: Aggregat-Instanziierungsregeln für `B` und `C` kennen das Identitäts-Constraint von `A` nicht.

Problem 2: Aggregat-Entfaltungsregel. Eine Aggregat-Entfaltungsregel beschreibt den Abwärtsschritt in der kompositionellen Hierarchie. Sie erzeugt die Teile eines Aggregats, wenn das Aggregat selbst bereits instanziiert wurde und dies nicht durch eine Aggregat-Instanziierungsregeln geschehen ist, sondern beispielsweise durch Inferenztechniken, welche auf Allgemeinwissen (engl.: *common sense*) basieren. Eine Möglichkeit dies zu realisieren wäre, zunächst alle Teile zu erzeugen

(möglicherweise unter Beachtung von Constraints). Die entsprechende Regel lautet:

```
(defrule refuelling-extraction-rule
    ?ref-id <- (Refuelling                                          (5-18)
                    (name (?ref)
                    (has-part-A nil)
                    (has-part-B nil)
                    (has-part-C nil))
    =>
    (assert (Tanker-Positioning (name new_tp)))
    (assert (Tanker-Positioned (name new_tpd)))
    (assert (Tanker-Leaves-Tanking-Zone (name new_tlz)))
    (modify ?ref-id (has-part-A new_tp)
                    (has-part-B new_tpd)
                    (has-part-C new_tlz)))
```

In einem anschließenden Schritt müssten dann evtl. Instanzverschmelzungen stattfinden, da einige Evidenzen der Teile vorliegen könnten. Bei einer hohen Anzahl von Evidenzen könnte dies durch die Kombinatorik zu einer Vielzahl von Überprüfungen und damit zu Performanzproblemen führen. Eine andere Möglichkeit wäre, das Entfalten des Aggregats und die Instanzverschmelzung miteinander zu kombinieren. Eine Regel, welche feuert, wenn eine passende Evidenz für `Tanker-Positioning` vorhanden ist, lautet entsprechend:

```
(defrule refuelling-extraction-rule                                 (5-19)
    ?ref-id <- (Refuelling
                    (name (?ref)
                    (has-part-A nil)
                    (has-part-B nil)
                    (has-part-C nil))
    (Tanker-Positioning (name ?tp))
    =>
    (assert (Tanker-Positioned (name new_tpd)))
    (assert (Tanker-Leaves-Tanking-Zone (name new_tlz)))
    (modify ?ref-id (has-part-A ?tp)
                    (has-part-B new_tpd)
                    (has-part-C new_tlz)))
```

Jedoch müssten derartige Regeln für jede Kombinationen der Teile (ohne den Fall, dass alle Teile vorhanden sind) erzeugt werden, d.h. $2^{\#Teile} - 1$ Regeln. Weitere Mechanismen wären nötig, um sicherzustellen, dass diejenige Regel zuerst feuern würde, welche die meisten Evidenzen in Betracht zieht. Beides ist problematisch.

Problem 3: Halluzinieren. Bereits in [82] wurde beschrieben, dass Szeneninterpretation über die Verarbeitung visueller Evidenz hinausreicht, beispielsweise bei der Vorhersage des weiteren Verlaufs einer dynamischen Szene oder der Halluzination von Objekten, für die keine Evidenz vorhanden ist. Gründe dafür können Unzu-

länglichkeiten in der niederen Bildverarbeitung oder auch Verdeckung sein. Für komplexe Domänen der realen Welt sind Instanziierungen mit unvollständiger visueller Evidenz sicherlich eher die Regel als die Ausnahme. Die Erfahrungen mit der Flughafendomäne bestätigen dies. Max Clowes (1971) drückte diesen Sachverhalt in seinem bekannten Satz „*Vision is controlled hallucination*“ aus.

Auch in dieser Arbeit wird der Ansatz verfolgt, fehlende Evidenz zu halluzinieren, um vielversprechende Threads weiterzuführen. Doch ein entsprechender Mechanismus ist durch Regeln im Allgemeinen schwer zu realisieren, denn in einer dynamischen Szene könnte alleine das Voranschreiten der Zeit das Halluzinieren einer Evidenz erzwingen. Wenn beispielsweise das Aggregat `Refuelling` so modelliert ist, dass ein Tankvorgang höchstens 60 Minuten dauert, dann könnte nach Ablauf dieser Zeit – gemessen vom Auftreten der Evidenz, dass der Tanker in die Tankzone gefahren ist – das Herausfahren halluziniert werden, wenn die entsprechende Evidenz fehlt. Regeln reagieren jedoch auf (sich ändernde) Fakten, nicht auf deren Abwesenheit, die möglicherweise zudem an globale zeitliche Constraints gebunden ist.

Der in dieser Arbeit entwickelte Mechanismus, um die oben genannten Probleme zu lösen, ist das Erstellen von *Hypothesen*.

Hypothesen. Hypothesen können als Erwartungen der zu erkennenden Handlungsabläufe oder statischen Strukturen betrachtet werden, dargestellt durch *Hypothesengraphen*. Sie repräsentieren die partonomische Struktur von Teilmodellen, unter Berücksichtigung der Identitäts-Constraints, die durch die SWRL-Regeln beschrieben werden. Dabei müssen nicht nur alle SWRL-Regeln der Aggregate des Teilmodells berücksichtigt werden, sondern auch die SWRL-Regeln zugehöriger Oberkonzepte. Nur auf diese Weise ist die in Abschnitt 5.1 beschriebene Vererbung von Constraints gewährleistet.

Für jedes Teilmodell der OWL Wissensbasis wird im Transformationsprozess ein Hypothesengraph generiert. Jeder dieser Graphen repräsentiert demzufolge ein unabhängiges Interpretationsteilziel. Die Hypothesengraphen ermöglichen eine einfache Evidenzzuweisung und Aufspaltung des Suchbaums in alternative Interpretationen, die parallel verfolgt werden. Weiterhin können bereits bei der Zuweisung der Evidenz Identitäts-Constraints in Betracht gezogen werden, welche höher in der kompositionellen Hierarchie definiert sind. Darüber hinaus können Hypothesengraphen verwendet werden, um fehlende Evidenzen zu halluzinieren. Auf diese Mechanismen wird in den folgenden Kapiteln detailliert eingegangen. Abbildung 25 zeigt den Hypothesengraph für das Teilmodell `Arrival`. Jedes Element eines Hypothesengraphen wird dem Arbeitspeicher der Regelmaschine als Fakt hinzugefügt. Werden gleiche Konzepte mehrmals in einem Teilmodell verwendet, werden sie durch individuelle Namen voneinander unterschieden.

Die Verwendung von Hypothesengraphen ist jedoch nicht nur aus technischer, sondern auch aus kognitionswissenschaftlicher Sicht motiviert. Hawkins [62] vertritt die These, dass die Vorhersage eine der primären Funktionen des Neocortex und die Grundlage der Intelligenz ist. Er beschreibt in dem sogenannten *Gedächtnis-Vorhersage-Modell*, dass das Gehirn Wissen über Ereignisfolgen in Form von invarianten Repräsentationen speichert, die auch unter veränderten Umständen gelten. Durch komplizierte Informationsflüsse zwischen den verschiedenen Schichten des Neocortex versucht das Gehirn, eine konsistente Interpretation der Welt zu erstellen. Vereinfacht dargestellt, wandert sensorischer Input die Hierarchie hinauf und führt zu Aktivierungen von Ablaufmodellen, die zu Vorhersagen führen, welche die Hierarchie hinab wandern. Je höher sich eine Schicht in der Hierarchie befindet, desto höher ist der Grad der Invarianz bzw. Abstraktion der darin enthaltenen Informationen. Musterfolgen können die Hierarchie hinaufwandern und dort zu einer stabilen Wahrnehmung führen. Wenn eine Person beispielsweise um einen Kühlschrank herum läuft, ändert sich der sensorische Input und damit die Aktivierung der Neuronen in den unteren Schichten des Neocortex ständig, während viel weniger Neuronen in höheren Schichten das Objekt „Kühlschrank“ repräsentieren und eine konsistente Wahrnehmung des Objekts ermöglichen. Wenn ein Muster die Hierarchie hinunterwandert, wird es zu Musterfolgen *aufgefaltet*, z.B. wenn man sich an Details eines Objekts oder Ereignisses erinnert oder Vorhersagen getroffen werden.

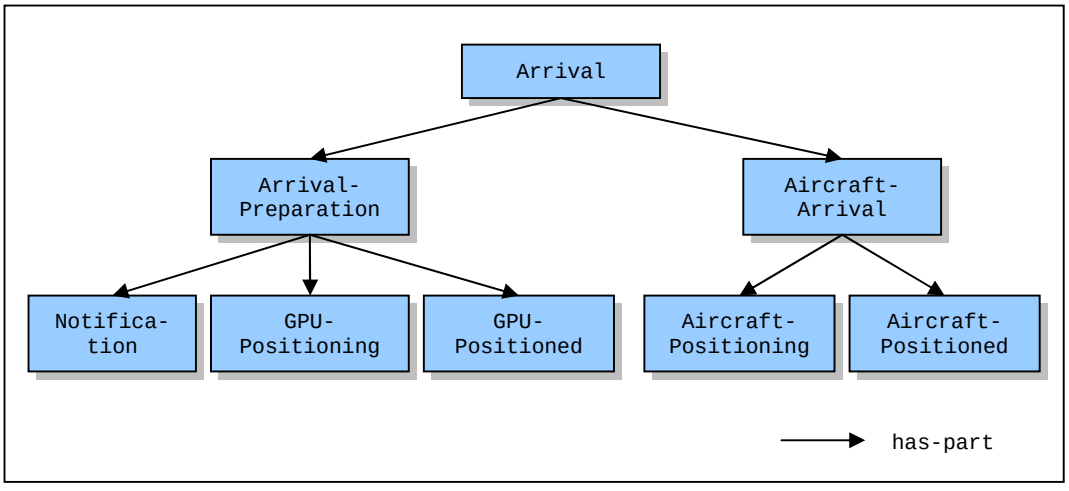


Abbildung 25: Hypothesengraph für das Teilmodell `Arrival`.

Der Mechanismus der Hypothesengraphen greift diese Idee der Vorhersage in sehr vereinfachter Form auf. Wird eine Evidenz zugeordnet, werden seine Zeitstempel in das zugehörige zeitliche Constraint-Netz (s. Abschnitt 5.2.3) eingespeist und die Werte durch das Netz propagiert. Dadurch öffnen sich für die fehlenden Evidenzen Zeitfenster, in denen sie zugeordnet werden können. Sind die Zeitfenster

überschritten, ist dies der Auslöser für eine mögliche Halluzinierung der fehlenden Evidenz.

Abbildung 26 verdeutlicht, wie der Hypothesengraph für das abstrakte Beispiel aus Abbildung 24 die Identitäts-Constraints reflektiert. Für eine Evidenz `e` gibt es nur eine Zuordnungsmöglichkeit, wodurch `e` als Teil einer Instanziierung von `B` und `C` festgelegt wird und damit das Identitäts-Constraint des Aggregats `A` erfüllt ist.

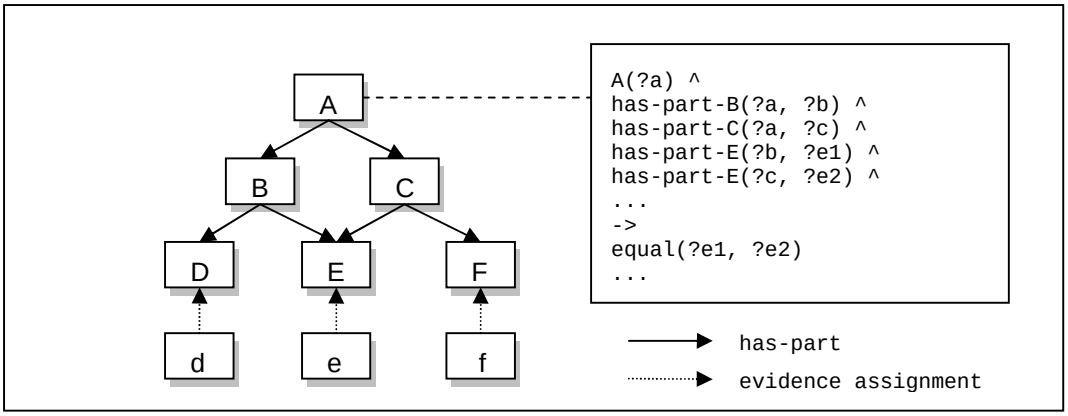


Abbildung 26: Hypothesengraphen reflektieren Identitäts-Constraints.

5.2.2.4 Regeln

Im nächsten Schritt des Transformationsprozesses werden die folgenden Regeln vollautomatisch aus der OWL Wissensbasis generiert:

- Evidenz-Zuordnungsregeln,
- Aggregat-Instanziierungsregeln,
- Aggregat-Entfaltungsregeln,
- Spezialisierungsregeln,
- Zeitaktualisierungsregeln.

Evidenz-Zuordnungsregeln. Eine Evidenz-Zuordnungsregel ordnet eine Evidenz einem Hypothesenelement zu, welches ein Blatt des entsprechenden Hypothesengraphs repräsentiert. Bei der Evidenz kann es sich dabei um eine durch die niedere

Bildverarbeitung gelieferte primitive Evidenz oder um eine durch einen anderen Thread gelieferte, höhere Evidenz handeln. Das generische Schema für eine Evidenz-Zuordnungsregel stellt sich folgendermaßen dar (das Überprüfen der zeitlichen Constraints und das Aktualisieren des Constraint-Netzes ist hier aus Gründen der Übersichtlichkeit nur angedeutet):

```
(01) (defrule aggregate-x-ea-rule                                    (5-20)
(02)     ?e-id <- (template-x (name ?e)
(03)                  (status evidence)
(04)                  (dependes-on-instances $?e-ins))
(05)     ?h-id <- (template-x (name ?h) (status ?status_1))
(06)     (test (or (eq ?status_1 hypothesised)
(07)               (eq ?status_1 hallucinated)))
(08)     ?m-id <- (matched-instances (has-parts $?m-ins))
(09)
(10)     (test (eq (intersection$ $?e-ins $?m-ins)
(11)                              (create$)))
(12)     ;; check temporal constraints
(13)     =>
(14)     (modify ?e-id (status assigned))
(15)     (modify ?h-id (status instantiated)
(16)                   (depends-on-instances $?e-ins))
(17)     (modify ?m-id (has-parts union$ $?m-ins $?e-ins))
(18)     ;; update temporal constraint net
(19) )
```

In der Prämisse der Regel wird das Element einer Evidenz des Aggregats bzw. primitiven Konzepts `x (template-x)` und ein passendes Hypothesenelement adressiert. Außerdem werden die Zeitstempel der Evidenz mit Hilfe einer Testfunktion in das korrespondierende Constraint-Netz eingespeist und propagiert (hier durch Zeile 12 nur angedeutet). Kommt es dabei zu einem Konflikt, wird `false` zurückgegeben, andernfalls `true`. Die Prozessierung der zeitlichen Constraints wird im Detail in Abschnitt 5.2.3 behandelt. Im Aktionsteil der Regel wird der Status der Evidenz auf `assigned` gesetzt und das Hypothesenelement auf `instantiated`. Außerdem wird das zeitliche Constraint-Netz aktualisiert.

Während die Identitäts-Constraints bei der Zuweisung primitiver Evidenz durch den Hypothesengraph berücksichtigt werden (für ein mehrfach genutztes Konzept gibt es nur eine mögliche Zuordnung, siehe Abbildung 26), ist es notwendig, dies bei der Zuweisung höherer Evidenz explizit zu prüfen. In der ersten Realisierung des Systems galt hier die Einschränkung, dass Identitäts-Constraints nur innerhalb eines Teilmodells definiert werden dürfen, nicht zwischen verschiedenen Teilmodellen. Für alle Instanzen, die nicht als identisch definiert werden, wird implizit angenommen, dass sie verschieden sein müssen. Bei der Zuweisung einer höheren Evidenz e_h zu einem Teilmodell M_1 muss dann lediglich geprüft werden, ob die

Menge der Instanzen, die zu e_h gehören, disjunkt ist zur Menge aller Instanzen bisher zugeordneter (höherer) Evidenzen (gespeichert in dem Fakt `matched-instances`, der in jedem Thread vorhanden ist, Zeile 8). Die Disjunkheit wird in Zeile 10 und 11 geprüft. Im Aktionsteil wird `matched-instances` aktualisiert (Zeile 17), und die primitiven Evidenzen werden im Hypothesenelement gespeichert (Zeile 16), damit sie in den Aggregat-Instanziierungsregeln weitergegeben werden können.

Für die Flughafenvorfeld-Domäne stellt diese Einschränkung kein Problem dar, da die Teilmodelle in sich abgeschlossene Aktivitäten repräsentieren, mit jeweils eigenen Agenten (hier: Fahrzeugen). Für andere Domänen, die in Kapitel 7 vorgestellt werden, stellte sich diese Einschränkung jedoch als unvorteilhaft dar, so dass die Aggregat-Instanziierungsregeln entsprechend erweitert wurden, um auch Identitäts-Constraints zwischen Teilmodellen zu erlauben.

Evidenz-Zuordnungsregeln werden für folgende Aggregate bzw. primitive Konzepte generiert:

- für alle primitiven Konzepte `C` der Domänen-Ontologie Σ_{Domain}, mit: `(C⊑Primitive-State)⊔(C⊑Primitive-Event)`,
  welche direkte Unterklassen der Top-Level-Ontologie Σ_{Upper} sind (Zuweisung primitiver Evidenz).
- für alle Aggregate `C` der Domänen-Ontologie Σ_{Domain}, die mit der Eigenschaft `context-free` markiert sind und die direkte Unterklassen der Top-Level-Ontologie Σ_{Upper} sind (Zuweisung höherer Evidenz).

Die Evidenz-Zuordnungsregeln werden sowohl für die primitive, als auch für die höhere Evidenz nur für die jeweils generellsten Konzepte bzw. Aggregate der Domänen-Ontologie generiert. Für speziellere Konzepte und Aggregate sichert der Vererbungsmechanismus von Jess und die hierarchische Template-Struktur, dass die entsprechenden Evidenzen ebenfalls mit diesen Regeln zugewiesen werden.

Aggregat-Instanziierungsregeln. Eine Aggregat-Instanziierungsregel instanziiert ein Aggregat mit dem Status `hypothesised`, wenn alle Teile instanziiert oder halluziniert sind, d.h. den Status `instantiated` oder `hallucinated` haben. Dies ist ein Aufwärtsschritt in der kompositionellen Hierarchie. In Regel 5-21 ist zunächst das generische Schema für die erste Variante der Aggregat-Instanziierungsregel dargestellt, welche keine Identitäts-Constraints zwischen Teilmodellen erlaubt. Durch den Mechanismus der Hypothesengraphen kann die Aggregat-Instanziierungsregel relativ einfach ausgedrückt werden. Zeitliche

Constraints müssen nicht mehr überprüft werden, da dies bereits durch die Evidenz-Zuordnungsregeln geschehen ist. Im Aktionsteil der Regel wird lediglich der Status des Hypothesenelements auf `instantiated` gesetzt (Zeile 21) und das Set der zugehörigen primitiven Evidenzen aktualisiert (Zeile 22, 23).

```
(01) (defrule aggregate-x-ai-simple-rule                          (5-21)
(02)     ?h-id <- (template-x (name ?h)
(03)                  (status hypothesised)
(04)                  (has-part-1 ?p_1)
(05)                   ...
(06)                  (has-part-n ?p_n))
(07)     (template-part-1 (name ?p_1)
(08)                      (status ?status_1)
(09)                      (depends-on-instances $?p_1_ins))
(10)
(11)     (test (or (eq ?status_1 instantiated)
(12)               (eq ?status_1 hallucinated)))
(13)      ...
(14)     (template-part-n (name ?p_n)
(15)                      (status ?status_n)
(16)                      (depends-on-instances $?p_n_ins))
(17)
(18)     (test (or (eq ?status_n instantiated)
(19)               (eq ?status_n hallucinated)))
(20)     =>
(21)     (modify ?h-id (status instantiated)
(22)                   (depends-on-instances (union$
(23)                                $?p_1_ins...$?p_n_ins))))
```

Die erweiterte Aggregat-Instanziierungsregel, welche Identitäts-Constraints zwischen Teilmodellen ermöglicht, ist komplexer. Das generische Schema für ein Beispiel, bei dem die Agenten zweier Teile des Aggregats als identisch vorausgesetzt werden, ist in Regel 5-23 dargestellt. Es ist zu bemerken, dass die Identitäts-Constraints nur zwischen direkten Kindern bzw. physikalischen Objekten der Teile formuliert werden dürfen. Denn bei der Weitergabe einer höheren Evidenz werden ihre Teile nicht mitgeführt, mit anderen Worten, in der Jess-Regel kann nicht über mehrere Ebenen auf Teile einer höheren Evidenz zugegriffen werden, denn die entsprechenden Fakten liegen in der Regelmaschine nicht vor. Sollen Identitäts-Constraints zwischen Objekten tieferer Ebenen verschiedener Teilmodellen formuliert werden, so müssen diese bis zur Wurzel des jeweiligen Aggregats hoch gereicht werden; in der SWRL-Regel 5-22 ist beispielhaft dargestellt, wie der Agent von Teil `A` des Aggregats `X` an die Wurzel weitergereicht wird.

```
Aggregat(?x) ^
has-agent(?xa) ^
has-part-A(?x, ?a) ^
has-agent(?a, ?aa)
->
equals(?xa, ?aa)
```
(5-22)

Bei der Übersetzung der OWL Wissensbasis in die Jess Wissensbasis wird analysiert, welche Objekte durch `equal` als identisch definiert sind. Angenommen, dies sei für die Agenten von `part-1` und `part-2` des Aggregats `X` der Fall. Sie müssen in die Prämisse der Regel aufgenommen werden (Zeile 9 und 17).

```
(01) (defrule aggregate-x-ai-rule
(02)     ?h-id <- (template-x (name ?h)
(03)                  (status hypothesised)
(04)                  (has-part-1 ?p_1)
(05)                   ...
(06)                  (has-part-n ?p_n))
(07)     (template-part-1 (name ?p_1)
(08)                      (status ?status_1)
(09)                      (has-agent ?a_1)
(10)                      (depends-on-instances $?p_1_ins))
(11)
(12)     (test (or (eq ?status_1 instantiated)
(13)               (eq ?status_1 hallucinated)))
(14)
(15)     (template-part-2 (name ?p_2)
(16)                      (status ?status_2)
(17)                      (has-agent ?a_2)
(18)                      (depends-on-instances $?p_2_ins))
(19)
(20)     (test (or (eq ?status_2 instantiated)
(21)               (eq ?status_2 hallucinated)))
(22)      ...
(23)     (HelpFunctions (OBJECT ?hf-obj))
(24)      ...
(25)     =>
(26)     (bind $?ins-set (create$ $?p_1_ins $?p_2_ins...))
(27)     (if (and
(28)             (call ?hf-obj isEqual ?a_1 ?a_2)
(29)             (bind $?ins-set complement (create$ ?a_1)
(30)                                         $?ins-set))
(31)             (call ?hf-obj hasNoEqualElements $?ins-set))
(32)      then
(33)         (modify ?h-id (status instantiated)
(34)                       (depends-on-instances (union$
(35)                             $?p_1_ins $?p_2_ins...)))))
```
(5-23)

Die Überprüfung auf Identität der entsprechenden Instanzen muss im Aktionsteil der Regel stattfinden, denn die Teile sind bereits instanziiert. Würde die Überprüfung in der Prämisse erfolgen und `false` ergeben, dann könnte die Regel nie feuern; der entsprechende Thread würde weiterhin aktiv bleiben und kostbare Ressourcen blockieren. In Zeile 28 wird über das Element `HelpFunctions (?hf-obj` ist die Referenz des korrespondierenden Java-Objekts, welches diverse Hilfsfunktionen bereitstellt) `isEqual` aufgerufen, um zu prüfen, ob die jeweiligen Instanzen identisch sind. Gibt die Funktion `true` zurück, wird die Instanz aus der Menge der Vereinigung sämtlicher abhängiger Instanzen aller Teile entfernt (zu berücksichtigen ist dabei, dass für halluzinierte Instanzen immer `true` zurückgegeben werden muss). Dies geschieht für alle `equals`-Definitionen der korrespondierenden SRWL-Regel. Dann wird geprüft, ob die Menge der verbleibenden Instanzen disjunkt ist. Ist dies der Fall, wird im `then`-Teil der eigentliche Aktionsteil ausgeführt, der der Regel 5-21 entspricht. Gibt die `isEqual`-Funktion oder die `hasNoEqualElements`-Funktion `false` zurück, wird der Thread durch die Java-Funktion als „gestorben“ markiert und nach dem Durchlauf der Regelmaschinen gelöscht.

Aggregat-Instanziierungsregeln werden für alle Aggregate der Domänen-Ontologie Σ_{Domain} generiert.

Spezialisierungsregeln. Eine Spezialisierungsregel spezialisiert eine primitive Evidenz. Dies kann passieren, wenn in der Ontologie spezialisiertere Konzepte definiert sind, als sie von der niederen Bildverarbeitungskomponente geliefert werden. Spezialisierungsregeln setzen damit die logische Inferenz der *ABox-Realisation* um (s. [27]), die für Instanzen der ABox einer OWL Ontologie automatisch inferiert werden. In Jess ist es notwendig, diesen Schritt durch Spezialisierungsregeln abzubilden. Das generische Schema für eine Spezialisierungsregel stellt sich folgendermaßen dar:

```
(01) (defrule aggregate-x-s-rule                                    (5-24)
(02)     ?e-id <- (template-x (name ?e)
(03)                  (status evidence)
(04)                  (has-property ?p)
(05)                   ...)
(06)     (template-sp (name ?p))
(07)     (not (template-sx (name ?e)))
(08)     =>
(09)     (retract ?e-id)
(10)     (assert (template-sx (name ?e)
(11)                 (status evidence)
(12)                 (has-property ?p)
(13)                  ...)))
```

Angenommen, es liegt die Evidenz `Vehicle-Positioned (template-x)` mit den Eigenschaften `Vehicle` (`?p` bezeichne eine Evidenz von `Vehicle`) und `Zone` vor. Wenn es sich bei dem Fahrzeug um einen `Tanker (template-sp)` handelt und bei der `Zone` um die `Tanking-Zone`, dann wird die Evidenz von `Vehicle-Positioned` zu `Tanker-Positioned (template-sx)` spezialisiert. Dies ist ein Abwärtsschritt in der taxonomischen Hierarchie. Der höhere Spezialisierungsgrad in der Ontologie ermöglicht eine einfachere Modellierung, da bestimmte Eigenschaften reifiziert werden, z.B. die Eigenschaft des Fahrzeugtyps im Konzept `Tanker-Positioned`. Diese können dann direkt verwendet werden, und das „Durchgreifen“ auf bestimmte Eigenschaften über möglicherweise mehrere Ebenen (auch in den SWRL-Regeln) entfällt.

Spezialisierungsregeln werden für alle primitiven Konzepte der Domänen-Ontologie Σ_{Domain} generiert, für die ein spezielleres Konzept existiert.

Aggregat-Entfaltungsregeln. Eine Aggregat-Entfaltungsregel halluziniert ein Teil eines Aggregats, wenn dieses selbst halluziniert (oder möglicherweise durch andere Inferenzmechanismen instanziiert) ist. Das generische Schema für eine Aggregat-Entfaltungsregel stellt sich folgendermaßen dar:

```
(1) (defrule aggregate-x-ae-rule                                    (5-25)
(2)     (template-x (name ?h)
(3)         (status hallucinated)
(4)         (has-part ?p))
(5)     ?p-id <- (template-p (name ?p)
(6)                  (status hypothesised))
(7)     =>
(8)     (modify ?p-id (status hallucinated)))
```

Dies ist ein Abwärtsschritt in der kompositionellen Hierarchie. Für jedes Teil des Aggregats wird eine separate Regel generiert. Diese Regeln werden dann aktiv, wenn ein Fakt nicht durch eine Aggregat-Instanziierungsregel instanziiert wurde, sondern z.B. durch sogenannte Common-Sense-Regeln, welche auf Allgemeinwissen basieren (oder andere Inferenzmechanismen) und nicht auf konkreter Evidenz. Diese Fakten bekommen den Status `hallucinated`. Wenn Teile bereits durch Evidenzzuordnung oder Aggregatinstanziierung instanziiert wurden, dann feuert die entsprechende Regel nicht; ein Mechanismus für Instanzverschmelzung ist nicht notwendig.

Im Interpretationssystem dieser Arbeit ist nur ein sehr einfacher Mechanismus zum Halluzinieren realisiert; Wenn durch das zeitliche Constraint-Netz abgeleitet wird, dass das Zeitfenster für das Auftreten einer Evidenz abgelaufen ist, wird in

der Ontologie abgefragt, ob das betreffende Ereignis bzw. der Zustand als `hallucinatable` markiert ist. Ist dies der Fall, wird das entsprechende Ereignis bzw. der Zustand halluziniert, andernfalls stirbt der Thread.

Zeitaktualisierungsregeln. Zeitaktualisierungsregeln setzen die Eigenschaft `has-finished` eines Zustands im Hypothesengraph auf `true` und speisen die Zeitmarke für das Ende des Zustands in das Constraint-Netz ein. Primitive Zustände haben eine Dauer und werden bereits zum Beginn den Regelmaschinen als Evidenz hinzugefügt. Dadurch können Evidenz-Zuordnungsregeln und Aggregat-Instanziierungsregeln feuern, bevor der primitive Zustand beendet ist. Dies ist sinnvoll, da der Anfang und das Ende eines Aggregats in beliebiger Weise von den Zeitpunkten seiner Teile abhängen können. Das Aggregat muss also nicht zwangsläufig die zeitliche konvexe Hülle seiner Teile bilden. Finalisierungsregeln setzen die Eigenschaft `has-finished` für ein Aggregat auf `true`, wenn die aktuelle Zeit die Endzeit des Aggregats im Constraint-Netz überschreitet. Auf diese Weise kann ein Aggregat vollständig instanziiert werden, obwohl zugehörige primitive Zustände noch nicht beendet sind. Beispielsweise ist der Vorgang `Aircraft-Arrival` beendet, wenn sein Teil `Aircraft-Positioned` begonnen hat, das Flugzeug also positioniert ist. Wenn das Aktualisieren zu einer Inkonsistenz des Constraint-Netzes führt, stirbt der Thread.

5.2.3 Das zeitliche Constraint-Netz

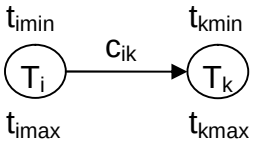
Zeitliche Constraints sind, neben räumlichen Constraints, in der dynamischen Szeneninterpretation von entscheidender Bedeutung. In dieser Arbeit wird zum Modellieren und Verarbeiten der zeitlichen Constraints die *konvexe Zeitpunktalgebra* [103] verwendet. Naheliegend für eine ontologische Modellierung scheint zunächst die Verwendung der *SWRLTemporalOntology* [14] zu sein. Da sie jedoch auf Allens Intervallalgebra [24] basiert, eignet sie sich nur zur Verarbeitung qualitativer zeitlicher Beziehungen. Die Komplexität der hier im Fokus stehenden Domänen erfordert jedoch quantitative zeitliche Modelle.

Die Grundlage der konvexen Zeitpunktalgebra sind Ungleichungen der Form:

$$T_i - T_k \leq c_{ik}\,, \tag{5-26}$$

mit den Zeitpunkten T_i und T_k, die durch integerwertige Intervalle $[t_{min} \ldots t_{max}]$ beschrieben werden und c_{ik}, einer integerwertigen Konstante. Das Intervall bezeichnet mit $t_{i\,min}$ den minimalen und mit $t_{i\,max}$ den maximalen Wert, den der Zeitpunkt T_i annehmen kann, um mit den modellierten Constraints konsistent zu

sein. Jedes konzeptuelle Objekt wird durch einen Start- und Endzeitpunkt beschrieben (wie bereits erwähnt, haben aus Vereinfachungsgründen auch Ereignisse einen Start- und Endzeitpunkt, welche immer identisch sind). Die zeitlichen Constraints eines Aggregats können durch einen gerichteten bewerteten Graph $G=(V,E)$ repräsentiert werden, mit einer Menge V von Knoten, die den Zeitpunkten entsprechen und einer Menge E von bewerteten Kanten, welche die zeitlichen Abstände (Offsets) darstellen:



(5-27)

Ein derartiger Graph wird als zeitliches Constraint-Netz (ZCN) bezeichnet. Ein ZCN wird für jedes Teilmodell generiert. Dazu wird zunächst ein ZCN für jedes Aggregat `C` der Ontologie erzeugt. Dabei werden die Definitionen der zugehörigen SWRL-Regel und aller SWRL-Regeln von Konzepten, die höher in der taxonomischen Hierarchie stehen, also $D_i \in (\Sigma_{Domain} \cup \Sigma_{Upper})$ mit $C \sqsubseteq D_i$, ausgewertet. Dadurch wird die in Abschnitt 5.1 beschriebene Vererbung von Constraints gewährleistet. Durch die Verwendung abstrakter Handlungsmuster der Top-Level-Ontologie müssen auch deren SWRL-Regeln in Betracht gezogen werden, daher reicht es nicht aus, nur die Domänen-Ontologie auszuwerten.

Schließlich werden die einzelnen ZCNs zu einem ZCN für das Teilmodell zusammengesetzt. Abbildung 27 zeigt den vereinfachten Algorithmus im Pseudocode, der rekursiv auf der Struktur des Hypothesengraphs arbeitet. Die Funktion `gerneriereZCN` wird mit dem Wurzelknoten des Hypothesengraphs und einem leeren ZCN aufgerufen, welches am Ende das ZCN des Teilmodells enthält. Beim Zusammenfügen der ZCN werden Start- und Endzeitpunkt eines Blattkonzepts im zu generierenden ZCN mit dem ZCN des entsprechenden Aggregats verschmolzen. Die zeitlichen Constraints können in spezialisierten Aggregaten verschärft werden. Beispielsweise könnte ein Aggregat `C` ein Unterkonzept von `TwoSequence` (s. Formel 5-5) sein und das zeitliche Constraint dahingehend verschärfen, dass zwischen den Teilen ein Offset von mindestens zwei Minuten liegen muss. Daher kann es bei dem Zusammenfügen der ZCN vorkommen, dass zwei Kanten c_{ij} und c_{kl} verschmolzen werden müssen. In diesem Fall bekommt die Kante den strikteren Wert $max(c_{ij}, c_{kl})$.

```
generiereZCN(Knoten, globalesZCN)
{
    forall(Teilknoten von Knoten)
    {
        if(Teilknoten noch nicht besucht)
        {
            hole ZCN vom Teilknoten

            if(Teilknoten ist Blatt vom Hypothesengraph)
            {
                nimm nur die Wurzelnoten des ZCN
            }

            verschmelze ZCN des Teilkonzepts mit dem globalen ZCN

            generiereZCN(Teilknoten, globalesZCN);
        }
        else
        {
            Knoten ist schon besucht worden (mehrfache Verwendung eines
            Teilkonzepts) -> erzeuge kein neues ZCN, aber verschmelze
            oder erzeuge möglicherweise Kanten
        }
    }
}
```

Abbildung 27: Rekursiver Algorithmus zum Generieren des ZCN eines Teilmodells aus den einzelnen ZCN der Aggregate.

Initial sind alle Intervalle der Zeitpunkte offen, d.h. $[-\infty\ +\infty]$. Durch das Zuweisen von Evidenz bzw. das Feuern von Zeitaktualisierungsregeln werden die zugehörigen Zeitstempel in das jeweilige ZCN des Teilmodells eingespeist und bekommen konkrete Werte, d.h. für sie gilt $t_{min} = t_{max}$. Diese neuen Werte werden anschließend folgendermaßen durch das Netz propagiert (s. [80]):

- Minima in Kantenrichtung: $t_{k\,min}' = max(t_{k\,min}, t_{i\,min} + c_{ik})$, (5-28)
- Maxima entgegen der Kantenrichtung: $t_{i\,max}' = min(t_{i\,max}, t_{k\,max} - c_{ik})$. (5-29)

Durch das Propagieren können sich Intervalle anderer Zeitpunkte weiter einschränken. Dies ist wichtig für eine Vorhersage des weiteren zeitlichen Verlaufs der Szene und für das Halluzinieren von Ereignissen oder Zuständen. Mögliche Zeitintervalle von Zeitpunkten noch nicht initialisierter Aggregate oder primitiver Konzepte können aus dem propagierten ZCN direkt entnommen werden. Dadurch ist das System jederzeit in der Lage, eine zeitliche Vorhersage zu generieren, beispielsweise für die Dauer der vollständigen Flugzeugabfertigung.

Das ZCN ist konsistent, wenn für alle $T_i \in V$ gilt: $t_{max} \geq t_{min}$. Zyklen im ZCN sind erlaubt und werden oftmals benötigt, da minimale Offsets in Kantenrichtung beschrieben werden, maximale entgegengesetzt. Seien $c_{12}, c_{23}, \ldots, c_{kl}$ die Offsets eines Zyklus, dann gilt für

$$\sum c_{ij} > 0\,, \tag{5-30}$$

dass jede Propagierung im ZCN zu einer Inkonsistenz führt. Diese Bedingung wird in der Initialisierungsphase des Systems für jedes ZCN geprüft und führt zu einer Fehlermeldung.

Es kann gezeigt werden, dass bei der Propagierung neuer Werte jede Kante nur einmal durchlaufen werden muss (s. [80]), daher ist die Komplexität dieser Operation in einem ZCN mit N Knoten $O(N^2)$. Im Normalfall wird ein ZCN jedoch ein dünner Graph sein, so dass die Operation wesentlich schneller ist. Das Auftreten von Zyklen beeinflusst die Komplexität nicht.

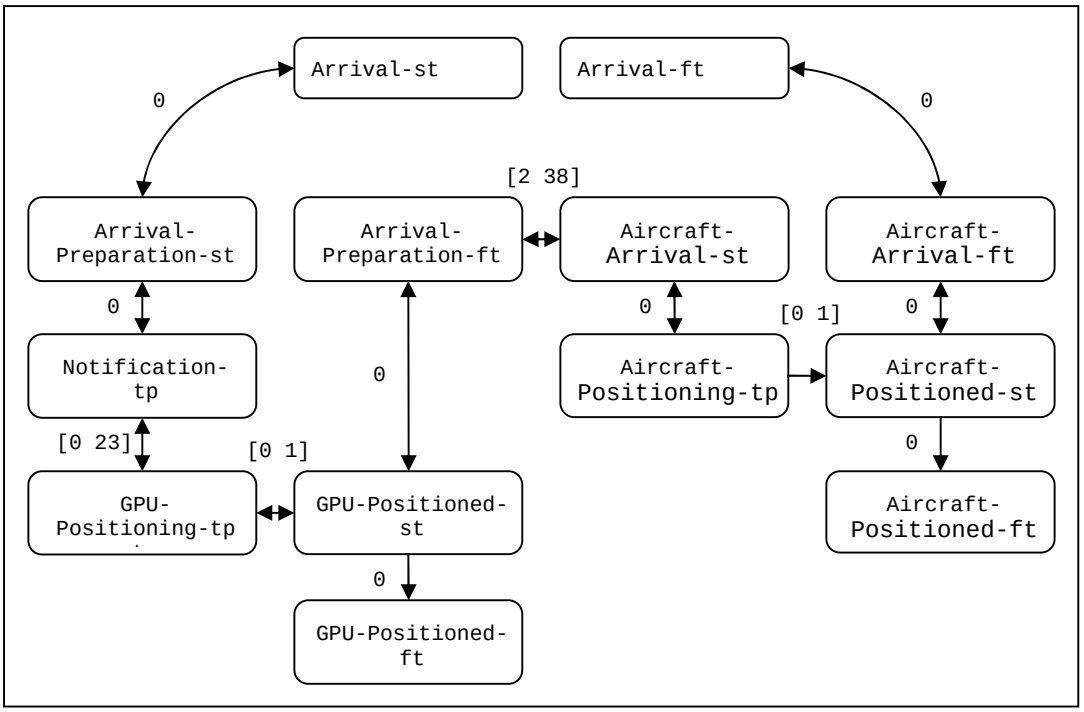


Abbildung 28: Zeitliches Constraint-Netz des Aggregats `Arrival`.

Abbildung 28 zeigt das zeitliche Constraint-Netz des Aggregats `Arrival`, dessen Hypothesengraph in Abbildung 25 vorgestellt wurde. Das Suffix „st“ bezeichnet einen Startzeitpunkt und „ft“ einen Endzeitpunkt. Zur besseren Übersicht sind

Ereignisse als einzelne Zeipunkte mit dem Suffix „tp“ dargestellt. In Pfeilrichtung werden minimale Offsets ausgedrückt, entgegen der Pfeilrichtung maximale Offsets. Die Bezeichnung `[2 38]` bedeutet, dass zwischen dem Ende von `Arrival-Preparation` und dem Beginn von `Aircraft-Arrival` mindestens zwei Minuten, höchstens jedoch 38 Minuten liegen. Ein Offset von Null in beiden Richtungen drückt die Gleichheit der verbundenen Zeitpunkte aus.

5.2.4 Verifikation der Regelbasis

In diesem Abschnitt wird untersucht, inwieweit die ontologiebasierte Architektur und der automatische Transformationsprozess des Interpretationssystems die Verifikation der erzeugten Regelbasis unterstützt.

Szeneninterpretationssysteme werden vorzugsweise für innovative Anwendungen in dynamischen Umgebungen entwickelt. Änderungen in den Domänenmodellen sind somit keine Seltenheit. Weiterhin ist die Entwicklung einer Wissensbasis oftmals ein inkrementeller Prozess, charakterisiert durch Verbesserungen und Verfeinerungen. Die Erfahrungen im Projekt Co-Friend bestätigen dies. Die einfache Anpassung der Wissensbasis an die Veränderungen der Domäne bzw. das Verständnis der Domäne ist daher eine wichtige Anforderung für den Erfolg eines Szeneninterpretationssystems.

Mit der Verwendung von OWL als einem inhärenten Bestandteil der Architektur des hier vorgestellten Szeneninterpretationssystems wird versucht, dieser Anforderung gerecht zu werden und oft genannten Nachteilen regelbasierter Systeme entgegenzuwirken (s. [71], [92]):

- Umfangreiche Regelbasen sind schwer zu warten und zu verifizieren.
- Manuell erstellte regelbasierte Systeme bieten keine klare Struktur.

Wichtige Kriterien für die Verifikation eines manuell erstellten regelbasierten Systems sind nach Ligęza [71]:

- Redundanzfreiheit (identische Redundanz, subsumierende Redundanz, äquivalente Regeln),
- Konsistenz (Indeterminismus, Konflikt, Inkonsistenz),
- Vollständigkeit.

5.2.4.1 Redundanz

Zur Betrachtung der Redundanz werden zunächst die Begriffe der *logischen Redundanz* und der *funktionellen Redundanz* definiert:

Definition 5-1: Logische Redundanz

Eine Wissensbasis repräsentiert durch die Regelbasis R ist logisch redundant, wenn ein R' existiert, welches aus R durch Entfernen einer Regel r entsteht und es gilt: $R \models R'$ und $R' \models R$.

Definition 5-2: **Funktionelle Redundanz**

Eine Wissensbasis repräsentiert durch die Regelbasis R ist funktionell redundant, wenn ein R' existiert, welches aus R durch Entfernen einer Regel r entsteht und sich exakt wie R verhält, d.h. für jeden Input der Faktenbasis FB gilt: $R(FB) \equiv R'(FB)$.

Mit anderen Worten, eine Regelbasis R ist dann funktionell redundant, wenn eine Regel r entfernt werden kann, ohne das Verhalten des Systems zu ändern. Durch den komplexen Aktionsteil der hier betrachteten Regeln und die Verknüpfung mit den prozeduralen Berechnungen des zeitlichen Constraint-Netzes ist die Betrachtung der logischen Redundanz nicht direkt anwendbar, weshalb hier ausschließlich die funktionelle Redundanz untersucht wird.

Identische Regeln. Die einfachste Form der Redundanz in einer Regelbasis sind identische Regeln. Da die Aggregatdefinitionen in der OWL Wissensbasis die Grundlage für die Erzeugung aller Regeln darstellen, werden keine identischen Regeln erzeugt, wenn es keine identischen Aggregate gibt. Dies kann einfach durch Reasoner wie Racer und Pellet überprüft werden.

Subsumption. Ein interessanterer Fall ist die Subsumption von Regeln. Von der logischen Betrachtungsweise subsumiert eine Regel $r_1 : c_1 \rightarrow h_1$ eine andere Regel $r_2 : c_2 \rightarrow h_2$, wenn sie aus schwächeren Prämissen stärkere (oder gleiche) Konklusionen ableiten kann, d.h. $c_2 \models c_1$ und $h_1 \models h_2$. In diesem Fall ist die subsumierte Regel r_2 redundant und könnte entfernt werden. Übertragen auf die funktionelle Redundanz kann hier die Frage gestellt werden, ob es Regeln $r_1 : c_1 \rightarrow h_1$ und $r_2 : c_2 \rightarrow h_2$ gibt, für die $c_2 \models c_1$ gilt und bei denen h_1 und h_2 zum gleichen Verhalten des Systems führen. In diesem Fall wäre die Regel r_2 redundant. Betrachten wir also zunächst, ob es Regeln gibt, die $c_2 \models c_1$ erfüllen.

Für die Evidenz-Zuordnungsregeln gilt, dass sie jeweils nur für die generellsten Konzepte der Domänen-Ontologie Σ_{Domain} generiert werden, d.h., es gibt keine Evidenz-Zuordnungsregeln für Konzepte A und B mit $A \sqsubseteq B$, dies wäre jedoch die Voraussetzung, um $c_2 \models c_1$ zu erfüllen.

Aggregat-Instanziierungsregeln werden nur für die Blattkonzepte der taxonomischen Hierarchie von komplexen Ereignissen und Zuständen generiert. Grundsätzlich gilt also, dass es keine Konzepte A und B mit $A \sqsubseteq B$ gibt, für die Aggregat-Instanziierungsregeln erzeugt werden, und dies wäre die Voraussetzung für die Erzeugung subsumierender Regeln. Jedoch kann es bei alternativen Modellen vorkommen, dass es subsumierende Konzepte gibt, die als Blattkonzepte in der Hierarchie auftreten. Betrachtet man beispielsweise die Teilmodelle `Service-With-Refuelling` (C_1) und `Service-Without-Refuelling` (C_2), dann gilt:

$$\begin{aligned} \texttt{Service-With-Refuelling} &\sqsubseteq \texttt{Service} \\ \texttt{Service-Without-Refuelling} &\sqsubseteq \texttt{Service} \end{aligned} \tag{5-31}$$

Eine vergleichbare Situation ist in Abbildung 23 dargestellt. Nun kann jedoch, abhängig von der Modellierung, auch $C_1 \sqsubseteq C_2$ gelten. Werden die SWRL-Regeln für C_1 und C_2 hier außer Acht gelassen oder wird angenommen, dass sich ihre Prämissen ebenfalls subsumieren, dann gilt für die entsprechenden Jess-Regeln: $c_1 \models c_2$. Initial werden für die Teilmodelle C_1 und C_2 separate Threads mit eigenen Hypothesengraphen erzeugt. Da alle Threads mit der kompletten Regelbasis ausgestattet werden, stellt sich die Frage, ob die Aggregat-Instanziierungsregel für C_2 im Thread des Teilmodells für C_1 feuern kann. Dies wäre ein unerwünschtes Systemverhalten. Doch das ist nicht der Fall, da in der Prämisse von c_2 auf das Hypothesenelement von `Service-Without-Refuelling` abgeglichen wird und dieses Element ist im Thread von C_1 nicht enthalten. Betrachtet man dagegen ein hypothetisches Modell für `Turnaround`, welches C_1 *und* C_2 enthält, dann wären beide Hypothesenelemente der Konzepte C_1 und C_2 im entsprechenden Thread existent. Da es sich jedoch um Teilmodelle handelt, sind diese Konzepte lediglich Blätter des Hypothesengraphs für `Turnaround` und die Aggregat-Instanziierungsregeln sind dann nicht relevant. Dieses Beispiel zeigt, dass der Mechanismus der Hypothesengraphen und Teilmodelle für die korrekte Verarbeitung der Aggregat-Instanziierungsregeln sorgt, selbst wenn sich deren Prämissen subsumieren.

Aggregat-Entfaltungsregeln werden für alle Aggregate und deren Teile erzeugt. Analog zur Argumentation der Aggregat-Instanziierungsregeln wird die korrekte Verarbeitung der Regeln durch Hypothesengraphen und Teilmodelle sichergestellt,

selbst wenn es alternative Teilmodelle C_1 und C_2 gibt, für deren Wurzelkonzept $C_1 \sqsubseteq C_2$ gilt.

Spezialisierungsregeln werden für alle primitiven Konzepte der Domänen-Ontologie Σ_{Domain} generiert, für die ein spezielleres Konzept existiert. Vereinfacht dargestellt werden für die Konzepte A, B und C mit $A \sqsubseteq B$ und $B \sqsubseteq C$ Regeln $r_1 : c_1 \rightarrow h_1$ und $r_2 : c_2 \rightarrow h_2$ generiert, die eine Instanz c zu b, bzw. b zu a spezialisieren (Kleinbuchstaben beschreiben Instanzen der Klassen mit dem entsprechenden Großbuchstaben). Die taxonomische Hierarchie der Ontologie bleibt in Jess durch die Template-Struktur erhalten, so dass hier ohne weitere Maßnahmen $c_2 \models c_1$ gelten würde. Durch die Verwendung des NOT-Befehls (siehe Regel 5-24) wird genau dies verhindert und die korrekte Spezialisierung gewährleistet.

Äquivalente Regeln. Der Fall der äquivalenten Regeln kann auf die Subsumption von Regeln zurückgeführt werden, da zwei Regeln äquivalent sind, wenn sie sich gegenseitig subsumieren.

Ausgehend von einer OWL Ontologie, die frei von Redundanz ist (dies kann weitgehend durch verfügbare Reasoner festgestellt werden), wird also durch die automatische Transformation gewährleistet, dass die erzeugte Regelbasis ebenfalls redundanzfrei ist.

5.2.4.2 Konsistenz

Zur Betrachtung der Konsistenz werden Indeterminismus, Konflikt und die Inkonsistenz einer Regelbasis untersucht.

Definition 5-3: Indeterminismus

Zwei Regeln $r_1 : c_1 \rightarrow h_1$ *und* $r_2 : c_2 \rightarrow h_2$ *sind nichtdeterministisch, falls ein Zustand beschrieben durch die Formel* ϕ *existiert, so dass gleichzeitig* $\phi \models c_1$ *und* $\phi \models c_2$ *und* $h_1 \neq h_2$ *gilt.*

Mit anderen Worten, nichtdeterministische Regeln können gleichzeitig feuern, aber ihre Konklusionen sind verschieden. Wie bereits erwähnt ist die Mehrdeutigkeit eine inhärente Eigenschaft der Szeneninterpretation. Evidenz-Zuordnungsregeln sind grundsätzlich mehrdeutig, da es mehrere Zuordnungsmöglichkeiten innerhalb eines Modells und zwischen alternativen Modellen geben kann. Alternative Modelle werden in separaten Threads mit eigener Regelmaschi-

ne verwaltet, so dass nur der Fall der Mehrdeutigkeit innerhalb eines Modells betrachtet werden muss. Diese ist eine gewünschte Eigenschaft des Interpretationsmechanismus. Werden mehrere Evidenz-Zuordnungsregeln gleichzeitig aktiviert, wird der Thread entsprechend oft geklont und jede mögliche Zuordnung in einem Thread realisiert (der Interpretationsprozess wird in Abschnitt 5.3 im Detail behandelt).

Bei Aggregat-Instanziierungsregeln und Aggregat-Entfaltungsregeln werden Mehrdeutigkeiten durch die Verwendung von Hypothesengraphen und den Mechanismus der Aufspaltung des Suchbaums in alternative Interpretationen verhindert (s. auch Abschnitt 5.2.4.1 und 5.3).

Eine mögliche Mehrdeutigkeit bei den Spezialisierungsregeln wird durch Verwendung des `NOT`-Befehls (siehe Regel 5-24**)** verhindert.

Definition 5-4: Konflikt

Zwei Regeln $r_1 : c_1 \rightarrow h_1$ *und* $r_2 : c_2 \rightarrow h_2$ *stehen in Konflikt, falls ein Zustand beschrieben durch die Formel* ϕ *existiert, so dass gleichzeitig* $\phi \models c_1$ *und* $\phi \models c_2$ *gilt, aber* $\not\models_I h_1 \wedge h_2$ *unter der Interpretation* I *.*

Mit anderen Worten, zwei Regeln können gleichzeitig feuern, aber ihre Konklusionen stehen in Konflikt, d.h., h_1 und h_2 können nicht gleichzeitig wahr sein. Zum Beispiel kann ein Fahrzeug zu einem bestimmten Zeitpunkt nicht gleichzeitig innerhalb und außerhalb einer Zone sein.

Ein Konflikt kann durch Nichtdeterminismus entstehen. Da dieser jedoch in der automatisch erzeugten Regelbasis nicht auftritt bzw. durch das Aufspalten in separate Threads kontrolliert wird, kann es nicht zu Konflikten aufgrund von Nichtdeterminismus kommen. Bei der hier erzeugten Regelbasis handelt es sich jedoch um Produktionsregeln mit Attributlogik, mit einer sich dynamisch verändernden Faktenbasis, die zusätzlich mit einem zeitlichen Constraint-Netz verknüpft ist. Die Frage nach der Konfliktfreiheit der Faktenbasis ist hier also komplexer, denn der Inferenzprozess eines derartigen Regelsystems ist nicht zwangsläufig monoton (s. Abschnitt 4.3.2).

In dem hier vorliegenden System spiegelt die Faktenbasis eines Threads den Zustand des entsprechenden Hypothesengraphs wieder. Dabei ist ein Zustand charakterisiert durch die Instanziierungen von primitiven Konzepten und Aggregaten. Der hier entwickelte Interpretationsprozess sieht nicht vor, dass eine Instanziierung zurückgenommen wird, daher ist der Inferenzprozess monoton (s. Abschnitt 5.3), und es kann kein Konflikt entstehen, vorausgesetzt die Modelle der Wissens-

basis sind konfliktfrei. Potentielle Konflikte können durch das Löschen von Fakten entstehen, welches nur bei Spezialisierungsregeln angewandt wird (s. Regel 5-2). Doch auch hier bleibt die Monotonie erhalten, da lediglich Instanzen spezialisiert werden und alle Attributwerte erhalten bleiben, d.h., diese spezialisierten Instanzen (bzw. Fakten) können nicht im Widerspruch zur Faktenbasis stehen. Das Löschen von Fakten ist hier nur eine technische Notwendigkeit, um ein Fakt einem spezielleren Template zuzuordnen.

Definition 5-5: Inkonsistenz

Zwei Regeln $r_1 : c_1 \rightarrow h_1$ *und* $r_2 : c_2 \rightarrow h_2$ *sind inkonsistent, falls ein Zustand beschrieben durch die Formel* ϕ *existiert, so dass gleichzeitig* $\phi \models c_1$ *und* $\phi \models c_2$ *gilt, aber* $\not\models h_1 \wedge h_2$.

Mit anderen Worten, zwei Regeln können gleichzeitig feuern, aber ihre Konklusionen sind logisch inkonsistent, zum Beispiel $h_1 \wedge \neg h_2$. Die Inkonsistenz ist ein Spezialfall des Konflikts und kann daher auf die obigen Betrachtungen zur Konfliktfreiheit zurückgeführt werden.

5.2.4.3 Vollständigkeit

Das Problem der Verifikation der Vollständigkeit kann nach [71] definiert werden als Überprüfung, ob es für jede beliebige Kombination von Eingaben mindestens eine Regel gibt, deren Prämisse erfüllt ist.

Sei die Regelbasis gegeben durch:

$$\begin{aligned} & r_1 : c_1 \rightarrow h_1, \\ & r_2 : c_2 \rightarrow h_2, \\ & \ldots \\ & r_m : c_m \rightarrow h_m. \end{aligned} \tag{5-32}$$

Definition 5-6: Logische Vollständigkeit

Eine Regelbasis $R = \{r_1, r_2, ..., r_m\}$ *ist logisch vollständig, wenn gilt:*

$$\models c_1 \vee c_2 \vee ... \vee c_m , \tag{5-33}$$

d.h. die Vereinigung der Prämissen der Regeln ist eine Tautologie.

Die logische Vollständigkeit der Regelbasis ist dahingehend gewährleistet, dass die Evidenz-Zuordnungsregeln für alle von der Mittelschicht gelieferten Ereignisse bzw. Zustände feuern, die Instanzen von Konzepten der OWL Wissensbasis sind. Das gilt für jeden separaten Thread, denn Evidenzen, die nicht zum Modell zugeordnet werden können, entweder weil das entsprechende primitive Konzept im Modell nicht existiert oder weil die Constraints nicht erfüllt sind, werden dem speziellen Konzept *Clutter* zugeordnet.

Doch die Betrachtung der logischen Vollständigkeit ist hier eher von theoretischem Interesse. Wichtig ist, dass das System (prinzipiell) alle möglichen Interpretationen liefern kann, die bei gegebener Evidenz unter Beachtung aller Constraints konsistent sind mit den Definitionen der OWL Ontologie. Dies kann zumindest prinzipiell dadurch gewährleistet werden, dass der Transformationsprozess Regeln für alle taxonomischen und kompositionellen Strukturen der Wissensbasis generiert.

Abschließend kann festgehalten werden, dass der automatische Transformationsprozess eine vergleichsweise einfache Anpassung der Ontologie an eine dynamische Domäne ermöglicht und in vielen Aspekten die Konsistenz und Vollständigkeit der erzeugten Regelbasis garantieren kann.

5.3 Interpretationsprozess

Der Interpretationsprozess von SCENIOR besteht aus zwei Phasen:

- der Initialisierungsphase,
- der Interpretationsphase.

Die Funktionsweise beider Phasen wird in diesem Abschnitt ausführlich erläutert.

5.3.1 Initialisierungsphase

Die Initialisierungsphase gliedert sich in folgende wesentliche Schritte:

- Die konzeptuelle Wissensbasis, formuliert in OWL und SWRL, wird automatisch in die konzeptuelle Jess Wissensbasis übersetzt. Dabei werden alle nötigen Templates und Regeln generiert, gemäß Abschnitt 5.2.2.
- Für jedes Teilmodell – dessen Wurzelkonzept in der Ontologie durch die Eigenschaft `context-free` markiert ist – wird ein Hypothesengraph erstellt.
- Basierend auf den Definitionen der SWRL-Regeln werden zunächst zeitliche Constraint-Netze (ZCNs) für jedes Aggregat generiert (s. Abschnitt 5.2.3). Anhand der Struktur der Hypothesengraphen wird anschließend aus den einzelnen ZCNs ein zusammenhängendes ZCN für das Teilmodell generiert (s. Abbildung 27).
- Zur Modellierung der probabilistischen Modelle werden in diesem System Bayes'sche Kompositionelle Hierarchien (BCHs) verwendet (s. Abschnitt 2.4.3). Die Funktionsweise und Integration der BCHs in SCENIOR wird im Detail in Abschnitt 5.4 erläutert. Sie werden ebenfalls für jedes Teilmodell generiert. Die probabilistischen Modelle sind nicht in der Ontologie enthalten (s. Abschnitt 5.1.3), sondern in separaten Dateien, die während der Initialisierung eingelesen werden. Dabei wird überprüft, ob die Definitionen der Modelle mit denen in der Ontologie konsistent sind.
- Ein separater Interpretations-Thread wird für jedes Teilmodell initialisiert. Jeder Thread hat eine eigene unabhängige Jess-Regelmaschine und wird mit dem entsprechenden Hypothesengraphen, dem ZCN und der BCH ausgestattet.

5.3.2 Interpretationsphase

Nach Abschluss der Initialisierungsphase ist das System bereit für den Interpretationsprozess, dieser ist in Abbildung 30 illustriert.

5.3.2.1 Einlesen primitiver Evidenzen

Wie bereits in Abschnitt 5.2.1 dargelegt, erwartet SCENIOR die primitiven Evidenzen Frame-basiert, wie sie bei der dynamischen Szeneninterpretation typischerweise von der niederen Bildverarbeitungskomponente und der Mittelschicht geliefert werden. Jeder Frame enthält dabei beliebig viele primitive Evidenzen,

deren Konzepte mit den Definitionen in der Ontologie übereinstimmen müssen. Diese Evidenzen sind also Instanzen von Konzepten `E`, für die gilt:

$$\texttt{E} \sqsubset (\texttt{Primitive-Event} \sqcup \texttt{Primitive-State}). \quad (5\text{-}34)$$

Die Evidenzen bzw. die Frames sind mit einem quantitativen Zeitstempel versehen: Primitive Ereignisse sind punktuell und kommen nur in einem Frame vor, während primitive Zustände in einer beliebigen Anzahl von aufeinanderfolgenden Frames geliefert werden. Die Zeitstempel des ersten und des letzten Frames definieren den Anfangszeitpunkt bzw. den Endzeitpunkt des Zustands. Des Weiteren ist jede Evidenz mit einer eindeutigen Identifikation (ID) in Form eines Integerwerts gekennzeichnet. Darüber hinaus können die Evidenzen mit weiteren Eigenschaften versehen sein. Primitive Konzepte sind dadurch charakterisiert, dass sie keine konzeptuellen Objekte als Teile haben, sie können jedoch physikalische Objekte besitzen. Der Relationssuffix `has-part` ist der kompositionellen Hierarchie vorbehalten, so dass hierfür andere Rollenbezeichner gewählt werden. Dem Zustand `Tanker-Positioned` beispielsweise ist ein `Tanker` zugeordnet, der durch die Eigenschaft `has-agent` definiert wird und eine `Zone`, die durch `has-location` definiert wird. Die physikalischen Objekte könnten weitere Eigenschaften, wie Farbe, Größe etc., besitzen.

Die primitiven Ereignisse und Zustände werden in SCENIOR zunächst in eine Eingabe-Queue geladen. Diese arbeitet nach dem FIFO-Prinzip. Damit wird eine korrekte Verarbeitung sichergestellt, auch wenn der Interpretationsprozess temporär der Lieferung neuer Evidenzen nachläuft. Von der Eingabe-Queue werden die Evidenzen eines Frames dann mit Hilfe des *Jess-Fakten-Konverters* in Jess-Fakten umgewandelt. Das gilt auch für die zugehörigen physikalischen Objekte. Diese werden von SCENIOR verwaltet, so dass für mehrfach referenzierte Objekte nur ein Fakt generiert wird.

Schrittweise werden die Evidenzen in die Arbeitsspeicher aller aktiven Threads gespeist. Zu Beginn sind dies die initialen Threads, welche für jedes Teilmodell generiert wurden. Nachdem eine Evidenz eingespeist wurde, werden alle Regelmaschinen parallel gestartet. Der Interpretationsprozess ist als synchronisierte Parallelverarbeitung organisiert, d.h., es wird solange mit der weiteren Verarbeitung gewartet, bis alle Regelmaschinen ihren Durchlauf beendet haben, also keine Regel mehr feuert. Dieser Vorgang wird als ein *Zyklus* definiert. Die Abarbeitung eines Frames kann somit aus mehreren Zyklen bestehen, abhängig von den darin enthaltenen Evidenzen. Die Synchronisation der Threads ist notwendig, da nach Abschluss jedes Zyklus möglicherweise die probabilistischen Bewertungen der Alternativen miteinander verglichen oder höhere Evidenzen weitergegeben werden müssen. Wie in Abschnitt 5.2.2.4 erläutert, werden primitive Zustände bereits zum Anfangszeitpunkt an die Regelmaschine weitergereicht. SCENIOR speichert alle angefangenen Zustände und registriert, wenn ein bestimmter Zustand in ei-

nem Frame nicht mehr existiert. Dann wird der Endzeitpunkt für die Evidenz dieses Zustands in allen Threads eingetragen (dies geschieht im Java Hintergrund) und die Eigenschaft `has-finished` auf `true` gesetzt. Die Zeitaktualisierungsregeln reichen diese Information im Hypothesengraph jedes Threads weiter und tragen den Wert in das entsprechende ZCN und die BCH ein. Wird das ZCN dadurch inkonsistent, stirbt der Thread.

Ereignisse repräsentieren also Aussagen über Zeitpunkte und Zustände über Zeitintervalle. Wenn ein Objekt nicht mehr detektiert wird (z.B. weil es sich nicht mehr in der Szene befindet), dann wird dadurch der Endzeitpunkt des entsprechenden primitiven Zustands gesetzt; es wird jedoch keine Evidenz-Zuweisung zurückgenommen. Da die Aussage (z.B. `Vehicle-Inside-Zone`) nur für das entsprechende Zeitintervall gilt, bleibt die Monotonie erhalten.

5.3.2.2 Aufbau des Suchbaums

Wie bereits in Abschnitt 5.1.3 angedeutet, kann es vorkommen, dass eine Evidenz mehrfach in einem Teilmodell oder in verschiedenen Teilmodellen zugeordnet werden kann. Die Zuordnungen zu alternativen Modellen werden durch die Verarbeitung der parallelen Threads behandelt, daher können die Betrachtungen hier auf ein Teilmodell beschränkt werden.

Besitzt ein Teilmodell der Wissensbasis mehrere Blätter desselben Konzepts, so kann es grundsätzlich mehrere Möglichkeiten der Evidenzzuweisung geben, sofern die Constraints erfüllt sind. Beispielsweise könnte ein Aggregat zwei `Vehicle-Positioning`-Konzepte beinhalten, wenn zwei Fahrzeuge in der Aktivität involviert sind. Durch die Verwendung eindeutiger Rollenbezeichner können auch gleiche Konzepte als Rollenfüller unterschieden werden. Zusätzlich gibt es immer die Möglichkeit, dass die Evidenz dem speziellen Konzept *Clutter* zugewiesen wird, welches ein sehr einfaches Störungsmodell realisiert. Dieses Konzept ist automatisch Bestandteil jedes Teilmodells. In Abbildung 29 ist die Situation schematisch dargestellt.

Bei der Art der Störungen kann zwischen zwei prinzipiellen Fällen unterschieden werden. Zum Einen kann es sich dabei um Evidenzen von Aktivitäten handeln, die nicht in der Ontologie modelliert wurden. Der Grund dafür können unvorhersehbare, ungeplante Ereignisse sein, z.B. ein Fahrzeug, welches über das Vorfeld fährt und nicht Teil der Serviceaktivitäten ist.

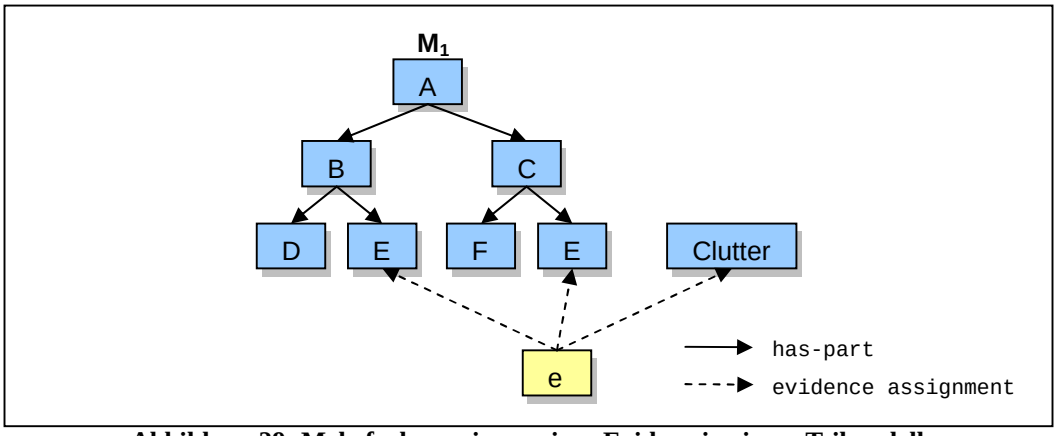


Abbildung 29: Mehrfachzuweisung einer Evidenz in einem Teilmodell.

Weiterhin beschreiben Modelle immer nur eine vereinfachte Version der komplexen realen Welt. Die ersten Versionen der Ontologie im Projekt Co-Friend stellten sich beispielsweise als zu ambitioniert und komplex heraus; sie beinhalteten Aktivitäten von Personen, die nicht robust von der niederen Bildverarbeitung detektiert werden konnten. Zum Anderen sind es ebenfalls Probleme der Bildverarbeitungskomponenten, die durch typische Schwierigkeiten in komplexen Domänen zur Lieferung fehlerhafter Evidenzen führen. Dazu gehören z.B. ungünstige Licht- und Wetterverhältnisse, Verdeckungen und generelle Tracking- und Klassifikationsprobleme von Bildverarbeitungsalgorithmen (s. [37], [53]).

Mehrfachzuweisungen führen zur gleichzeitigen Aktivierung mehrerer Evidenz-Zuordnungsregeln. Diese wird als Trigger genutzt, um den Thread aufzuspalten. Bei k Aktivierungen werden durch Klonen des ursprünglichen Threads $k-1$ neue Threads generiert. Jeder erhält dabei die vollständige Regelbasis und die aktuelle Faktenbasis, der den bisherigen Interpretationszustand widerspiegelt. In jedem der k Threads wird nun das Feuern *einer* Evidenz-Zuordnungsregeln forciert. Auf diese Weise wird ein Suchbaum etabliert, der (zunächst) alle Interpretationsmöglichkeiten parallel verfolgt. Durch die Verwendung des Cluttermodells wird sichergestellt, dass immer ein Thread existiert, der eine erneut auftretende Evidenz desselben Konzepts aufnehmen kann, da es sich bei der bereits zugewiesenen Evidenz um eine Störung handeln könnte. Dies bedeutet jedoch, dass jede zugewiesene Evidenz die Anzahl der Threads erhöht. Unter der Voraussetzung, dass alle Constraints erfüllt sind und es für jede Evidenz nur eine Zuordnungsmöglichkeit gibt, werden für n Evidenzen 2^n Threads erzeugt, für jedes Element der Potenzmenge der Evidenzen (s. auch Abschnitt 5.3.2.5). In Abbildung 30 wird der Mechanismus verdeutlicht.

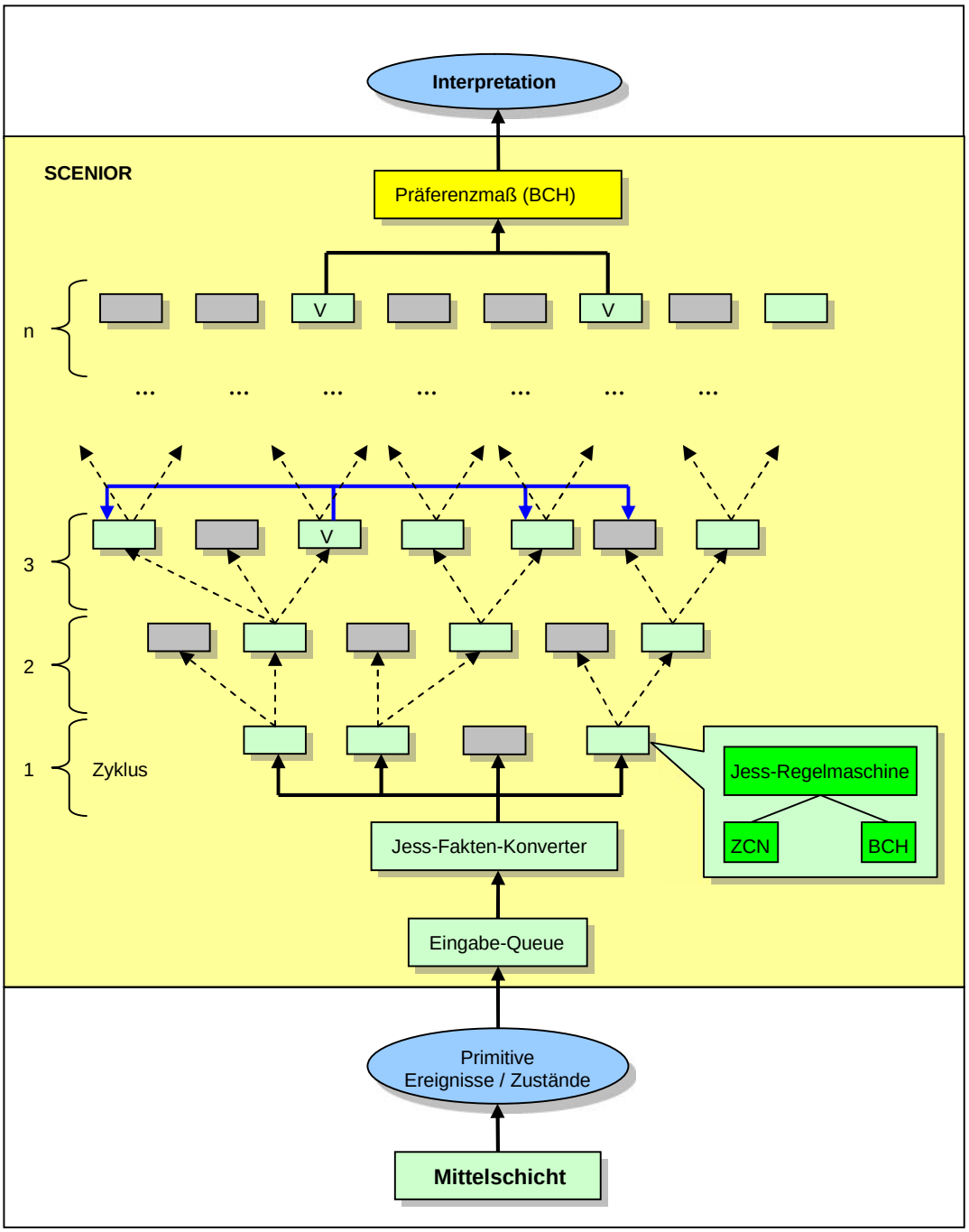


Abbildung 30: Interpretationsprozess von SCENIOR.

SCENIOR kann auf einem leistungsfähigen PC mit derzeitigem Stand der Technik mehr als 150 Threads in Realzeit parallel prozessieren. Durch das exponentielle Wachstum der Threads in Abhängigkeit von der Anzahl der Evidenzen wird diese Grenze jedoch früher oder später erreicht. Die Experimente in der Domäne der Flughafenvorfeldaktivitäten zeigen, dass bei der Interpretation eines vollständigen Servicevorgangs in allen Fällen die maximale Anzahl der Threads erreicht wird; in der Regel in der zweiten Hälfte des Ablaufs. Dies macht die Notwendigkeit eines Präferenzmaßes deutlich, mit dem die alternativen Interpretationen bewertet werden können und nur die vielversprechensten weiterverfolgt werden. Dazu wird in SCENIOR ein probabilistisches Präferenzmaß verwendet, welches auf BCH-Modellen basiert. Die prinzipielle Funktionsweise des probabilistischen Präferenzmaßes wird in Abschnitt 5.4 beschrieben.

In jedem Zyklus werden die Regeln, die in Modulen organisiert sind, in folgender Reihenfolge ausgeführt:

- Spezialisierungsregeln,
- Evidenz-Zuordnungsregeln,
- Zeitaktualisierungsregeln,
- Aggregat-Instanziierungsregeln,
- Aggregat-Entfaltungsregeln.

Evidenzen müssen zunächst möglicherweise spezialisiert werden, bevor sie zugeordnet werden können. Dann reichen Zeitaktualisierungsregeln evtl. neu gesetzte Endzeitpunkte im Hypothesengraph weiter. Anschließend können Aggregat-Instanziierungsregeln und -Entfaltungsregeln feuern. Wenn die Regelmaschinen aller aktiven Threads ihren Durchlauf beendet haben, wird überprüft, ob es Teilmodelle gibt, die vollständig instanziiert und damit erkannt sind. Für das Wurzelkonzept `C` wird dann eine sogenannte höhere Evidenz erzeugt und an alle Regelmaschinen geliefert, deren Hypothesengraphen Blätter `D` besitzen, für die $C \sqsubseteq D$ gilt (in Abbildung 30 sind vollständige Threads mit `V` gekennzeichnet und die Weitergabe der höheren Evidenzen durch blaue Pfeile dargestellt). Dies geschieht im Java Hintergrund, wie grundsätzlich alle Mechanismen, die Interaktionen zwischen den Regelmaschinen erfordern. Höhere Evidenzen werden also im Gegensatz zu primitiven Evidenzen erst dann weitergereicht, wenn sie abgeschlossen sind. Dies wird zum Einen dadurch motiviert, dass ein Teilmodell eine sinnvolle Teilaktivität darstellt, die es wert ist, als Ganzes erkannt zu werden. Zum anderen würde eine Weitergabe zum Beginn eines zusammengesetzten Ereignisses (höhere Evidenz) den Interpretationsprozess komplizierter gestalten, denn das nachträgliche Sterben des Threads (z.B. durch eine Zeitaktualisierungsregel) müsste auch zum Sterben der Threads führen, an die die Evidenz weitergereicht wurde.

5.3.2.3 Sterben von Threads

Es gibt drei verschiedene Mechanismen, welche zum Sterben eines Threads führen können (in Abbildung 30 als graue Kästchen dargestellt):

- das Setzen des Endzeitpunkts einer primitiven Evidenz,
- die Überprüfung der Erfüllbarkeit des ZCN, damit gekoppelt ist das Halluzinieren von Ereignissen und Zuständen,
- niedrige Bewertung.

Setzen des Endzeitpunkts einer primitiven Evidenz. Wenn der Endzeitpunkt einer primitiven Evidenz in das ZCN eingespeist wird und die Propagierung zu einer Inkonsistenz führt, stirbt der Thread.

Überprüfung der Erfüllbarkeit des ZCN. Hierzu wird zunächst der Begriff der Erfüllbarkeit eines ZCN definiert:

Definition 5-7: Erfüllbarkeit eines ZCN

Das ZCN eines Teilmodells ist zu einem Zeitpunkt t *nicht mehr erfüllbar, wenn die Einspeisung eines Zeitpunktes* $t_e \geq t$ *einer weiteren Evidenz zur Inkonsistenz des ZCN führt.*

Der Algorithmus zur Überprüfung der Erfüllbarkeit des ZCN eines Teilmodells ist in Abbildung 31 dargestellt. Ein Zeitpunkt (Knoten) gilt als instanziiert, wenn das korrespondierende Konzept instanziiert ist. Dabei gilt bei einem gestarteten Zustand auch nur der Startzeitpunkt als instanziiert, wenn das Ende noch offen ist.

Die Erfüllbarkeit der ZCN wird in konfigurierbaren Zeitabständen (beispielsweise alle zehn Sekunden, s. auch Anlage B.1) überprüft, um die Anzahl der aktiven Threads zu minimieren. Falls ein ZCN nicht mehr erfüllbar ist, führt dies jedoch nicht zwangsläufig zum Sterben des Threads. Der Algorithmus wird genutzt, um diejenigen Evidenzen zu identifizieren, die zum aktuellen Zeitpunkt spätestens hätten geliefert werden müssen. Für die zugehörigen Konzepte wird geprüft, ob sie in der Ontologie als `is-hallucinatable` markiert sind. Ist dies der Fall, werden die entsprechenden Evidenzen halluziniert. Falls dies für mindestens ein Konzept nicht gilt, so stirbt der Thread. Im weiteren Verlauf des Interpretationsprozesses werden halluzinierte Konzepte wie instanziierte Konzepte behandelt, z.B. bei Aggregatinstanziierungen. Die Start- und Endzeitpunkte können aus dem ZCN abgelesen werden und sind (möglicherweise) weiterhin Intervalle, gelten aber als instanziiert. Durch das Auftreten neuer Evidenzen können die Intervalle weiter eingeschränkt werden. Halluzinierte Konzepte sind darüber hinaus von

möglichen Constraint-Überprüfungen ausgenommen, d.h., es wird angenommen, dass sie alle Constraints erfüllen.

```
überprüfeErfüllbarkeitZCN(ZCN, t_aktuell)
{
    forall(Knoten T_i des ZCN)
    {
        if(T_i ist nicht instanziiert)
        {
            tmpZCN = Kopie des ZCN

            Setze t_min von T_i aus tmpZCN auf t_aktuell

            Propagiere den Wert t_min durch tmpZCN

            if(tmpZCN ist inkonsistent)
            {
                return false;
            }
        }
    }
    return true;
}
```

Abbildung 31: Algorithmus zur Überprüfung der Erfüllbarkeit eines ZCN.

Zur Überprüfung der Erfüllbarkeit des ZCN reicht es nicht aus, nur die Zeitpunkte der Blattkonzepte zu überprüfen, also derjenigen Konzepte, die durch Evidenzzuordnung instanziiert werden. Denn die Zeitpunkte eines Aggregats können in beliebiger Weise von den Zeitpunkten seiner Teile abhängen und müssen nicht deren konvexe Hülle bilden. Denkbar wären auch Teile, die überhaupt nicht durch zeitliche Constraints mit anderen Teilen oder dem Wurzelkonzept des Aggregats verknüpft sind. Sie würden durch eine Überprüfung, die sich lediglich auf die Blattkonzepte stützt, nicht identifiziert werden. Unter der Voraussetzung, dass der Endzeitpunkt des Aggregats eindeutig durch seine Teile oder durch andere Aggregate definiert ist, kann dieses durch den Algorithmus identifiziert und evtl. halluziniert werden. Durch die Aggregat-Entfaltungsregeln würden dann auch dessen Teile halluziniert, für die keine Evidenz vorläge. In diesem Fall könnten auch Teile halluziniert werden, die in der Zukunft liegen. Die Evidenz-Zuordnungsregeln erlauben deshalb die Zuordnung einer Evidenz zu einem Konzept, welches den Status `hypothesised` oder `hallucinated` hat (s. Regel 5-20).

Im Rahmen dieser Arbeit wurde nur dieser einfache Mechanismus des Halluzinierens realisiert. Daher sollte sehr sorgfältig abgewogen werden, welche Konzepte in der Ontologie als halluzinierbar definiert werden. Die Experimente der Flughafenvorfeldaktivitäten (s. Kapitel 7) zeigen jedoch, dass in dieser Domäne, und bei der unzureichenden Qualität der niederen Bildverarbeitung, ohne Halluzinieren in der Regel keine vollständige Abfertigung erkannt werden würde.

Niedrige Bewertung. Die dritte Möglichkeit, die zum Sterben eines Threads führen kann, ist eine niedrige Bewertung des Threads durch das probabilistische Präferenzmaß. Parallel zur Einspeisung von Zeitpunkten neuer Evidenz in das ZCN, werden die Zeitpunkte auch an die entsprechende BCH des Teilmodells weitergereicht. Sie berechnet ein Präferenzmaß (s. Abschnitt 5.4), welches den Thread bewertet. Überschreitet die Anzahl der Threads das Maximum der zulässigen Alternativen, werden sukzessive die Threads mit den niedrigsten Bewertung entfernt, bis die Anzahl wieder unterhalb des Maximums liegt.

Auf diese Weise wird eine sogenannte Strahlsuche etabliert; dabei handelt es sich um einen heuristischen Suchalgorithmus, der auf der Breitensuche basiert. Auf jeder Ebene, d.h. nach jedem Zyklus, werden alle möglichen Nachfolgezustände berechnet und durch eine heuristische Kostenfunktion – hier das probabilistische Präferenzmaß – bewertet. Dann wird nur eine bestimmte Anzahl von Nachfolgezuständen weiterverwendet – die hier dem Maximum der Threads entspricht (engl.: *beam width*) – die übrigen werden verworfen. Die Strahlsuche kann weder Vollständigkeit (d.h., dass eine Lösung gefunden wird) noch Optimalität (d.h., dass die beste Lösung gefunden wird) garantieren, denn die einzige oder beste Lösung könnte vor ihrer Vervollständigung verworfen werden.

5.3.2.4 Das Interpretationsergebnis

Der Interpretationsprozess ist beendet, wenn keine weiteren primitiven Evidenzen mehr von der Mittelschicht geliefert werden. Das Interpretationsergebnis setzt sich aus einer Teilmenge der vollständig instanziierten Teilmodelle zusammen. Es ist in SCENIOR konfigurierbar (`MAIN_HYPOTHESIS`, s. Anlage B.1), welches Konzept das Wurzelkonzept der kompositionellen Hierarchie bildet. Ausgehend von diesem Wurzelkonzept werden sukzessive alle Teilmodelle generiert – auf diese Weise können auch Teile der Ontologie getestet werden. Das Teilmodell, welches dieses Wurzelkonzept enthält, bildet das Ober- oder Dachmodell. Jedes vollständig instanziierte Obermodell repräsentiert ein mögliches Interpretationsergebnis. Anhand der eindeutigen IDs höherer Evidenzen können sämtliche kontextfreien und vollständig instanziierte Teilmodelle identifiziert werden, die zu einem vollständigen Obermodell gehören. Zusammen bilden sie eine Interpretationsalternative. In der Regel werden aufgrund der Mehrdeutigkeit mehrere Alternativen erkannt, d.h., es gibt mehrere vollständig instanziierte Obermodelle. Dann wird dasjenige mit der höchsten probabilistischen Bewertung als Interpretationsergebnis angesehen. Es ist jedoch nicht gewährleistet, dass es der Realität entspricht; es erfüllt lediglich die probabilistischen Modelle am besten. Weiterhin ist nicht gewährleistet, dass überhaupt eine vollständige Interpretation gefunden wird oder diejenige, die hinsichtlich der probabilistischen Modelle optimal ist, denn sie könnte verworfen worden sein. Falls keine vollständige Interpretation gefunden

wurde, können vollständig instanziierte Teilmodelle als Teilergebnis betrachtet werden.

5.3.2.5 Komplexität

Entscheidend für die Komplexität des Interpretationsprozesses ist die Anzahl der generierten Threads. Wenn von nur einem initialen Modell ausgegangen wird und dem ungünstigsten Fall, dass alle Evidenzkombinationen die zeitlichen (und evtl. andere) Constraints erfüllen, dann berechnet sich die theoretische Anzahl der Threads – d.h. mit unbeschränkter Strahlbreite – folgendermaßen: sei N_i die Anzahl der Blattknoten im Hypothesengraph eines Konzepts E_i und n_i die Anzahl der Evidenzen $\{e_1^i,...,e_{t_i}^i\}$ in den Eingabedaten für dieses Konzept. Weiterhin sei K die Menge der unterschiedlichen Konzepte, die als Blattknoten im Hypothesengraph auftreten. Dann ist die Anzahl der Blattknoten gegeben durch:

$$M_{Blatt} = \sum_{i=1}^{K} N_i \,, \tag{5-35}$$

und die Anzahl der gelieferten Evidenzen für diese Konzepte ist

$$M_{Evi} = \sum_{i=1}^{K} n_i \,. \tag{5-36}$$

Zunächst wird der Fall für *ein* Konzept E_i betrachtet und angenommen, dass genügend Evidenzen vorhanden sind, also $n_i \geq N_i$ gilt. Eine mögliche Zuordnung kann als injektive Abbildung der Blattknoten in die Menge der Evidenzen betrachtet werden. Daher berechnet sich die Anzahl Z_i möglicher vollständiger Instanziierungen, d.h., alle auftretenden Konzepte E_i im Hypothesengraph sind instanziiert, durch:

$$Z_{i\_voll} = \frac{n_i!}{(n_i - N_i)!} \,. \tag{5-37}$$

Dies berücksichtigt jedoch nicht die Menge der Teilinstanziierungen für das Konzept E_i. Die Anzahl möglicher Zuordnungen aller $m-elementigen$ Teilmengen aus der Menge der Blattknoten für ein Konzept zur Menge der entsprechenden Evidenzen ist gegeben durch:

$$Z_{i\_m} = \binom{N_i}{m} \cdot \frac{n_i!}{(n_i - m)!} . \tag{5-38}$$

Daher berechnet sich die Menge aller Teil- und Vollinstanziierungen eines Konzepts E_i durch:

$$Z_{i\_ges} = \sum_{m=0}^{N_i} \binom{N_i}{m} \cdot \frac{n_i!}{(n_i - m)!} . \tag{5-39}$$

Betrachtet man nun alle Blattknoten, dann ergibt sich die Menge aller möglichen Teil- und Vollinstanziierungen des Hypothesengraphs durch Multiplikation über alle Konzepte:

$$Z_{ges} = \prod_{k=1}^{K} \sum_{m=0}^{N_k} \binom{N_k}{m} \cdot \frac{n_k!}{(n_k - m)!} . \tag{5-40}$$

Für eine Abschätzung dieses Terms betrachten wir den ungünstigsten Fall $K = 1$. Dann entspricht N der Menge der Blattknoten und der Ausdruck vereinfacht sich zu:

$$Z_{un} = \sum_{m=0}^{N} \binom{N}{m} \cdot \frac{n!}{(n - m)!} . \tag{5-41}$$

Eine grobe Abschätzung kann gegeben werden durch:

$$Z_{un} = \sum_{m=0}^{N} \binom{N}{m} \cdot \frac{n!}{(n - m)!} < \sum_{m=0}^{N} \binom{N}{m} \cdot n! < n! \cdot \sum_{m=0}^{N} \binom{N}{m} < n! \cdot 2^N , \tag{5-42}$$

d.h., die Anzahl der Threads ist demnach von der Größenordnung $O(n! \cdot 2^N)$. Diese Abschätzung ist jedoch für reale Anwendungen sehr unrealistisch, da sie weder zeitliche Constraints in Betracht zieht, noch unterschiedliche Konzepte berücksichtigt.

Wird angenommen, dass es keine Mehrdeutigkeiten gibt, d.h. keine mehrfach auftretenden Konzepte, also $N_i = 1, \forall i \in \{1, \ldots, K\}$ und $M_{Blatt} = K = N$, dann gilt:

$$Z_{bes\_1} = \prod_{k=1}^{N} \left[\binom{1}{0} \cdot 1 + \binom{1}{1} \cdot n_k \right] = \prod_{k=1}^{N} \left[1 + n_k \right] \tag{5-43}$$

Wird weiterhin davon ausgegangen, dass es für jedes Konzept nur genau eine passende Evidenz gibt, d.h. $n_i = 1, \forall i \in \{1,...,N\}$, dann vereinfacht sich der Ausdruck zu:

$$Z_{bes\_2} = \prod_{k=1}^{N} 2 = 2^N \ , \tag{5-44}$$

d.h., dann entspricht die Anzahl der Threads der Mächtigkeit der Potenzmenge der Blattknoten. Doch auch diese Abschätzung berücksichtigt keine zeitlichen (oder evtl. andere) Constraints, die in der Regel die Anzahl der Threads während des Interpretationsvorgangs deutlich reduzieren. Das zeigen auch die Experimente in Kapitel 7.

5.3.2.6 Determinismus

Der Interpretationsprozess von SCENIOR ist deterministisch. Wie bereits in Abschnitt 5.2.4.2 dargestellt, sind regelbasierte Systeme jedoch nicht grundsätzlich deterministisch. Wenn es Situationen gibt, in denen gleichzeitig mehrere Regeln $r_1 : c_1 \rightarrow h_1$ und $r_2 : c_2 \rightarrow h_2$ aktiviert werden, deren Konklusionen verschieden sind, dann kann dies zu nichtdeterministischem Verhalten führen. Es hängt zusätzlich vom Mechanismus der Aktivierungen der Regeln und von der Konfliktlösungsstrategie der Regelmaschine ab, ob das Verhalten deterministisch ist oder nicht.

In SCENIOR wird nur jeweils eine Evidenz pro Zyklus in die Arbeitsspeicher der aktiven Threads gespeist, daher kann es in einem Thread nicht zu einer mehrfachen Aktivierung von Evidenz-Zuordnungsregeln durch die Situation kommen, dass *mehrere* Evidenzen im Hypothesengraph gleichzeitig zugeordnet werden können. Dies kann nur dadurch geschehen, dass *eine* Evidenz mehrfach im Hypothesengraph zugewiesen werden kann. Durch den Mechanismus der Aufspaltung in parallele Threads, in denen das Feuern jeder möglichen Aktivierung erzwungen wird, bleibt der Interpretationsprozess deterministisch. Nur die Position in der Liste der aktiven Threads könnte variieren, abhängig von der Reihenfolge der Aktivierungen. Wenn das Maximum der zulässigen Threads überschritten ist, wird diese Liste anhand des Präferenzmaßes sortiert. In dem unwahrscheinlichen Fall, dass mehrere der niedrig bewerteten Threads exakt das gleiche Präferenzmaß besitzen und nur einige davon verworfen werden, bevor das Maximum der zur Verfügung stehenden Threads wieder unterschritten wird, könnte dies zu einem nichtdeterministischen Verhalten des Systems führen. Um dem vorzubeugen, ist in SCENIOR eine einfache Konfliktstrategie implementiert, welche die Aktivierungen nach dem Zeitpunkt des Einfügens der betroffenen Fakten in den Arbeitsspei-

cher sortiert. Dies betrifft die Hypothesenelemente der Evidenzzuweisung, und deren Einspeisung in die jeweiligen Arbeitsspeicher während der Initialisierungsphase ist deterministisch.

Bei Aggregat-Instanziierungsregeln kann es zu mehrfachen Aktivierungen kommen, wenn ein Konzept, welches von mehreren Aggregaten genutzt wird, instanziiert wird und dies das letzte noch fehlende Teil für eine Instanziierung dieser Aggregate darstellt. Die beschriebene Konfliktstrategie verhindert auch hier eine zufällige Ausführung der Instanziierungsregeln. Für diesen Fall wäre sie jedoch nicht notwendig, da das Ergebnis – die Instanziierung der Aggregate – von der Reihenfolge unabhängig wäre. Ähnliches gilt für Aggregat-Entfaltungsregeln: Wird ein Aggregat halluziniert, werden die Aggregat-Entfaltungsregeln für alle Teile gleichzeitig aktiviert, doch das Ergebnis ist von der Reihenfolge der Ausführung unabhängig.

In Abschnitt 5.2.4.1 wurde beschrieben, dass die Spezialisierungsregeln so konstruiert sind, dass sie eine Evidenz nur schrittweise spezialisieren, d.h., hier kann es nicht zu mehrfachen Aktivierungen kommen, und damit ist der Determinismus der Spezialisierungsregeln ebenfalls gewährleistet.

5.4 Probabilistisches Präferenzmodell

Die intrinsische Mehrdeutigkeit der Szeneninterpretation, in Verbindung mit den Anforderungen an die Echtzeitfähigkeit und den in Abschnitt 5.3.2.5 dargestellten Betrachtungen zur Komplexität, verdeutlichen die Notwendigkeit eines Präferenzmaßes zur Bewertung von alternativen Interpretationsmöglichkeiten. Die Experimente aus der Domäne der Flughafenvorfeldaktivitäten bestätigen dies, wie in Kapitel 7 gezeigt werden wird.

Im Allgemeinen kann probabilistische Szeneninterpretation als evidenzbasiertes Schließen mit einer gemeinsamen Wahrscheinlichkeitsverteilung (GWV) modelliert werden (s. [79]). Ein probabilistisches Modell kann ausgedrückt werden durch:

$$P_{scene} = q_m \cdot P^{(m)} \cdot (\underline{X}_1^{(m)}, \ldots, \underline{X}_{N^{(m)}}^{(m)}, \underline{Y}_1^{(m)}, \ldots, \underline{Y}_{K^{(m)}}^{(m)}) \cdot P_{clutter} \,. \tag{5-45}$$

Angenommen, es gibt M konkurrierende Modelle mit Vorerwartung q_m, $m \in \{1, \ldots, M\}$. Jedes Modell ist beschrieben durch eine GWV von versteckten Va-

riablen $X=[X_1...X_N]$, welche durch Inferenzen berechnet werden und Variablen $Y=[Y_1...Y_K]$, deren Werte durch primitive oder höhere Evidenzen (bei Teilmodellen, deren Blätter zusammengesetzte Konzepte sind) geliefert werden. Die Indizes deuten unterschiedliche Aggregate an, jedes beschrieben durch einen Vektor von Zufallsvariablen. $P_{clutter}$ ist eine zusammengefasste Verteilung für Evidenzen, die nicht in das Modell passen.

Die M konkurrierenden Modelle stellen die alternativen Interpretationen eines Teilmodells dar, die durch unterschiedliche Evidenzzuweisungen entstanden sind. Ein Präferenzmaß eines Modells n, bei dem die Evidenzen $\underline{e}_n^+$ zugewiesen wurden, kann definiert werden als die Wahrscheinlichkeit, dass das Modell $\underline{e}_n^+$ als Teil der Aktivität generiert hat, die durch das Teilmodell repräsentiert wird, und $\underline{e}_n^-$ als Clutter:

$$R_n = q_n \cdot P^{(n)}(\underline{e}_n^+) \cdot P_{clutter}(\underline{e}_n^-). \tag{5-46}$$

Die Erfahrungen aus früheren Projekten (u.a. *eTRIMS*[7]) im Arbeitsbereich Kognitive Systeme[8] der Universität Hamburg, die sich mit Szeneninterpretation befasst haben, führten zu den von Neumann entwickelten *Bayes'sche Kompositionelle Hierarchien* (BCHs) [79], die speziell an die Anforderungen der Szeneninterpretation angepasst sind. Sie ermöglichen eine sehr effiziente Verarbeitung und sind daher besonders für die echtzeitfähige Szeneninterpretation geeignet; sie werden deshalb in SCENIOR verwendet.

Eine BCH ist ein probabilistisches Modell einer kompositionellen Hierarchie. Ihre grundlegende Struktur wurde bereits in Abschnitt 2.4.3 dargestellt: jedes Aggregat $A^{(i)}$ wird beschrieben durch eine gemeinsame Wahrscheinlichkeitsverteilung (s. [79], Abbildung 7):

$$P(A^{(i)}, B_1^{(i)}, ..., B_{k_i}^{(i)}, C^{(i)}). \tag{5-47}$$

[7] EC Grant IST 027113, Projekt eTRIMS.
[8] http://kogs-www.informatik.uni-hamburg.de/

In der Domäne der Flughafenvorfeldaktivitäten beispielsweise besteht das höchste Aggregat `Turnaround` aus einem Kopf $A^{(1)}$, welcher eine externe Beschreibung von `Turnaround` in Form einer Zeitdauer bereitstellt; sie abstrahiert von den Details der Teile des Aggregats. Des Weiteren gibt es eine interne Beschreibung, welche die zeitliche Struktur der Teile des Aggregats beschreibt: für `Turnaround` sind dies `Arrival` $B_1^{(1)}$, `Services` $B_2^{(1)}$ und `Departure` $B_3^{(1)}$, die wiederum selbst als Aggregate beschrieben sind. Die zeitlichen Constraints zwischen den Teilen werden durch $C^{(1)}$ ausgedrückt: in der BCH werden multivariate Normalverteilungen verwendet, um die zeitlichen Strukturen der Aggregate zu modellieren. Sie können in kompakter Form durch Mittelwerte und Kovarianzmatrizen beschrieben werden. In Abbildung 32 sind jeweils der Mittelwert und die Standardabweichung des Aggregats `Turnaround` und seiner Teile und des Offsets zwischen `Arrival` und `Services` angegeben. Beispielsweise dauert eine vollständige Abfertigung im Mittel 78 Minuten, die Standardabweichung beträgt 26 Minuten. Die Werte wurden aus der Statistik von 52 annotierten Abfertigungen berechnet. Die Normalverteilungen werden im Intervall von $[-2\pm\ldots+2\pm]$ benutzt. Die Verwendung multivariater Normalverteilungen in der BCH ermöglichen sehr effiziente Propagationen neuer Evidenz und die Berechung des Präferenzmaßes (für nähere Details, siehe [33], [79], [81], [83]).

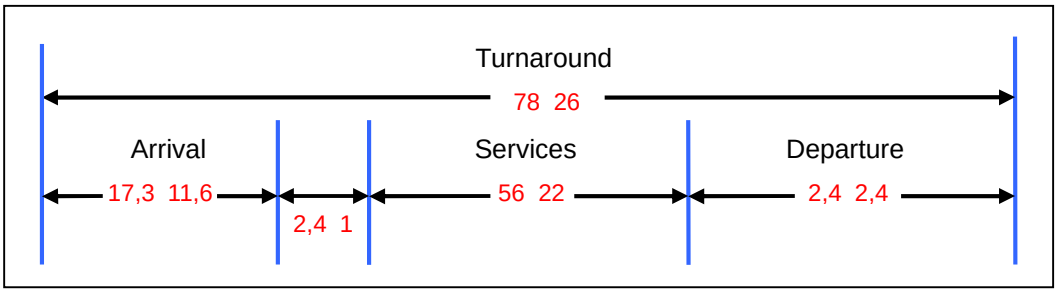


Abbildung 32: Mittelwert (erster Wert) und Standardabweichung (zweiter Wert) für die zeitliche Struktur von `Turnaround` in Minuten.

Das Zusammenspiel zwischen dem strengen zeitlichen Constraint-Netz (ZCN) und der probabilistischen BCH in SCENIOR ist u.a. durch die Entwicklungshistorie zu erklären, denn zu Beginn arbeitete SCENIOR nur mit den in der Ontologie modellierten strengen zeitlichen Constraints, d.h. den ZCNs (s. [32]). Doch die ersten Experimente in der Domäne der Flughafenvorfeldaktivitäten zeigten schnell, dass ein probabilistischen Präferenzmaß unentbehrlich ist. Somit wurde die separat entwickelte BCH in SCENIOR integriert. Es stellte sich heraus, dass das Zusammenwirken beider System Vorteile hat. Beispielsweise können die Normalverteilungen durch die strengen Constraints im Bereich $[-2\pm\ldots+2\pm]$ abgeschnitten werden, aber auch unsymmetrische Modellierungen sind möglich,

die der Realität z.T. besser entsprechen. Abbildung 33 zeigt beispielsweise die Normalverteilung für `Arrival`, mit dem Mittelwert $\mu = 17{,}3$ und der Standardabweichung $\sigma = 11{,}6$. Sie spiegelt die statistische Dauer von `Arrival` wieder. Es gibt jedoch Ausreißer mit einer Dauer von bis zu 60 Minuten. Um diese nicht von vornherein auszuschließen, können die zeitlichen Constraints des ZCN relativ großzügig formuliert werden, z.B. könnten die Minimum- und Maximumwerte der annotierten Abfertigungen verwendet werden, in diesem Fall eine und 60 Minuten. Das führt zu der in Abbildung 33 dargestellten „unsymmetrischen" Normalverteilung.

Grundsätzlich wäre es möglich, das probabilistische System so zu modifizieren, dass es die Aufgaben des ZCN mit übernehmen könnte, in dem die Wahrscheinlichkeiten außerhalb eines bestimmten Bereiches Null wären, doch die Kombination beider Systeme erschien aus entwicklungstechnischer Sicht eine zweckmäßige Lösung zu sein. Weiterhin ist anzumerken, dass auch die Verwendung anderer Verteilungen als die Gaußverteilung grundsätzlich möglich wären, aber zu weitaus komplexeren Implementierungen führen würden.

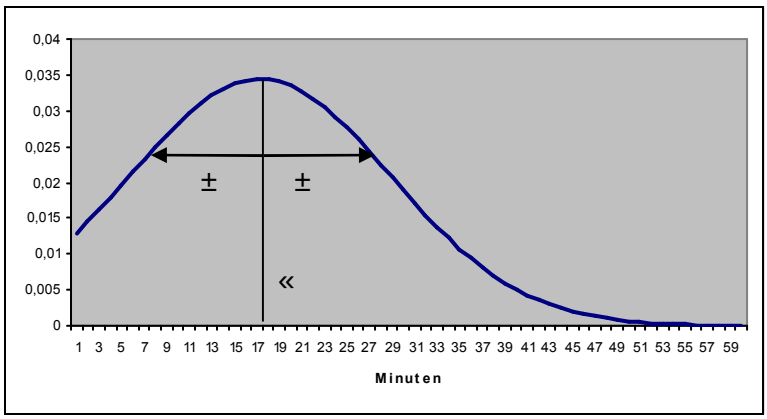


Abbildung 33: Unsymmetrische Normalverteilung für `Arrival`, modelliert durch Kombination mit strengen Constraints des ZCN.

6 Implementierung

In diesem Kapitel wird die Implementierung des Szeneninterpretationssystems SCENIOR vorgestellt. Dazu wird in Abschnitt 6.1 zunächst auf die Kernkomponenten des Systems eingegangen. Dann wird in Abschnitt 6.2 die Benutzungsoberfläche von SCENIOR beschrieben. In Abschnitt 6.3 werden Konfigurierungsmöglichkeiten behandelt und in Abschnitt 6.4 schließlich Erweiterungsmöglichkeiten diskutiert.

6.1 Kernkomponenten

SCENIOR ist ein eigenständiges, modular aufgebautes Szeneninterpretationssystem, welches im Rahmen des EU-Projekts Co-Friend, des DFG-Projekts *PRAESINT*[9] und dieser Arbeit entwickelt wurde. Es basiert auf der Programmiersprache Java (JDK Version 6) und der regelbasierten Sprache Jess (Version 7.1p2). Darüber hinaus verwendet es eine Reihe von externen Programmbibliotheken, die in Abschnitt 6.1.2.2 betrachtet werden. Diese können innerhalb einer Entwicklungsumgebung wie *Eclipse*[10] eingebunden werden oder zusammen mit dem SCENIOR-Quellcode zu einer ausführbaren *.jar-Datei kompiliert werden, die dann innerhalb einer Java-Laufzeitumgebung eigenständig lauffähig ist.

[9] DFG-Projekt Ne 278/9-1, PRAESINT.
[10] http://www.eclipse.org/

6.1.1 Module

Die Klassen von SCENIOR sind in Java-Packages strukturiert, die als Module betrachtet werden können. Eine Übersicht der Module und ihrer hierarchischen Struktur ist in Abbildung 34 dargestellt. Die wichtigsten Module und Klassen werden im nächsten Abschnitt anhand von Klassendiagrammen erläutert.

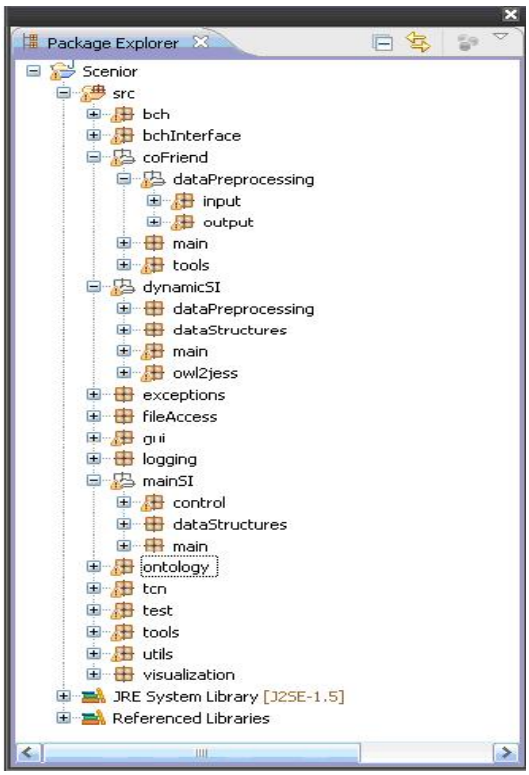


Abbildung 34: Modulstruktur von SCENIOR.

6.1.2 Klassendiagramme

Die in diesem Abschnitt dargestellten Klassendiagramme wurden mit dem Softwaretool *Enterprise Architect*[11] (Version 7.1) erstellt. Sie entsprechen der UML-2-Notation (Version 2.1). In den Diagrammen werden jeweils nur die wichtigsten Klassen dargestellt; sie bilden eine Teilmenge aller entwickelten Klassen. Alle

[11] http://www.sparxsystems.com/products/ea/index.html

unbeschrifteten Kompositionen und Aggregationen besitzen eine Multiplizität von eins.

6.1.2.1 Die Hauptklassen von SCENIOR

Ein Diagramm der Hauptklassen von SCENIOR ist in Abbildung 35 dargestellt. Die Klassen werden im Folgenden nach ihrer Modulzugehörigkeit beschrieben.

Das Modul `mainSI`

In diesem Modul befinden sich Klassen, die allgemeine Funktionen für das Szeneninterpretationssystem zur Verfügung stellen. Die Hauptklasse im Untermodul `mainSI.main` ist die abstrakte Klasse `SIMain` (`SI` ≙ *scene interpretation*). In dieser Klasse werden die Aktionen der graphischen Benutzungsoberfläche mit den Funktionen des Systems verknüpft, wie z.B. `openOntology()` zum Öffnen der Ontologie, `setDataFile()` zum Auswählen einer Eingabedatendatei, `startProcessing()`, `stopProcessing()`, `oneStepProcessing()` zum Steuern des Interpretationsprozesses etc. Die Funktion `checkNewTransformationNeeded()` prüft, ob die Ontologie seit der letzen Konvertierung in die Jess Wissensbasis verändert wurde. Ist dies nicht der Fall, werden die Dateien mit der Jess Wissensbasis direkt gelesen, um eine performante Initialisierung zu erreichen.

Die meisten Funktionen dieser Klasse sind abstrakt und werden in der abgeleiteten Klasse `DSMain` (`DS` ≙ *dynmaic scene interpretation*) implementiert. Des Weiteren befinden sich in dieser Klasse Funktionen für das Protokollieren und für die Fehlerbehandlung.

Angelehnt an das Softwareentwurfsmuster der sogenannten *Modell-Präsentation-Steuerung* (engl.: *model view controller*), sind die Daten des Systems strikt von der Repräsentation in der Benutzungsoberfläche getrennt. Die Klasse `SIMain` übernimmt hier die Funktion der Steuerung. Diese Trennung ermöglicht es in einfacher Weise (durch den Konfigurationsschalter `CONSOLE_MODE`), SCENIOR für Experimente auch ganz ohne Benutzungsoberfläche zu verwenden und beliebig viele Datensätze automatisch nacheinander zu verarbeiten. Sämtliche Interaktionen mit der Benutzungsoberfläche werden über die Klasse `MainFrame` gesteuert. Sie bildet als Realisation des sogenannten *Fassaden-Musters* (s. [97]) die Fassade zum Modul `gui`, welches in Abschnitt 6.1.2.2 behandelt wird.

Eine weitere zentrale Klasse des Untermoduls `mainSI.control` ist die Klasse `Control`. Sie ist als `protected` deklariert, um den abgeleiteten Klassen von `SIMain` den vollen Zugriff auf `Control` zu erlauben. Das `Control`-Objekt ver-

waltet die Jess-Regelmaschinen und die parallelen Threads, und stellt Funktionen zu deren Manipulation bereit. Prinzipiell ist die Anzahl der Threads, die verwaltet werden können, nur durch die Systemressourcen beschränkt. Das `Control`-Objekt enthält Listen der aktiven und abgeschlossenen (d.h. vollständig instanzierten) Threads und Funktionen, wie z.B. `initializeEngine()` zum Initialisieren einer Regelmaschine; dabei werden die generierten Templates und Regeln aus Textdateien gelesen und in die Regelmaschine eingespeist. Weitere Funktionen sind `addFact()` und `modifyFact()`, um Fakten zu einer oder mehreren Regelmaschinen hinzuzufügen oder zu modifizieren.

Mit Hilfe des Java-Mechanismus *Event-Listener* werden über die Methode `sendEvent()` die Vorgänge in den Regelmaschinen in Form von *Events* versendet. Durch Implementierung der Funktion `controlEventHappened()` kann ein anderes Objekt entsprechend auf diese Vorgänge reagieren. Sie ist in den Klassen `SIMain, DSMain` und `MainFrame` implementiert, kann aber auch in jeder abgeleiteten Klassen von `DSMain` implementiert werden, um domänenspezifisch auf bestimmte Ereignisse in den Regelmaschinen reagieren zu können.

Von zentraler Bedeutung ist hier auch die Funktion `copyEngine()` zum Kopieren einer Regelmaschine. Diese wurde ursprünglich durch die von der Jess-Bibliothek (s. Abschnitt 6.1.4) zur Verfügung gestellten Funktionen `bsave()` und `bload()` durch De-/Serialisierung implementiert (s. auch [8]). Später wurde das Kopieren unter Verwendung von `createPeer()` realisiert, da dies wesentlich performanter ist, Speicherplatz spart und somit mehr parallele Threads erlaubt. Dabei werden die Regelmaschinen geklont, besitzen jeweils einen separaten Arbeitsspeicher, aber nur eine gemeinsame Regelbasis und ein Rete-Netz (s. [8], [52]). Zu beachten ist bei dem Kopieren, dass nicht nur alle Fakten kopiert werden müssen, sondern auch die mit der Regelmaschine zusammenarbeitenden Constraint-Mechanismen, wie das ZCN und die BCH (s. auch Abschnitt 6.1.2.3), Konfliktstrategien, Event-Listener (die auf die Aktionen der Regelmaschine, wie z.B. das Feuern einer Regel reagieren) und weitere Hilfsobjekte.

Weitere Funktionen der Klasse `Control` sind `killThread()` zum Beenden von Threads und `runEngines()` zum Starten der Regelmaschinen. Wenn der Konfigurationsschalter für Multithreading (`MULTITHREADING`) auf `true` gesetzt ist, werden alle aktiven Regelmaschinen in parallelen Java-Threads gleichzeitig gestartet. Dies bringt auf Mehrprozessorsystemen deutliche Performanzvorteile. Ist Multithreading deaktiviert, werden die Regelmaschinen sequentiell abgearbeitet. Auf das Ergebnis des Interpretationsprozesses hat dies keinen Einfluss.

Eine ebenfalls zentrale Funktion ist `splitEngine()`, die das Aufspalten von Threads und damit den Aufbau des Suchbaums (s. Abschnitt 5.3.2.2) realisiert. Sie analysiert dazu die Liste der Aktivierungen, die durch Evidenz-Zuordnungsregeln

ausgelöst wurden. Anschließend werden mit `copyEngine()` entsprechend viele Klone der Regelmaschine generiert. Durch das Setzen einer speziellen Konfliktstrategie (`ConflictStrategyByName`), die den Namen der Aktivierung beachtet, wird in jedem Klon das Feuern jeweils einer dieser Regeln forciert. Nach dem Feuern dieser Regel wird die Konfliktstrategie umgehend auf die ursprüngliche Strategie zurückgesetzt (dies wird durch einen Event-Listener ermöglicht; es kann nicht gewartet werden, bis die Regelmaschine ihren Durchlauf beendet hat, denn in einem Zyklus können mehrere Regeln feuern).

Die Klasse `Engine` des Untermoduls `mainSI.control` enthält die Klasse `Rete` der Jess-Bibliothek (s. Abschnitt 6.1.4) und bildet damit die Schnittstelle zu Jess. Des Weiteren enthält sie eine Zuordnungstabelle (*Map*) der Klasse `Constraint-Control` als Schnittstelle zu den Constraint-Mechanismen, die in Abschnitt 6.1.2.3 behandelt werden.

Die Klassen `JessFileGenerator`, `JessFactConverter`, `Hypothesizer` und `DataThread` des Untermoduls `mainSI.main` sind rein abstrakte Klassen und werden im Modul `DSMain` erläutert. Die Klasse `Config` dient zur Konfiguration von SCENIOR (s. Abschnitt 6.3).

Das Modul `ontology`

Die zentralen Klassen `OntologyAccessor` und `RuleAccessor` dieses Moduls realisieren Funktionen zum Verarbeiten von OWL Ontologien und SWRL-Regeln. Sie verwenden dafür die externe Bibliothek `owlapi-bin.jar` (s. Abschnitt 6.1.4). Die Klasse `OntologyAccessor` ist als `protected` deklariert, um den abgeleiteten Klassen von `SIMain` vollen Zugriff auf diese Klasse zu ermöglichen.

Das Modul `logging`

Dieses Modul enthält Klassen und Schnittstellen für Protokollierungsfunktionalitäten, z.B. zur Protokollierung von Funktionsaufrufen (engl.: *function logging*) und spezifischen Programmausgaben. Fünf verschiedene Protokollierungsstufen können konfiguriert werden: `OFF`, `FINE`, `FINER`, `FINEST` und `WARNING`. Die Ausgaben der Protokollierung können auf die Konsole, die Benutzungsoberfläche (s. Abschnitt 6.2) oder in eine Textdatei geleitet werden.

Die Module `fileAccess` und `exceptions`

Das Modul `fileAccess` enthält Klassen für Dateizugriffe, und das Modul `exceptions` enthält Klassen für die Fehlerbehandlung. Sämtliche Ausnahmezustände (engl.: *exceptions*) werden an die Klasse `SIMain` hochgereicht, um sie dort

zentral zu verarbeiten, beispielweise um sie in der Benutzungsoberfläche anzuzeigen.

Das Modul `dynamicSI`

In diesem Modul befinden sich Klassen, die Funktionalitäten der dynamischen Szeneninterpretation zur Verfügung stellen, d.h., die zur Verarbeitung und Interpretation von Eingabedaten aus Videosequenzen dienen. Insbesondere kennt diese Klasse die Klassen des ZCN und der BCH. Die Hauptklasse im Untermodul `dynamicSI.main` ist die Klasse `DSMain`. In dieser Klasse werden alle abstrakten Funktionen der Superklasse `SIMain` implementiert, z.B. `startProcessing()`, `stopProcessing()` und `oneStepProcessing()` zum Steuern des Interpretationsprozesses. Sie selbst enthält einige wenige abstrakte Funktionen, die domänenspezifisch sind und deshalb in der abgeleiteten Klasse `CFMain` (`CF` $\hat{=}$ *Co-Friend*) implementiert sind, d.h., die Klasse selbst ist abstrakt.

Des Weiteren koordiniert das `DSMain`-Objekt die Objekte `Control` (siehe oben), `DSJessFileGenerator, DSHypothesizer, DSFileAccess` und `DSDataProcessThread` (die im Weiteren erläutert werden). Das `DSMain`-Objekt bildet die zentrale Steuereinheit von SCENIOR.

Das `DSMain`-Objekt initialisiert darüber hinaus die Schnittstellen `TCNControl` und `BCHControl` und fügt diese dem `ConstraintControlMap`-Objekt hinzu, welches über das `Control`-Objekt an das zu initialisierende `Engine`-Objekt weitergereicht wird. Für Erweiterungen der bestehenden Domäne oder der Realisierung anderen Domänen können dem `ConstraintControlMap`-Objekt weitere Constraint-Mechanismen hinzugefügt werden (siehe Abschnitt 6.1.2.3). Dies geschieht durch Reimplementierung der Funktion `addToConstraintControl()` in der abgeleiteten Klasse von `DSMain`. Weiterhin steuert das `DSMain`-Objekt Mechanismen mit dem ZCN, die nicht durch das Feuern von Regeln ausgelöst werden, beispielsweise die Überprüfung der Erfüllbarkeit des ZCN (s. Abschnitt 5.3.2.3), die zu konfigurierbaren Zeitintervallen durchgeführt wird. Damit zusammenhängend steuert es auch das Halluzinieren von Ereignissen und Zuständen. Eine weitere wichtige Funktion dieser Klasse ist `createHighEvidence()`, zum Generieren und Weitergeben höherer Evidenz.

Weitere Klassen im Untermodul `dynamicSI.main` sind `DSJessFactConverter()` mit Funktionalitäten zum Konvertieren von Eingabedaten, die von der Mittelschicht geliefert werden, in Jess-Fakten (s. Abschnitt 5.3.2.1), `DSFileAccess` für Dateizugriffe, `DSConflictStrategy`, welche die in Abschnitt 5.3.2.6 beschriebene Konfliktstrategie implementiert, und `DSRating`, welche als Schnittstelle zur probabilistischen Bewertung der BCH (s. Abschnitt 5.4) dient.

Im Untermodul dynamicSI.owl2jess befindet sich die Klasse DSHypothesizer, welche Methoden zum Generieren von Hypothesengraphen bereitstellt. Das DSHypothesizer-Objekt hält eine Map aller Hypothesengraphen bereit, die durch die DSHypothesisGraph-Objekte repräsentiert werden. Die Klasse DSJessFileGenerator enthält Funktionalitäten zum Generieren der konzeptuellen Jess Wissensbasis (s. Abschnitt 5.2.2), beispielsweise die Funktionen generateTemplates(), generateSpecialisationRules(), generateMatchingRules() und generateAggregateRules(). Die generierten Templates und Regeln werden in From von Textdateien abgespeichert und vom Control-Objekt eingelesen.

Im Untermodul dynamicSI.dataProcessing befindet sich die Klasse DataProcessThread, welche Methoden zum Einlesen von Eingabedaten bereitstellt. Dies ist als separater Thread realisiert, um die Bedienbarkeit der Benutzungsoberfläche, welche im Haupt-Thread läuft, während des Interpretationsprozesses zu gewährleisten. Sie enthält die rein abstrakte Klasse DSInObjectsConverter. Die davon abgeleitete Klasse CFInObjectsConverter implementiert Methoden, die domänenspezifische Eingabedaten zunächst in ein internes SCENIOR-Format konvertieren. Die Daten können für das Projekt Co-Friend auf drei verschiedene Arten eingelesen werden (siehe Modul coFriend). Nach der Konvertierung in das interne Format, werden die Daten vom DSJessFactConverter-Objekt in Jess-Fakten konvertiert und vom DSDataProcessThread-Objekt an das DSMain-Objekt zur Weiterverarbeitung übergeben.

Das Modul coFriend

Das Modul coFriend enthält domänenspezifische Klassen für das Projekt Co-Friend. Im Untermodul coFriend.main befindet sich die Klasse CFLauncher zum Starten von SCENIOR. Das Untermodul coFriend.dataProcessing.input enthält neben CFInObjectsConverter u.a. vier Klassen zur Datenverarbeitung, welche Eingabedaten lesen und sie in die Eingabe-Queue einspeisen (s. Abschnitt 5.3.2.1), die anschließend vom DSDataProcessThread-Objekt gelesen werden:

- SQLDataPushThread liest Eingabedaten aus einer PostgreSQL-Datenbank.
- XMLDataPushThread liest Eingabedaten aus einer XML-Datei.
- ListDataPushThread liest Eingabedaten aus einer Datei, welches ein eigenständiges, einfaches Format hat.
- MWClientThread liest Eingabedaten, die im Rahmenwerk des Co-Friend-Projekts über Corba geliefert werden.

Das Untermodul `coFriend.dataProcessing.output` enthält die Klassen `CFOutObjectsConverter`, welche Methoden zum Erzeugen von Ausgabeobjekten für Corba und eines eigenständigen Formats für eine Dateiausgabe bereitstellt, und `MWServerThread` mit Methoden zur Übergabe der Objekte an Corba.

Die Module `test, utils, tools, visualization`

Das Modul `test` enthält Klassen für einzelne Modultests für SCENIOR, weiterhin die Klasse `AllTests` zum Ausführen aller Modultests. Die Module `utils` und `tools` enthalten Hilfsprogramm und Werkzeuge, hauptsächlich für die Durchführung von Experimentreihen. Das Modul `visualization` enthält Klassen zur synchronisierten Visualisierung der Interpretation und eines Videos der Vorfeldaktivitäten. Diese benutzen die externen Bibliotheken `xmlrpc.client-3.1.3.jar` und `xmlrpc-common-3.1.3.jar` (s. Abschnitt 6.1.4).

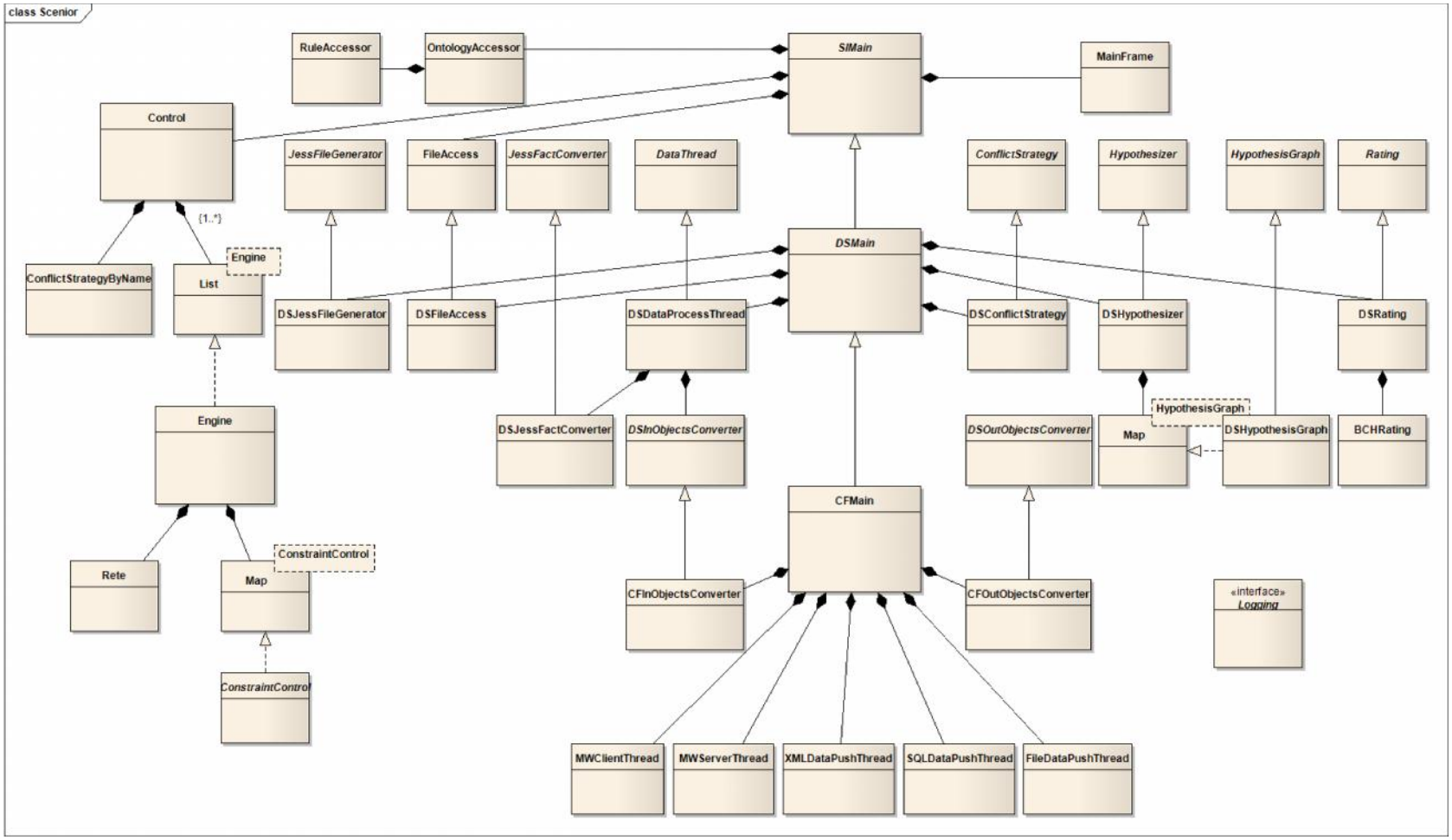


Abbildung 35: Hauptklassendiagramm von SCENIOR.

6.1.2.2 Klassen der graphischen Benutzungsoberfläche

Das Modul `gui` enthält die Klassen der Benutzungsoberfläche. In Abbildung 36 ist das Diagramm der wichtigsten Klassen dargestellt. Die Klasse `MainFrame` bildet die Fassade für dieses Modul. Durch Implementierung der Funktion `controlEventHappened()` werden die Vorgänge in den Regelmaschinen in der Benutzungsoberfläche angezeigt (s. Abschnitt 6.2). Mit dem Java-Mechanismus der *Aktion-Listener* wird auf Aktionen des Benutzers reagiert, welche an die Klasse `SIMain` weitergeleitet werden. Die aufgerufenen Methoden sind in der abgeleiteten Klasse `DSMain` implementiert.

Die Klasse `PrimitivesFrame` dient zur Anzeige der eingelesenen primitiven Evidenzen, die Klasse `PropertiesFrame` stellt Methoden für das Dialogfenster zum Einstellen von Optionen bereit. Die Klasse `SimpleTextFrame` wird für die Anzeige von Protokollierungen verwendet. Die Klasse `MainMenuBar` enthält Methoden für die Menüs und Bedienelemente von SCENIOR.

In SCENIOR wird für jeden Thread des Suchbaums (s. Abschnitt 5.3.2.2) dynamisch jeweils ein Ausgabefenster (repräsentiert durch ein `ThreadPanel`-Objekt) für

- die Ausgabe der Regelmaschine (*Jess-Tab*),
- die textuelle Ausgabe des Interpretationsergebnisses (*Recognition-Tab*),

erzeugt. Optional können noch weitere Ausgabefenster hinzugenommen werden (s. Abschnitt 6.3), die spezielle Ausgaben der Regelmaschinen anzeigen und die Hypothesengraphen visualisieren:

- Ausgabefenster für gefeuerte Regeln (*Rules-fired-Tab*),
- Ausgabefenster für Manipulation von Fakten (*Facts-Tab*),
- Ausgabefenster für Aktivierungen von Regeln (*Activations-Tab*),
- Visualisierung der Hypothesengraphen (*Visualisation-Tab*).

Jess-Tab, Recognition-Tab, Rules-fired-Tab, Facts-Tab und Activations-Tab enthalten `TextFrame`-Objekte, der Visualisation-Tab ein `Hypothesis-VisFrame`-Objekt. Die Klasse `TextFrame` ist direkt, die Klasse `Hypothesis-VisFrame` indirekt von der Klasse `Jess-Frame` abgeleitet. Ein `Jess-Frame`-Objekt enthält eine Map mit allen `ThreadPanel`-Objekten (dabei ist konfigurierbar, ob auch gestorbene Threads angezeigt werden). Das `MainFrame`-Objekt enthält eine

Map der `Jess-Frame`-Objekte (jeweils ein Objekt für die oben aufgeführten Tabs).

Die Klasse `Hypothesis-Vis-Frame` enthält die von der Klasse `Graph` abgeleitete Klasse `HypothesisVisGraph`, die zum Visualisieren der Hypothesengraphen dient. Die Klasse `Graph` benutzt die externe Bibliothek `jgraph.jar` (s. Abschnitt 6.1.4).

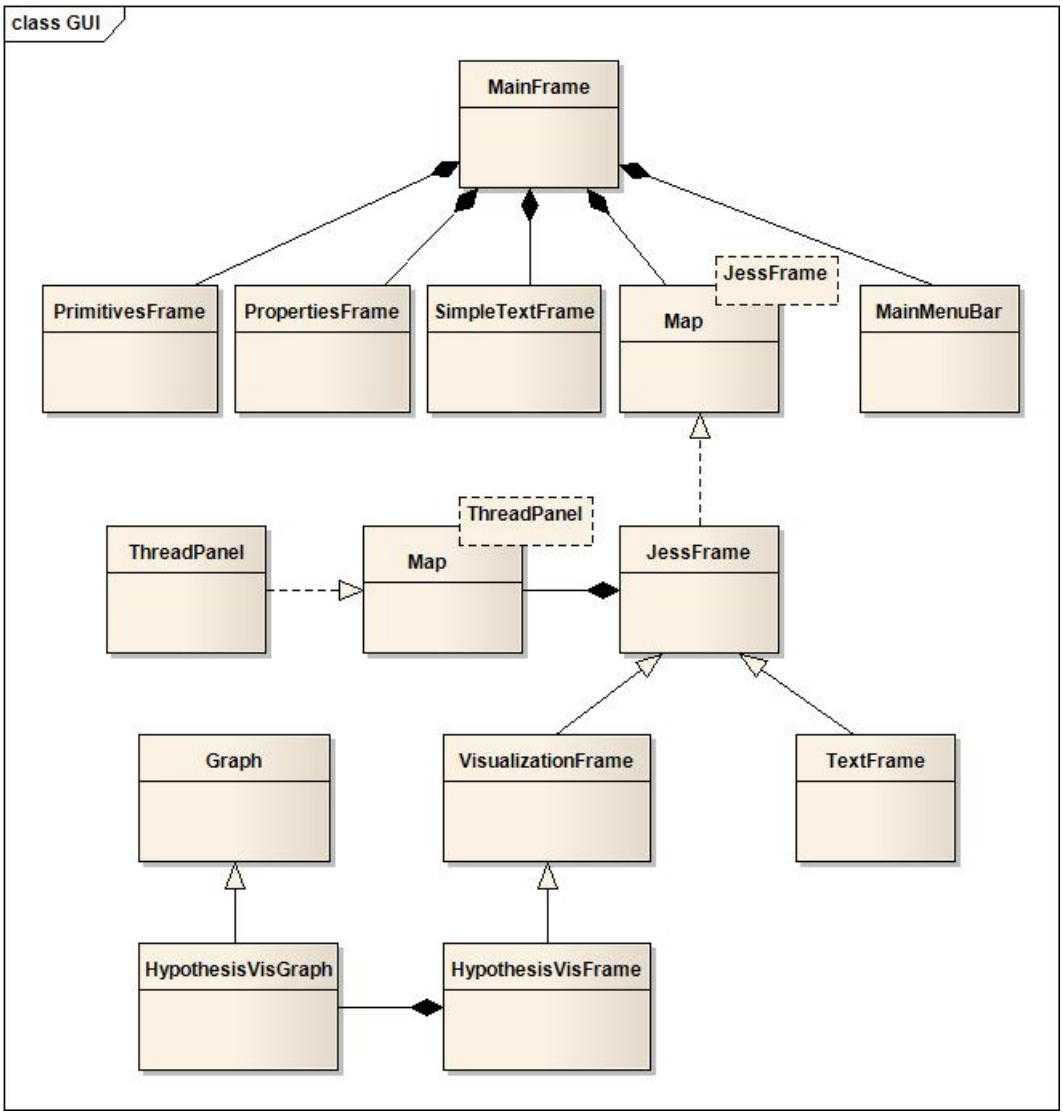


Abbildung 36: Klassendiagramm der graphischen Benutzungsoberfläche.

6.1.2.3 Klassen zur Verarbeitung von Constraints

Die Verarbeitung von Constraints spielt in SCENIOR eine zentrale Rolle. Die beiden Constraint-Mechanismen ZCN und BCH sind aufgrund ihres prozeduralen Charakters in Java implementiert, arbeiten jedoch eng mit den Jess-Regeln zusammen. Dies wird durch den sogenannten *Shadow-Fact-Mechanismus* realisiert (s. [52]). Dadurch können Fakten (sog. *Shadow-Facts*) im Arbeitsspeicher einer Regelmaschine an Java-Objekte gekoppelt werden, um eine effiziente Verbindung beider Welten zu ermöglichen; so können beispielweise Veränderungen von Slot-Werten eines Fakts direkt über den Shadow-Fact Änderungen des zugehörigen Java-Objekts bewirken und umgekehrt. Weiterhin können in Jess-Regeln Java-Methoden aufgerufen werden.

Jede Klasse, die einen Constraint-Mechanismus realisiert, wird von der abstrakten Klasse `ConstraintControl` abgeleitet. Das Klassendiagramm ist in Abbildung 37 dargestellt. Die Objekte dieser Klasse sind durch Implementierung der Schnittstelle `ShadowObject` als Shadow-Facts definiert. Die Klasse `ConstraintControl` erzwingt bei den abgeleiteten Klassen u.a. die Implementation einer `copy()`-Funktion, welche bei dem Klonen von Threads dafür sorgt, dass von den Constraint-Mechanismen – das ZCN und die BCH – tiefe Kopien erstellt werden.

Die Klasse `TCNControl` enthält die Klasse `TCNFactory`, u.a. mit der Methode `recursiveCreateTCN()`, zum Generieren eines ZCN für ein Teilmodell aus den einzelnen ZCN der Aggregate (s. Abbildung 27). Alle ZCNs werden in einer Map vorgehalten. Die Klasse `BCHControl` enthält die Klasse `BCHInterface`, welche die Schnittstelle zur externen Bibliothek `BCH.jar` darstellt (s. Abschnitt 6.1.4).

Das `TCNControl`-Objekt und das `BCHControl`-Objekt werden als Shadow-Facts dem Arbeitsspeicher jeder Regelmaschine hinzugefügt. Die korrespondierenden Java-Objekte befindet sich im `ConstraintControlMap`-Objekt (siehe Untermodul `mainSI.control`). Weitere Constraint-Mechanismen können dem `ConstraintControlMap`-Objekt durch Reimplementierung der Funktion `addToConstraintControl()` in abgeleiteten Klassen von `DSMain` hinzugefügt werden. Natürlich müssten dazu auch die Jess-Regeln – und das bedeutet deren automatische Generierung – angepasst werden. Dies ist bisher nicht sehr komfortabel möglich: Es wird dazu notwendig, in einer abgeleiteten Klasse von `DSJessFileGenerator` die zentralen Methoden, z.B. `generateMatchingRules()`, entsprechend der Interaktionen mit einem neuen Constraint-Mechanismus zu reimplementieren.

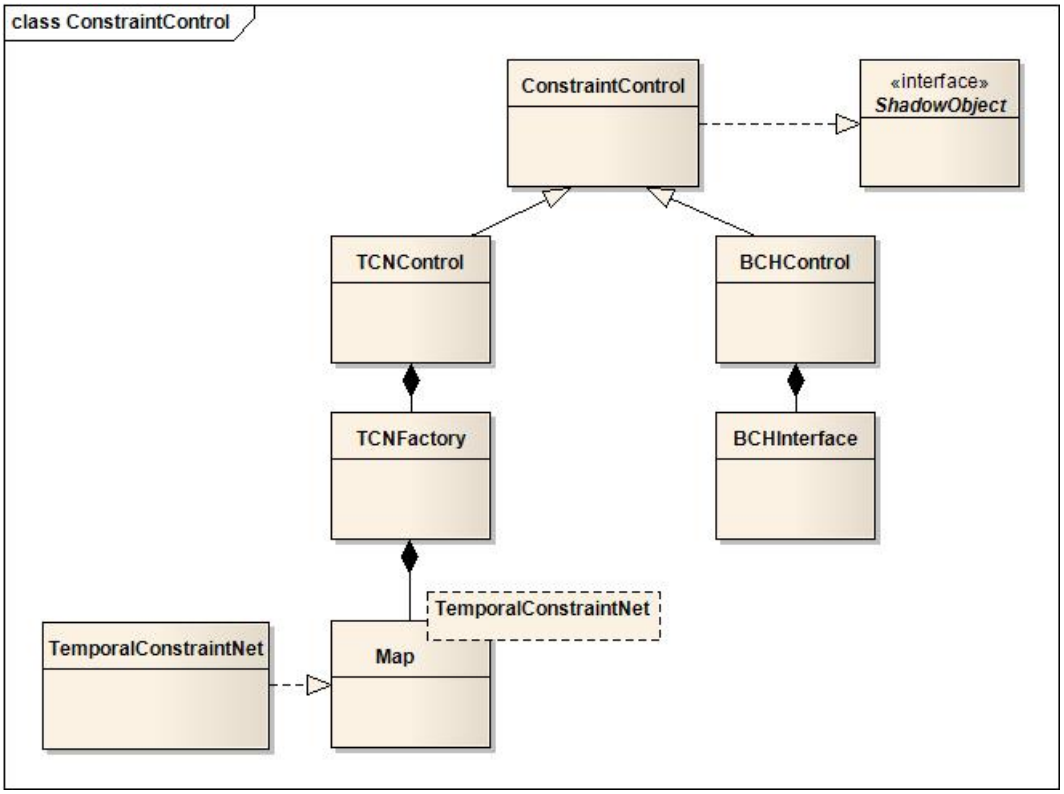


Abbildung 37: Klassendiagramm für Constraint-Mechanismen.

Als Beispiel ist in Regel 6-1 die Interaktion der Evidenz-Zuordnungsregeln mit dem ZCN dargestellt (s. auch Regel 5-20). In Zeile 3 wird durch das Schlüsselwort `OBJECT` eine Referenz auf das Java-Objekt zum Shadow-Fact `TCNControl` bereitgestellt. In Zeile 4 und 5 wird dann die Methode `checkConsistency()` des `TCNControl`-Objekts aufgerufen. Dies geschieht innerhalb einer Testfunktion (einem sog. *test conditional element*, s. [52]). Sie speist die Zeitstempel, mit

- `?h-st` $\hat{=}$ eindeutiger Namen des Zeitpunktobjekts im Hypothesengraph,
- `?tMin-i-st` $\hat{=} t_{i\,min}$,
- `?tMax-i-st` $\hat{=} t_{i\,max}$,

der Evidenz in eine Kopie des ZCN ein und propagiert die Werte (s. Abschnitt 5.2.3). Kommt es dabei zu einer Inkonsistenz des ZCN, wird `false` zurückgegeben, andernfalls `true`. Im Aktionsteil der Regeln werden die Werte in Zeile 8 und 9 endgültig ins ZCN eingespeist und propagiert.

```
(01)  (defrule aggregate-x-ea-rule
(02)       ...
(03)      (TCNControl (OBJECT ?tcn-control))
(04)      (test (call ?tcn-control checkConsistency
(05)                  ?h-st ?tMin-i-st ?tMax-i-st))
(06)       ...
(07)      =>
(08)      (call ?tcn-control update
(09)             ?h-st ?tMin-i-st ?tMax-i-st)
(10)       ...
(11)  )
```
(6-1)

6.1.3 Sequenzdiagramm

Das in Abbildung 38 dargestellte Sequenzdiagramm verdeutlicht die Interaktionen der beteiligten Objekte bei dem Interpretationsprozess von SCENIOR. Aus Gründen der Übersichtlichkeit sind jedoch nur die wichtigsten Objekte und Interaktionen aufgeführt.

Zu Beginn des Interpretationsprozesses werden vom `CFMain`-Objekt die Threads `ListDataPushThread`-Objekt, zum Liefern von Evidenzen (in einem `Frame-Objekt`) und `DSDataProcessThread`-Objekt, zum Verarbeiten der Evidenzen, gestartet. Vom `ListDataPushThread`-Objekt werden `Frame`-Objekte an die `inObjects`-Queue geliefert, die in der Schleife `processing` vom `DSDataProcessThread`-Objekt entnommen werden, bis keine neuen `Frame`-Objekte mehr geliefert werden. Alle nachfolgenden Aktionen befinden sich innerhalb dieser Schleife.

Zunächst wird ein `Frame`-Objekt vom `CFInObjectsConverter`-Objekt in eine Liste von `SceneObject`-Objekten konvertiert, dem internen Format zur Repräsentation von konzeptuellen und physikalischen Objekten. Diese werden nun einzeln in der Schleife `cycle` – die einem Zyklus entspricht (s. Abschnitt 5.3.2.1) – vom `DSJessFactConverter`-Objekt in Jess-Fakten konvertiert und anschließend an das `Control`-Objekt und von dort an die `Engine`-Objekte weitergereicht (das `engine1`-Objekt steht hier stellvertretend für alle aktiven `Engine`-Objekte). und in die Regelmaschinen eingespeist. Dies führt zu Aktivierungen von Regeln, anhand derer über die Funktion `splitEngines()` die `Engine`-Objekte geklont werden, um den Suchbaum aufzubauen.

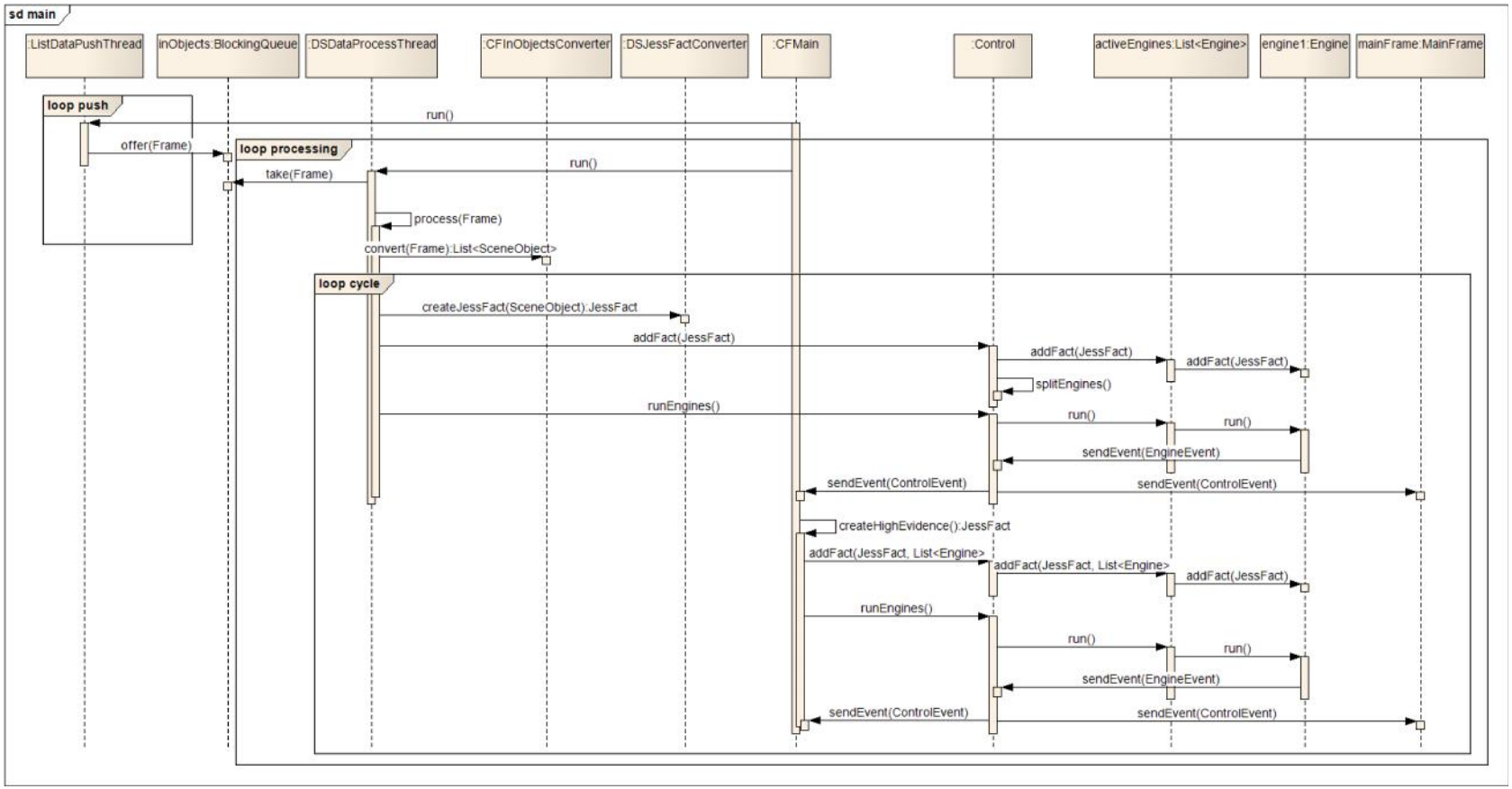


Abbildung 38: Sequenzdiagramm des Interpretationsprozesses von SCENIOR.

Dann wird über `runEngines()` das Signal an das `Control`-Objekt und von dort an die aktiven `Engine`-Objekte zum Starten der Regelmaschinen gegeben. Alle Regelmaschinen informieren das `Control`-Objekt via `sendEvent(EngineEvent)` über ihre Aktivitäten. Von dort werden diese und weitere Nachrichten (beispielsweise, dass der Durchlauf aller Regelmaschinen beendet ist) via `sendEvent(ControlEvent)` an das `DSMain`-Objekt und das `MainFrame`-Objekt geschickt, welches die Nachrichten auswertet und entsprechende Aktionen in der Benutzungsoberfläche auslöst.

Das `DSMain`-Objekt überprüft, ob das Teilmodell eines Threads vollständig instanziiert ist und erzeugt dann mit der Methode `createHighEvidence()` höhere Evidenzen, die daraufhin an die entsprechenden `Engine`-Objekte weitergeleitet werden. Die Verarbeitung dieser höheren Evidenzen verläuft analog zur beschriebenen Verarbeitung primitiver Evidenzen. Sind alle Evidenzen eines `Frame`-Objekts verarbeitet, wird das nächste `Frame`-Objekt aus der `inObjects`-Queue geholt und der Vorgang beginnt von Neuem.

6.1.4 Externe Programmbibliotheken

SCENIOR benötigt eine Reihe von externen Programmbibliotheken, die in Tabelle 5 zusammengefasst sind. Die Funktionalitäten der einzelnen Bibliotheken und ihre Nutzung in SCENIOR werden im Folgenden kurz erläutert:

- Die Bibliothek `BCH.jar` enthält die Funktionalität für die Bayes'schen Kompositionellen Hierachien (BCHs). Sie wurde von Zimmer [109] im Rahmen des DFG-Projekts PRAESINT im Arbeitsbereich Kognitive Systeme der Universität Hamburg entwickelt.

- Die Bibliothek `jess.jar` stellt das Rahmenwerk für die regelbasierte Sprache Jess dar und ist somit ein zentraler Bestandteil von SCENIOR.

- Die Bibliothek `jgraph.jar` (*JGraph*[12]) ist ein Rahmenwerk zum Zeichnen von Graphen und wird zur Darstellung der Hypothesengraphen verwendet.

[12] http://www.jgraph.com/

- Die Bibliothek `junit-4.8.2.jar` (*JUnit*) stellt Funktionalitäten für Modultests bereit.

- Die Bibliothek `owlapi-bin.jar`[13] ist ein Rahmenwerk zur Bearbeitung von OWL Ontologien. Sie wird im Wesentlichen für die Übersetzung der OWL / SWRL Wissensbasis in die Jess Wissensbasis verwendet.

- Die Bibliothek `postgresql-8.4-701.jdbc4.jar` ist ein Rahmenwerk für das Datenbanksystem *PostgreSQL*[14]. Es dient in SCENIOR zum Auslesen von Eingabedaten, die durch den Co-Friend-Projektpartner *AKKA Technologies*[15] u.a. in einer Datenbank bereitgestellt wurden.

- Die Bibliothek `SMForJava.jar` stellt eine Schnittstelle zum Corba-Rahmenwerk bereit, welche als Interprozesskommunikation zwischen den Komponenten der Co-Friend-Plattform (s. Abschnitt 7.1.2) dient. Sie wurde von AKKA Technologies entwickelt.

- Die Bibliotheken `xmlrpc.client-3.1.3.jar` und `xmlrpc-common-3.1.3.jar` stellen ein Rahmenwerk für den *XML-Remote-Procedure-Call*[16], d.h. den Methodenaufruf in verteilten Systemen zu Verfügung. Sie dienen in SCENIOR zur Kommunikation mit einer Visualisierungssoftware, die vom Projektpartner der Universität Leeds[17] entwickelt wurde. Dadurch kann parallel zum Interpretationsverlauf in SCENIOR das synchronisierte Video der Vorfeldaktivitäten gezeigt werden.

[13] http://owlapi.sourceforge.net/index.html
[14] http://www.postgresql.de/
[15] http://www.akka.eu/uk/corporate/world/france.php
[16] http://xmlrpc.scripting.com/spec.html
[17] http://www.comp.leeds.ac.uk/vision/index.html

Name	Beschreibung
BCH.jar	Funktionalitäten für Bayes'sche Kompositionelle Hierachien (BCHs).
jess.jar	Rahmenwerk für die Regelmaschine Jess (Java Expert System Shell). Frei verfügbar für nichtkommerzielle Nutzung.
jgraph.jar	Rahmenwerk JGraph zum Zeichnen von Graphen (open source).
junit-4.8.2.jar	Rahmenwerk JUnit für Modultests (open source).
owlapi-bin.jar	Rahmenwerk zum Bearbeiten von OWL Ontologien (open source).
postgresql-8.4-701.jdbc4.jar	Rahmenwerk für Datenbanksystem PostgreSQL (open source).
SMForJava.jar	Schnittstelle zum Corba-Rahmenwerk.
xmlrpc.client-3.1.3.jar	Rahmenwerk für XML-Remote-Procedure-Call, für den Methodenaufruf in verteilten Systemen.
xmlrpc-common-3.1.3.jar	

Tabelle 5: In SCENIOR verwendete externe Programmbibliotheken.

6.2 Graphische Benutzungsoberfläche

In diesem Abschnitt werden die wesentlichen Elemente der Benutzungsoberfläche von SCENIOR anhand von Screenshots erläutert.

6.2.1 Hauptmenü

Das Hauptmenü besteht aus folgenden Punkten (s. Abbildung 40):

- File, zum Öffnen einer Ontologie, einer Datendatei und zum Beenden des Programms.

- Options, zum Öffnen des Properties-Dialogs (s. Abbildung 39) – hier kann ein Teil der Konfigurationsmöglichkeiten eingestellt werden (siehe auch Abschnitt 6.3). Weiterhin Clear JCKB Directory, zum Löschen des Verzeichnisses der konzeptuellen Jess Wissensbasen.
- Run, mit Funktionen zum Steuern des Interpretationsprozesses (IP):
  - Run, zum Starten des IP.
  - Continue, zum Fortführen des IP nach Pause / Pause at Frame.
  - One Step, zum Verarbeiten der nächsten primitiven Evidenz. Danach wird der IP gestoppt.
  - Pause, zum Anhalten des IP.
  - Pause at Frame, zum Anhalten des IP bei Erreichen eines bestimmten Frames.
  - End, zum Beenden des IP. Durch erneutes Run wird der IP vom Beginn an gestartet.
- About, zum Öffnen des Fensters mit Information zu SCENIOR.

Darunter befinden sich die Anzeigefelder Ontology, für die geladene Ontologie, Data file, für die geladene Datendatei und die Auswahlbox Dataset, zum Selektieren eines Datensatzes, falls die Daten von der PostgreSQL-Datenbank gelesen werden.

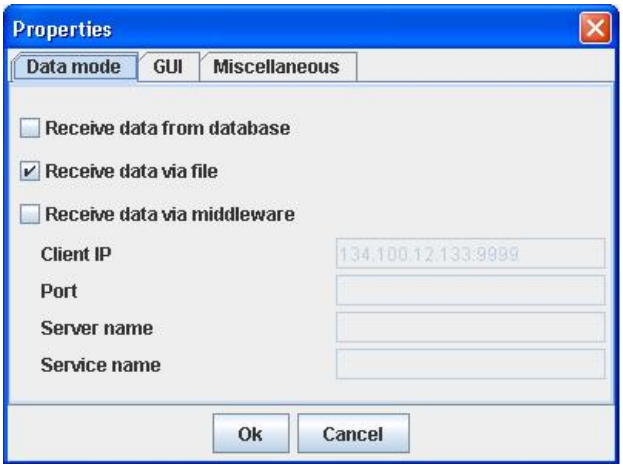


Abbildung 39: Der Properties-Dialog von SCENIOR.

6.2.2 Processing-Tab

Der Processing-Tab dient zum Anzeigen und Steuern des IP und bildet damit das zentrale Element der Benutzeroberfläche. Er enthält die folgenden Elemente (von rechts nach links, von oben nach unten):

- Vier Buttons mit den Funktionen Run, One Step, Pause, End. Wurde Pause aktiviert, kann der IP durch erneutes drücken des Run-Buttons fortgeführt werden.
- Eine Anzeige Processing für den Status des IP: grau bedeutet, dass kein IP läuft, blinkend grün/gelb, dass der IP aktiv ist und rot, dass ein Fehler aufgetreten ist.
- Das Feld Frame, zur Anzeige der ID des Frames, der aktuell verarbeitet wird.
- Das Feld Entries in queue, zur Anzeige der Anzahl der Frames, die noch zu verarbeiten sind.
- Eine Auswahlbox mit den Einträgen model und all. Sie bezieht sich auf die nachfolgenden Anzeigefelder für die Threads: bei model werden die Informationen für die Threads eines Teilmodells (welches in der Auswahlbox Models selektiert ist) angezeigt, bei all gelten die Informationen für alle Threads.
- Anzeigefelder für die Anzahl der Threads: Threads steht für alle Threads, Active für aktive Threads, Finished für abgeschlossene Threads (vollständig instanziiert) und Dead für gestorbene Threads.
- Die Auswahlbox Models, zum Selektieren eines Teilmodells. Alle Anzeigen auf dem Blackboard (s. Abschnitt 6.2.2.1) beziehen sich auf das ausgewählte Teilmodell (all zeigt alle Teilmodelle auf dem Blackboard an).
- Das Anzeigefeld Last primitive, zum Anzeigen der zuletzt eingelesenen primitiven Evidenz.
- Das Anzeigefeld Current time, zum Anzeigen des Zeitstempels des aktuellen Frames.
- Bis zu sechs Tabs (Jess, Rules fired, Facts, Activations, Visualisation, Recognition), die im großen Feld (dem Blackboard) Informationen über den IP zeigen. Die Bedeutung der einzelnen Tabs wurde bereits in Abschnitt 6.1.2.2 kurz erläutert. Weitere Details werden nachfolgend dargestellt.

6.2.2.1 Blackboard

Jeder der oben aufgeführten sechs Tabs enthält für jeden alternativen Thread des ausgewählten Teilmodells ein Fenster (nebeneinander angeordnet). Diese werden im Laufe des Interpretationsprozesses dynamisch erzeugt und können auch während des aktiven Interpretationsprozesses durch Scrollen betrachtet werden. Jedes Fenster zeigt zusätzlich folgende Informationen: die Identifikationsnummer des Threads, den Status:

- grau ≙ gestorben – dies wird außerdem durch einen dunklen Hintergrund des Fensters angezeigt,
- gelb ≙ aktiv,
- grün ≙ abgeschlossen,

das Teilmodell (wichtig, falls in der Auswahlbox Model der Status all selektiert ist) und das Feld Rating zur Anzeige des probabilistischen Präferenzmaßes. Es ist konfigurierbar, ob die gestorbenen Threads angezeigt werden und ob die Fenster dynamisch anhand des Präferenzmaßes sortiert werden (s. Abschnitt 6.3).

6.2.2.2 Visualisation-Tab

Der Visualisation-Tab zeigt die alternativen Interpretationen durch dynamische Visualisierung der Hypothesengraphen an (s. Abbildung 40). Aggregate und Evidenzen werden durch Boxen dargestellt. Die Farben der Boxen haben folgende Bedeutung:

- blau bedeutet, dass eine Instanz hypothetisiert ist,
- grün bedeutet, dass ein Konzept instanziiert ist,
- cyan bedeutet, dass eine Instanz halluziniert ist,
- gelb zeigt eine primitive oder höhere Evidenz an.

Der Status der Hypothesengraphen wird kontinuierlich aktualisiert und repräsentiert den Zustand der entsprechenden Regelmaschine des Threads. Für alle Konzepte und Evidenzen wird der Anfangs- und Endzeitpunkt dargestellt, entsprechend des aktuellen Zustands des zugehörigen ZCN. Auf diese Weise können zeitliche Vorhersagen, die den propagierten Werten des ZCN entsprechen, bei hypothetisierten Konzepten direkt abgelesen werden. Bei primitiven Evidenzen werden zusätzlich die Namen der zugehörigen physikalischen Objekte angezeigt.

6.2.2.3 Jess-Ausgabe-Tabs

Die Jess-Ausgabe-Tabs (Jess, Rules fired, Facts, Activations) zeigen dynamisch die textuelle Ausgabe der Regelmaschinen an. In Abbildung 41 ist dies beispielhaft anhand des Jess-Tabs dargestellt. Die originale Ausgabe der Regelmaschine wird in schwarz dargestellt, wobei zur besseren Übersicht das Feuern der Regeln in grün erscheint. Nachrichten, die vom `Control`-Objekt oder dem `DSMain`-Objekt stammen, werden in blau dargestellt. Sie zeigen übergeordnete Interaktionen der Threads an, beispielweise das Starten und das Klonen von Regelmaschinen. Auch die von Java ausgelösten Aktionen, wie die Überprüfung der Erfüllbarkeit des ZCN, werden in blau dargestellt.

6.2.2.4 Recognition-Tab

Wenn ein Thread vollständig instanziiert und damit abgeschlossen ist, wird das Ergebnis der Interpretation in textueller Form im Recognition-Tab ausgegeben. Diese Ausgabe kann zusätzlich in eine Textdatei geschrieben werden, um sie anschließend weiter zu verarbeiten und auszuwerten (s. Abschnitt 6.3). Die Auswertung der Interpretationsergebnisse wird in Kapitel 7 ausführlich behandelt.

6.2.3 Primitives-Tab

Der Primitives-Tab zeigt dynamisch die eingelesen primitiven Evidenzen an (s. Abbildung 42). Die blauen Linien visualisieren die Zeitintervalle, in denen die jeweilige Evidenz vorkommt. Die dargestellten Namen der Evidenzen beinhalten eine interne ID. In Klammern dahinter wird die originale ID der Evidenz angezeigt.

6.2.4 Log-Tab

Durch eine entsprechende Konfigurierung (s. Abschnitt 6.3) werden sämtliche Protokollierungen in diesem Tab ausgegeben. Dies ist vor allem dann nützlich, wenn SCENIOR nicht in einer Entwicklungsumgebung, sondern als Stand-Alone-Programm gestartet wird.

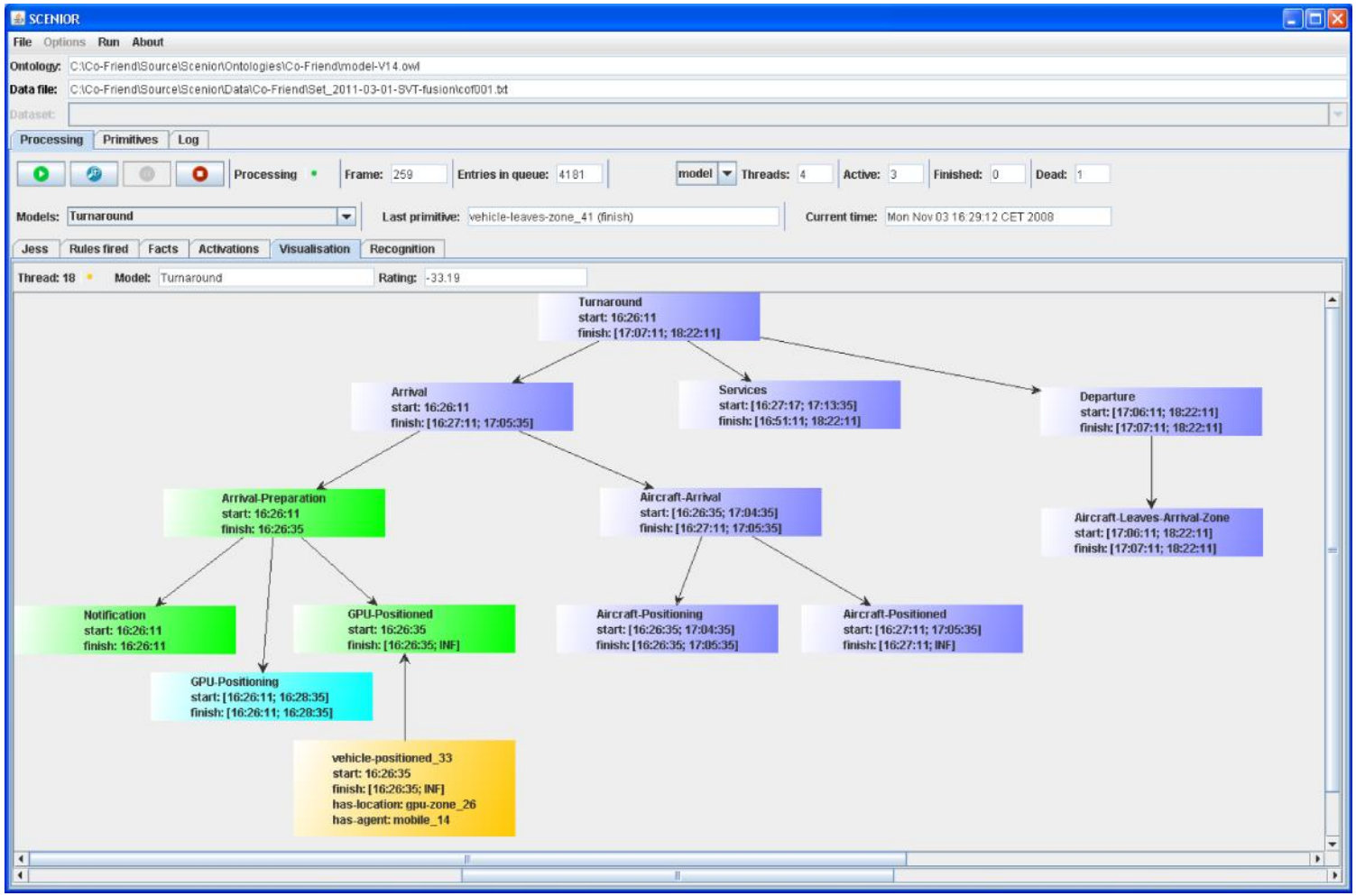


Abbildung 40: Screenshot von SCENIOR mit Visualisierung des Hypothesengraphs.

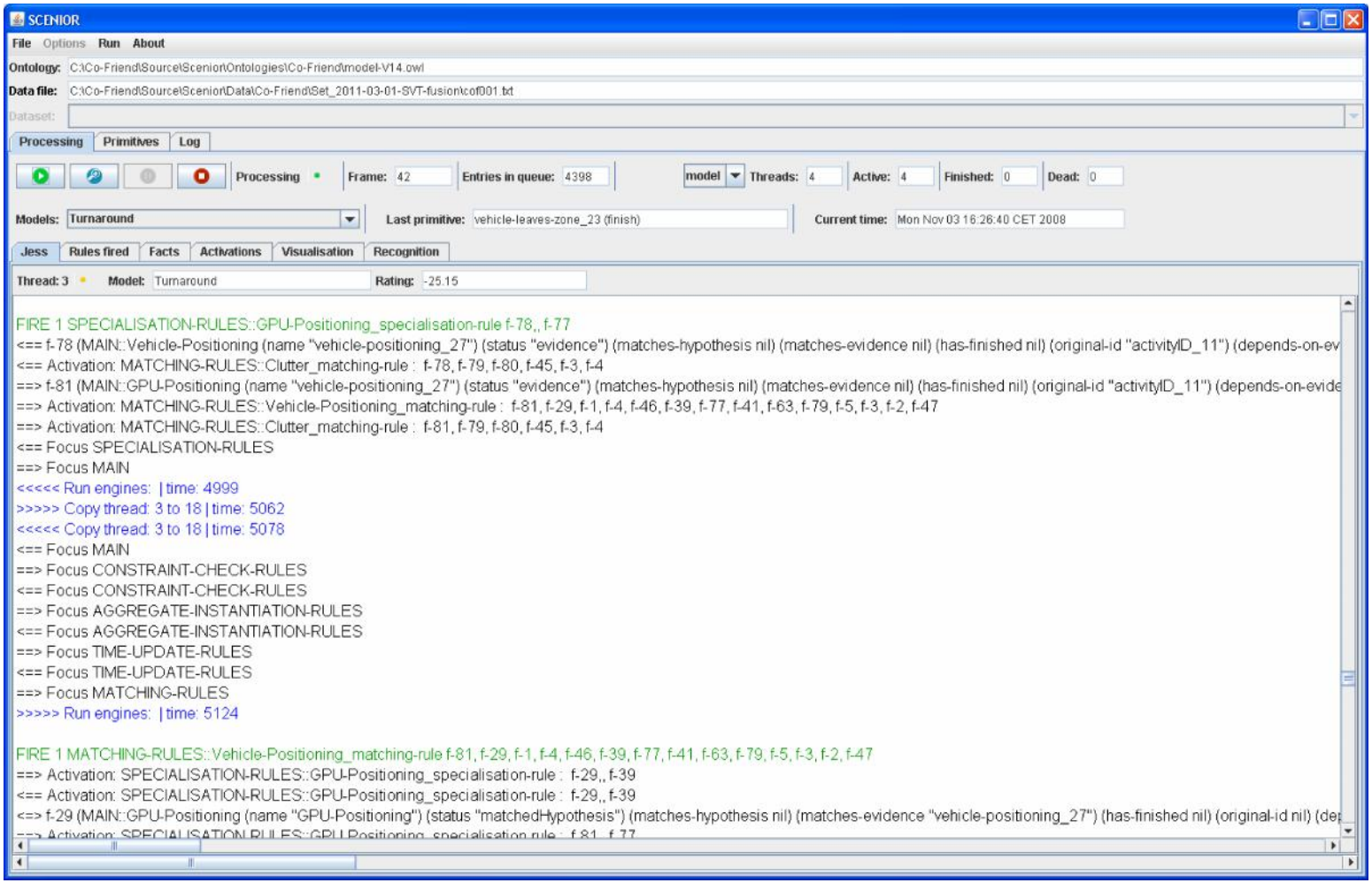


Abbildung 41: Screenshot von SCENIOR mit Jess-Ausgabefenster.

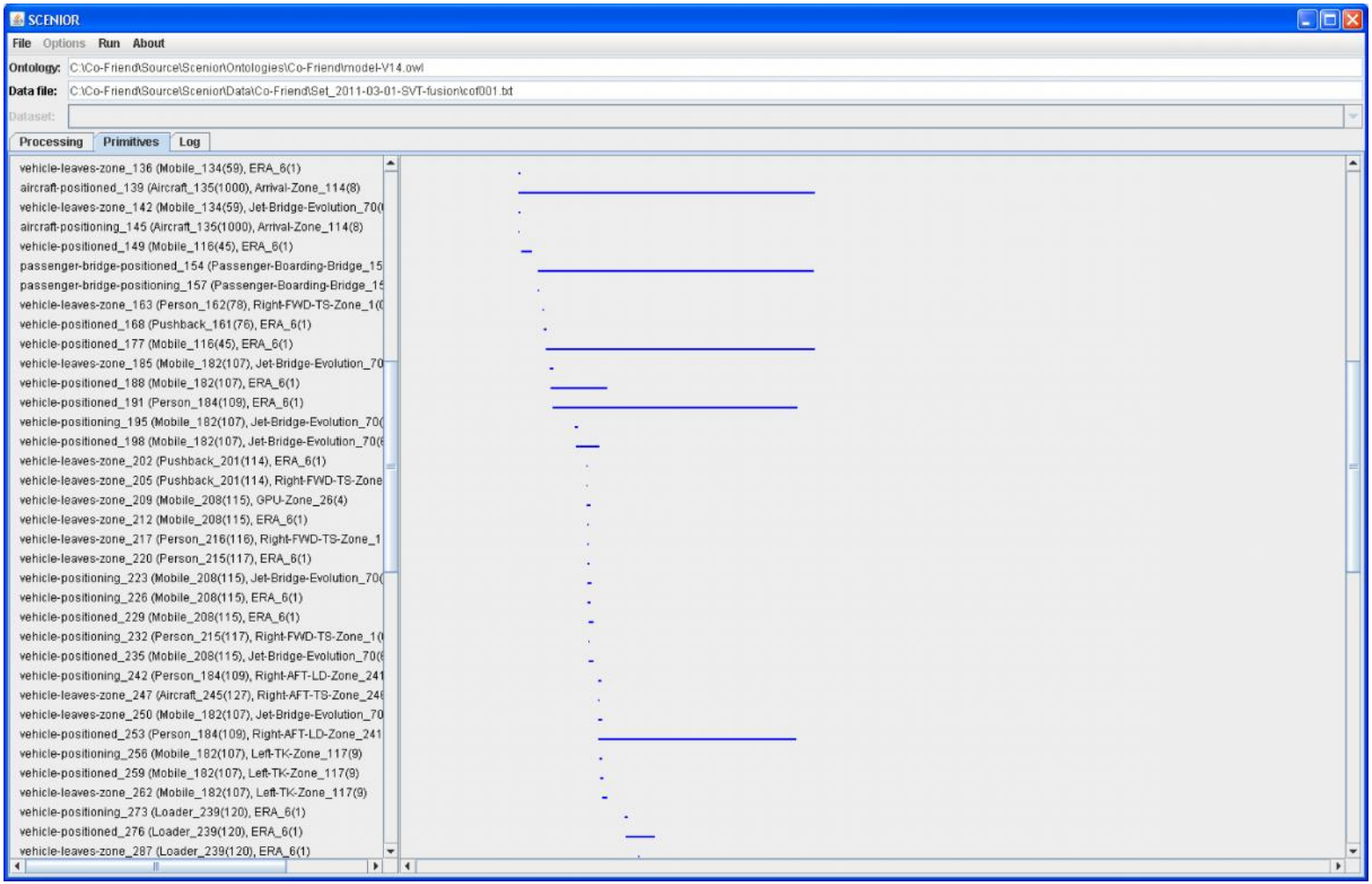


Abbildung 42: Screenshot von SCENIOR mit Anzeige der primitiven Evidenzen.

6.3 Konfigurierung

SCENIOR verfügt über eine Reihe von Konfigurationsschaltern, mit denen das System konfiguriert werden kann. Diese Konfigurationsschalter befinden sich in der Klasse `Config` im Modul `mainSI.main`. Eine Übersicht über wichtige Konfigurationsschalter ist in Anlage B.1 aufgeführt, tabellarisch und nach Themen sortiert.

Die Einstellungen der Konfigurationsschalter in der Klasse `Config` können durch eine private Konfigurationsdatei `*.properties` überschrieben werden. Das Einlesen einer solchen Datei geschieht in der Methode `loadFromFile()` der Klasse `Config`. Dies ist zum Einen nützlich, wenn mehrere Personen gleichzeitig Entwicklungsarbeiten am System durchführen und zum Anderen, um in einfacher Weise von einer Domäne zu einer anderen zu wechseln. Dazu können mehrere Konfigurationsdateien vorgehalten werden, z.B. `project_A.properties` und `project_B.properties`. Die entsprechende Konfigurationsdatei wird dann in der `main()`-Methode der Klasse, die SCENIOR startet (aktuell ist dies die Klasse `CFLauncher` im Modul `coFriend.main`), durch den statischen Aufruf `Config.loadFromFile(projekt_X.properties)` eingelesen.

6.4 Erweiterungsmöglichkeiten

Folgende Erweiterungsmöglichkeiten für SCENIOR sind denkbar:

- Integration neuer Constraint-Mechanismen,
- Nutzung für die statische Szeneninterpretation,
- Verarbeitung weiterer Eigenschaften von physikalischen Objekten,
- Auslösen weiterer Aktionen.

Integration neuer Constraint-Mechanismen. Die Möglichkeit, neue Constraint-Mechanismen in SCENIOR zu integrieren, wurden bereits in Abschnitt 6.1.2.3 behandelt.

Statische Szeneninterpretation. Durch die inkrementelle Verarbeitung von Evidenzen und die Wichtigkeit der Verarbeitung zeitlicher Constraints ist SCENIOR in weiten Teilen auf die dynamische Szeneninterpretation ausgerichtet. Eine einfache

Möglichkeit, SCENIOR für die statische Szeneninterpretation zu nutzen, wäre, alle Evidenzen mit dem gleichen Zeitstempel zu versehen und das ZCN abzuschalten (s. Abschnitt 6.3). Dadurch wird simuliert, dass sich alle Evidenzen in einem Frame befinden, und dies würde einer statischen Situation entsprechen. Für eine optimale Lösung wäre es notwendig, eine Klasse `SSMain` (`SS` ≙ *static scene interpretation*), analog zu `DSMain`, von der Klasse `SIMain` abzuleiten und entsprechend den Anforderungen einer statischen Verarbeitung zu implementieren. Außerdem müssten weitere Klassen, beispielsweise `DSHypothesizer`, `DSJessFileGenerator` und `DSDataProcessThread` erweitert werden (s. Abschnitt 6.1.2.1).

Verarbeitung von Eigenschaften physikalischer Objekte. Bisher können für physikalische Objekte keine weiteren Eigenschaften verarbeitet werden. Dies wäre jedoch wünschenswert, um beispielsweise in einem Constraint ausdrücken zu können, dass ein Fahrzeug gelb sein muss. Weitere Eigenschaften könnten Werte betreffen, wie Geschwindigkeit oder Temperatur (bei entsprechenden Sensoren, s. Abschnitt 7.2). Grundsätzlich stellt dies kein Problem dar, denn auch alle physikalischen Objekte werden bereits – genau wie konzeptuelle Objekte – durch separate Fakten in den Regelmaschinen repräsentiert. Sie können in Jess-Regeln in derselben Weise adressiert und verarbeitet werden, wie konzeptuelle Objekte. Des Weiteren wird in Abschnitt 7.2.2 eine Erweiterung von SCENIOR vorgestellt, die es erlaubt, dass primitive Evidenzen beliebig viele physikalische Objekte als Argumente enthalten können. Dies müsste auf Eigenschaften physikalischer Objekte erweitert werden.

Auslösen weiterer Aktionen. Es ist in einfacher Weise möglich, SCENIOR so zu erweitern, dass dynamisch weitere Aktionen ausgelöst werden, abhängig von bestimmten Interpretationsergebnissen. Dabei könnte es sich beispielsweise um das Auslösen eines Alarms handeln. Um dies zu erreichen, kann sich eine abgeleitete Klasse von `SIMain` oder `DSMain` bei dem `Control`-Objekt mit `control.addControlEventListener(this)` als sog. Event-Listener anmelden. Durch Implementierung einer Methode `controlEventHappened()` in dieser Klasse kann dann auf bestimmte Ereignisse der Regelmaschinen reagiert werden (s. Abbildung 43). Das `controlEvent`-Objekt enthält detaillierte Informationen über die Ausgabe der Regelmaschinen, beispielsweise, dass ein bestimmtes Teilmodell vollständig instanziiert wurde. Daraufhin können dann weitere Aktionen ausgelöst werden.

```
public void controlEventHappened(controlEvent ce)
{
    switch(ce.getType())
    {
        case JESS_OUTPUT:
             if (ce.jessOutput.model.equals(„Teilmodell“) &&
                 ce.jessOutput.hasFinished == true)
             {
                 // löse Alarm aus
             }
             break;
    }
}
```

Abbildung 43: Durch Implementierung der Methode `controlEventHappend()` kann auf Interpretationsergebnisse reagiert werden.

7 Experimente

In diesem Kapitel werden die vorgestellten Verfahren und das Interpretationssystem SCENIOR anhand von künstlich generierten und realen Daten demonstriert und evaluiert. In Abschnitt 7.1 werden dazu ausführliche Experimente zur Domäne der Flughafenvorfeldaktivitäten dargestellt. In Abschnitt 7.2 wird anhand zweier Beispiele aus der Domäne intelligenter Wohnumgebungen demonstriert, dass SCENIOR auch für andere Domänen verwendet werden kann.

7.1 Domäne 1: Flughafenvorfeldaktivitäten

Ziel des Co-Friend-Projekts [17] war die automatische Erkennung von Flugzeug-Serviceaktivitäten, wie z.B. Ankunftsvorbereitung, Be- und Entladen des Gepäcks, Betankung, Catering, Heran- und Wegfahren der Passagierbrücke, Abfahrt des Flugzeugs, etc. Mögliche Anwendungsszenarien eines derartigen Systems sind die automatische Überprüfung einer korrekten und vollständigen Abfertigung, die Einhaltung sicherheitsrelevanter Bestimmungen und die Vorhersage des zeitlichen Verlaufs einer Abfertigung.

Durch den Einsatz standardisierter Modellierungstechniken für die Entwicklung der Ontologie, welche die zu erkennenden Aktivitäten repräsentiert, und die Integration überwachter und unüberwachter Lernverfahren, sollte eine flexible Anpassung an neue Gegebenheiten gewährleisten werden. Dies umfasst die mögliche Verwendung des Systems auf anderen Flughäfen oder die Anpassung an Veränderungen der Abfertigungsabläufe, andere Fahrzeuge, Zonen und Ausrüstungsgegenstände, sowie neue Sicherheitsbestimmungen. Die Entwicklung neuer Lernverfahren stellten einen zentralen Aspekt des Co-Friend-Projekts dar (s. [17], [85], [100]).

Für die Videoakquisition wurde auf dem Flughafen Toulouse-Blagnac ein Kameranetzwerk installiert (s. Abschnitt 7.1.1). Die verschiedenen Systeme der Projektpartner wurden in einer gemeinsamen Co-Friend-Plattform integriert (s. Abschnitt

7.1.2). Auf diese Weise konnte das Gesamtsystem reale Videosequenzen im Online-Modus verarbeiten, mit SCENIOR als Interpretationskomponente. Die in diesem Kapitel dargestellten Experimente wurden ausschließlich im Offline-Modus durchgeführt, d.h., die Eingabedaten wurden von Dateien eingelesen, deren Inhalt durch die entsprechenden Teilsysteme der Co-Friend-Plattform erzeugt wurde.

7.1.1 Videoakquisition

Auf dem Vorfeld „Echo40“ des Flughafens Toulouse-Blagnac wurde ein Kameranetzwerk installiert, das aus sieben bereits vorhandenen festen Kameras besteht (die für das vorangegangene Projekt *Avitrack* [15] genutzt wurden) und für das Co-Friend-Projekt durch vier PTZ-Kameras und zwei zusätzliche feste Kameras erweitert wurde. Eine Übersicht über die Kamerapositionen ist in Abbildung 44 dargestellt. Der Projektpartner AKKA war für das Kameranetzwerk verantwortlich, für weitere technische Details siehe [23]).

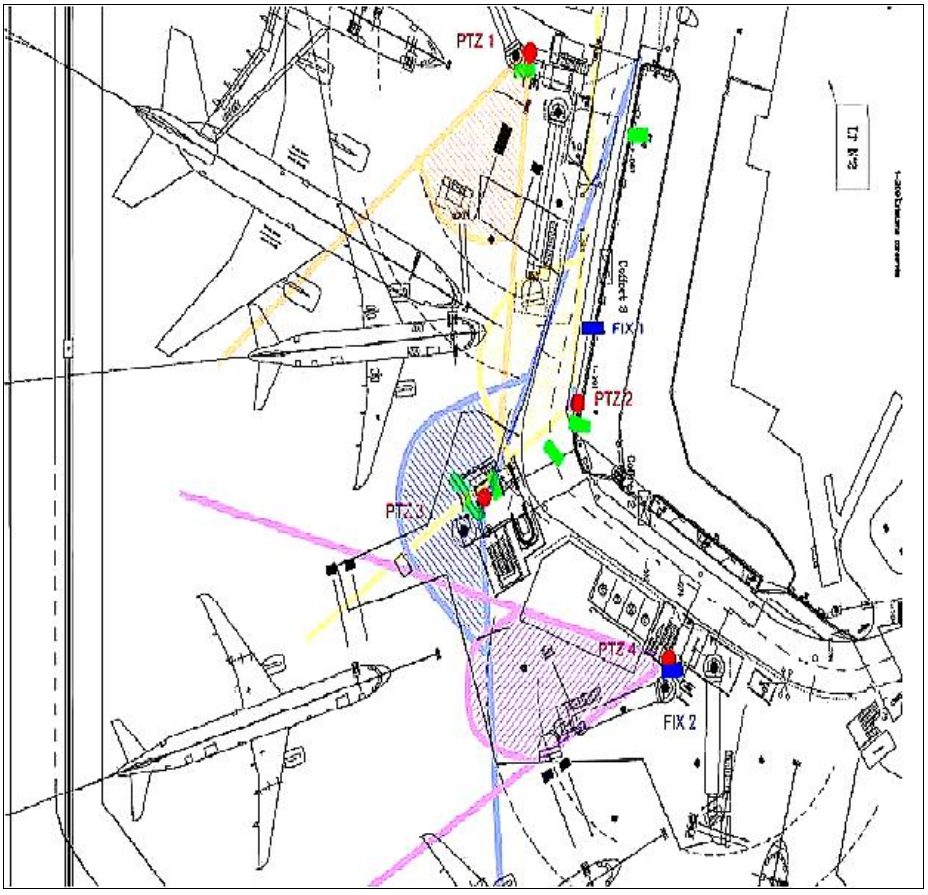


Abbildung 44: Kamerapositionen auf dem Flughafenvorfeld Toulouse-Blagnac. Feste Kameras sind grün (aus dem Avitrack-Projekt) und blau dargestellt, PTZ-Kameras rot (aus [49]).

7.1.2 Die Co-Friend-Plattform

In Abbildung 45 ist eine Übersicht über die globale Architektur der Co-Friend-Plattform dargestellt (s. auch [49], [96]). Die von den Kameras gelieferten Videosequenzen werden zunächst separat verarbeitet (symbolisiert durch die Kästchen „Tracker pro Kamera“). Die Verarbeitung besteht aus zwei wesentlichen Schritten: dem Detektieren und Verfolgen einzelner Objekte, beispielweise Fahrzeuge, Personen und Ausrüstungsgegenstände. Das Detektieren von Objekten basiert zum Einen auf dem Verfahren des sog. *Colour Mean and Variance* (s. [108]) – dessen Idee die Trennung von Regionen des Vordergrunds und des Hintergrunds ist – und zum Anderen auf dem Verfahren des *Adaptive Gaussian Mixture Model* (s. [110]), welches auf der Basis des optischen Flusses arbeitet. Das Verfolgen der Objekte wird u.a. durch das sog. *KLT Tracking* (*Kanade–Lucas–Tomasi Feature Tracking*) realisiert (s. [49]).

Die Ergebnisse der einzelnen Tracker werden anschließend an das Modul „Datenfusion“ weitergegeben. Dieses hat die Aufgabe, die separaten Observierungen zu einem Ergebnis zusammenzufassen, um dadurch eine hohe Robustheit zu erreichen, beispielsweise gegenüber Verdeckungen. Die Datenfusion besteht aus einer Reihe verschiedener Verfahren, auf die hier nicht näher eingegangen wird, für nähere Details siehe [50]). In Abbildung 46 (links) ist das Bild einer festen Kamera dargestellt. Die Boxen in Purpur zeigen Detektionen dieser Kamera an, die schwarzen Boxen das Ergebnis der Datenfusion (welches in dieses Bild zurückgerechnet wurde). Diese Module bilden zusammen die niedere Bildverarbeitungskomponente (s. Abbildung 1), für die der Projektpartner der Universität Reading[18] verantwortlich war.

Nach der Datenfusion werden die Ergebnisse an den Detektor für einfache Ereignisse und Zustände (engl.: *Simple event detector* $\hat{=}$ SED) und die PTZ-Kontrolle weitergegeben. Die PTZ-Kontrolle gehört zur Komponente der sog. *Specific Vision Tasks* (SVTs), die im weiteren Verlauf dieses Abschnitts besprochen wird.

[18] http://www.reading.ac.uk/sse/research/sse-computing.aspx

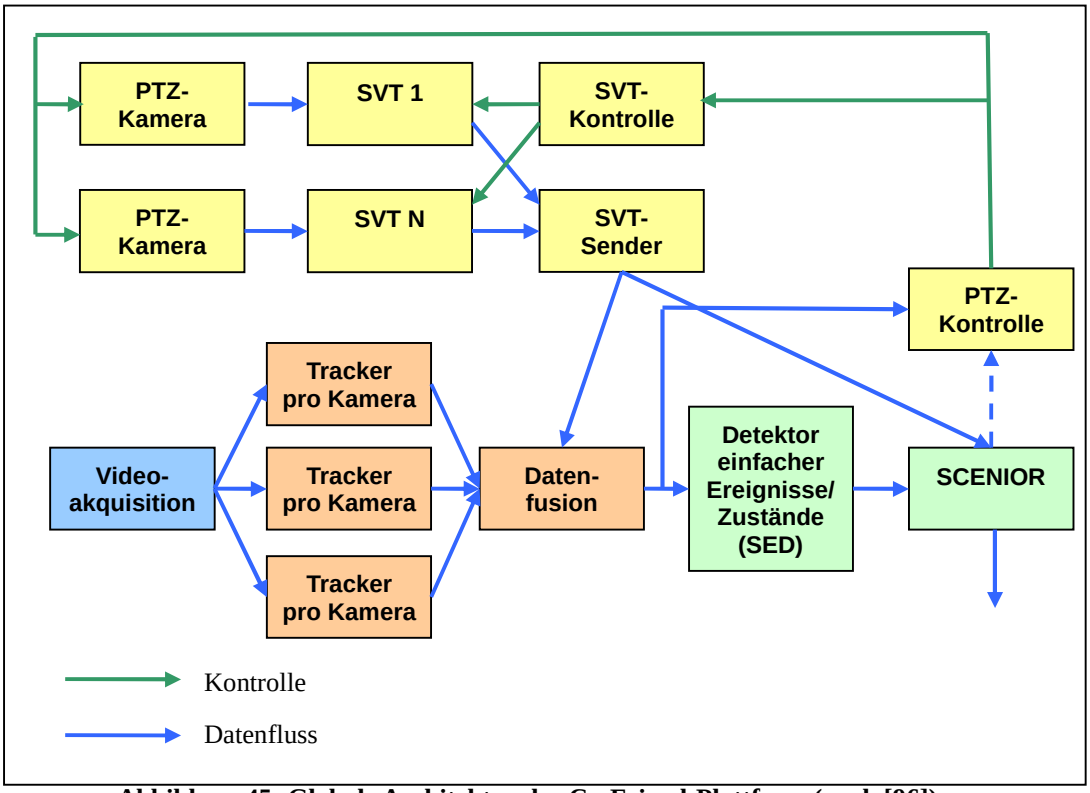


Abbildung 45: Globale Architektur der Co-Friend-Plattform (nach [96]).

Der vom Projektpartner INRIA[19] entwickelte SED kann als die Mittelschicht des in Abbildung 1 dargestellten allgemeinen Schichtenmodells für die Szeneninterpretation betrachtet werden. Da er jedoch, wie SCENIOR, ebenfalls mit Modellen einer Wissensbasis arbeitet, ist er tatsächlich eine Mischung aus Mittelschicht und höherer Bilddeutungskomponente. Die Wissensbasis des SED umfasst zwei Aspekte:

- eine geometrische Beschreibung der statischen Zonen des Vorfeldes und der beweglichen Zonen des Flugzeugs, welche nach dem Stillstand relativ zur Spitze des Flugzeugs definiert werden,
- Modelle für einfache Ereignisse und Zustände.

[19] http://www-sop.inria.fr/pulsar/index.html

Die Modelle des SED sind aus entwicklungstechnischen Gründen nicht Teil der Ontologie (wobei dies anzustreben wäre), mit der SCENIOR arbeitet. Sie sind in einem nichtstandardisierten Format beschrieben, welches von INRIA entwickelt wurde. In Definition 7-1 ist die Beschreibung des Models `Vehicle-Positioned` dargestellt; es wird instanziiert, wenn sich ein Fahrzeug `v1` länger als eine bestimmte Zeit (`MIN_STATE_TIME`) innerhalb einer Zone `z1` befindet.

```
Composite-State(Vehicle-Positioned,                                    (7-1)
    Physical-Objects((v1:Vehicle), (z1:Zone))
    Components((c1:Primitive-State Vehicle-Inside-Zone(v1,
                z1)))
    Constraints((duration(c1) >= MIN_STATE_TIME))
)
```

Die erkannten Ereignisse und Zustände vom SED dienen als Eingabedaten für SCENIOR. Zu beachten ist hier, dass `Vehicle-Positioned` in der Wissensbasis des SED ein `Composite-State` ist. Für SCENIOR sind jedoch grundsätzlich alle Evidenzen primitiv, die vom SED geliefert werden, denn die zugehörigen Konzepte bilden die Blätter der Modelle in der OWL Ontologie. Für detaillierte Informationen zum SED, siehe [38].

Abbildung 46: Links: Bild einer festen Kamera mit detektieren Objekten. Rechts: Bild einer PTZ-Kamera mit detektiertem Objekt (aus [50], [96]).

Die Idee der Specific Vision Tasks (SVTs) ist es, durch den Einsatz von PTZ-Kameras besonders schwierig zu erkennende Aktionen zu erfassen, wie z.B. das Andocken des Tankstutzens. Ausgelöst durch bestimmte Ereignisse oder Zustände – in diesem Fall die Positionierung des Tankfahrzeugs in der Tankzone – steuert die PTZ-Kontrolle über die SVT-Kontrolle ein oder mehrere PTZ-Kameras an, um sie durch Schwenken und Zoomen auf bestimmte Bereiche zu fokussieren (die Parameter der Ansteuerung sind in der Wissensbasis des SED enthalten und werden von der PTZ-Kontrolle ausgewertet). Durch *Template-Matching-Verfahren*

wird nun das Ereignis bzw. der Zustand erfasst (für nähere Details siehe [96]). Die Ergebnisse werden anschließend vom SVT-Sender an die Datenfusion und SCENIOR zur weiteren Verarbeitung geschickt (s. Abbildung 45). Angedacht war auch ein Datenfluss von SCENIOR zur PTZ-Kontrolle (symbolisiert durch den gestrichelten Pfeil in Abbildung 45), um in Abhängigkeit von Interpretationsergebnissen bestimmte SVTs auszulösen; dies wurde jedoch aus Zeitgründen nicht mehr realisiert.

7.1.3 Eingabedaten

Die hier durchgeführten Experimente basieren auf realen Videosequenzen des Flughafenvorfelds Toulouse-Blagnac, welche von der Co-Friend-Plattform verarbeitet wurden. Der SED liefert seine Ausgaben im Offline-Modus in Form von XML-Dateien. Sie stellen die Eingabe für SCENIOR dar und enthalten die prozessierten primitiven Evidenzen in einer Frame-basierten Darstellung mit einer Granularität von etwa einem Frame pro Sekunde. Je nach Länge der Abfertigung enthält eine Datei zwischen ca. 3000 bis 6000 Frames. In Abbildung 47 ist ein typischer Frame dargestellt. Er enthält im Wesentlichen folgende Informationen:

- den Zeitstempel des Frames,
- eine Liste der beteiligten physikalischen Objekte (`ListMobileObjects`),
- eine Liste der primitiven Ereignisse und Zustände (`ListActivities`).

Die physikalischen Objekte sind über IDs mit den zugehörigen Ereignissen und Zuständen verknüpft. Jedes Ereignis bzw. jeder Zustand wird über den Zeitraum seiner Existenz entsprechend in allen Frames mit einer eindeutigen ID geliefert.

Die Eingabedateien im XML-Format sind sehr groß und unhandlich: Das Editieren und Modifizieren zum Zweck experimenteller Untersuchungen gestaltet sich schwierig. Daher wurden die XML-Dateien in ein eigenständiges Format umgewandelt. In Anlage C.1 ist ein Datensatz dieses Formats dargestellt. Jede Zeile definiert ein primitives Ereignis bzw. einen primitiven Zustand, mit Angabe des Anfangs- und Endzeitstempel (er besteht zusätzlich aus einem Datum, welches hier aus Gründen der Übersichtlichkeit nicht dargestellt ist), dem Namen des Ereignisses / Zustands, den dazugehörenden physikalischen Objekten und deren IDs. Das Format ist demzufolge nicht mehr Frame-basiert, daher brauchen die primitiven Evidenzen selbst keine ID; sie bekommen von SCENIOR eine interne ID zugewiesen.

```
01 <SUVideoFrame ID="9" timeYear="2008" timeMonth="1" timeDay="20"
02                    timeHour="21" timeMin="1" timeSec="8" timeMs="0">
03   <ListMobileObjects>
04     <MobileObject ID="24" type="VEHICLE:51-OTHER:27-PERSON:20">
05       <TimeStart timeHour="21" timeMin="0" timeSec="0" timeMs="0"/>
06       <Info3D x="3.50" y="11.88" z="0.00" w3D="2.00" h3D="2.00" l3D="6.00"/>
07     </MobileObject>
08   </ListMobileObjects>
09   <ListActivities>
10     <Activity ID="14" name="v_Inside_Zone" Confidence="1.00" type="STATE"
11                      AType="" AText="">
12       <TimeStart timeHour="21" timeMin="1" timeSec="2" timeMs="0" />
13       <ListActivityPhysicalObjects>
14         <ActivityPhysicalObject type="0" ID="24" name="V_24" />
15         <ActivityPhysicalObject type="1" ID="17" name="GPU_Zone" />
16       </ListActivityPhysicalObjects>
17     </Activity>
18   </ListActivities>
19 </SUVideoFrame>
```

Abbildung 47: Ein Frame einer Ausgabedatendatei vom SED im XML-Format.

Aus entwicklungstechnischen Gründen sind die Bezeichnungen der Konzepte des SED nicht (vollkommen) identisch mit denen der Ontologie für SCENIOR; sie werden mit Hilfe einer entsprechenden Zuordnungstabelle übersetzt (siehe `MATCHING_TABLE` in Anlage B.1). Weiterhin werden alle primitiven Evidenzen mit einem Agenten und einer Zone geliefert, daher werden die Rollenbezeichner nicht explizit aufgeführt. Alle primitiven Evidenzen haben eine Instanz des Konzepts `Mobile` als Agenten, deshalb kann dieser auch eine Instanz des Konzepts `Person` sein.

Die Bildverarbeitungskomponente liefert zu den detektierten Objekten Konfidenzwerte, welche die Güte der Klassifizierung beschreiben (für nähere Details siehe [49]). In Abbildung 47 ist dies in Zeile 4 durch `type="VEHICLE:51-OTHER:27-PERSON:20"` ausgedrückt. Diese Konfidenzwerte können sich über den zeitlichen Verlauf der zugehörigen primitiven Evidenz in jedem Frame ändern. In Abbildung 48 ist dies anhand eines Beispiels dargestellt: Jede Linie beschreibt den Konfidenzwert für einen bestimmten Objekttyp, aufgetragen über die Zeit eines entsprechenden Zustands (`Vehicle-Inside-Zone`). Eine genauere Untersuchung der Konfidenzwerte ergab, dass die in Abbildung 48 dargestellte Situation kein Einzelfall ist: Die Klassifizierung der Objekte funktionierte bis zum Ende des Co-Friend-Projekts nicht robust. Da die primitiven Zustände bereits zu ihrem Startzeitpunkt von SCENIOR verarbeitet werden (s. Abschnitt 5.2.2.4), ist es unter diesen Umständen nicht sinnvoll, die Klassifizierung des ersten Frames des jeweiligen Zustands zu nutzen; dies käme einer zufälligen Auswahl gleich. Angedacht wurde daher ein Verfahren, das die Klassifizierungsergebnisse in einem Vorverarbeitungsschritt über die Zeit mittelt, und in dem die Evidenz erst eine gewisse Zeit nach dem Startzeitpunkt an die Regelmaschinen übergeben

wird. Doch auch die Mittelung der Klassifikationsresultate brachten keine robusten Ergebnisse. Daher werden die Typinformationen der Agenten in den Experimenten mit realen Daten ignoriert (dies kann über den Konfigurationsschalter `IGNORE_OBJECT_TYPES` gesteuert werden, s. Anlage B.1). Jeder Agent wird zunächst als Instanz des Konzepts `Mobile` behandelt und entsprechend spezialisiert. Beispielweise wird eine Evidenz `Vehicle-Positioning(GPU 17, Left_TK_Zone)` durch eine Spezialisierungsregel zu `Tanker-Positioning(TANKER 17, Left_TK_Zone)` spezialisiert. Mit anderen Worten: für jedes Objekt, welches sich in der Tankzone `(Left_TK_Zone)` befindet, wird angenommen, dass es sich um ein Tankfahrzeug handelt. Die Idee, eine Objektidentifizierung aus der Funktion des Objekts abzuleiten, hatte bereits Badler [28] geäußert (s. auch Abschnitt 2.1).

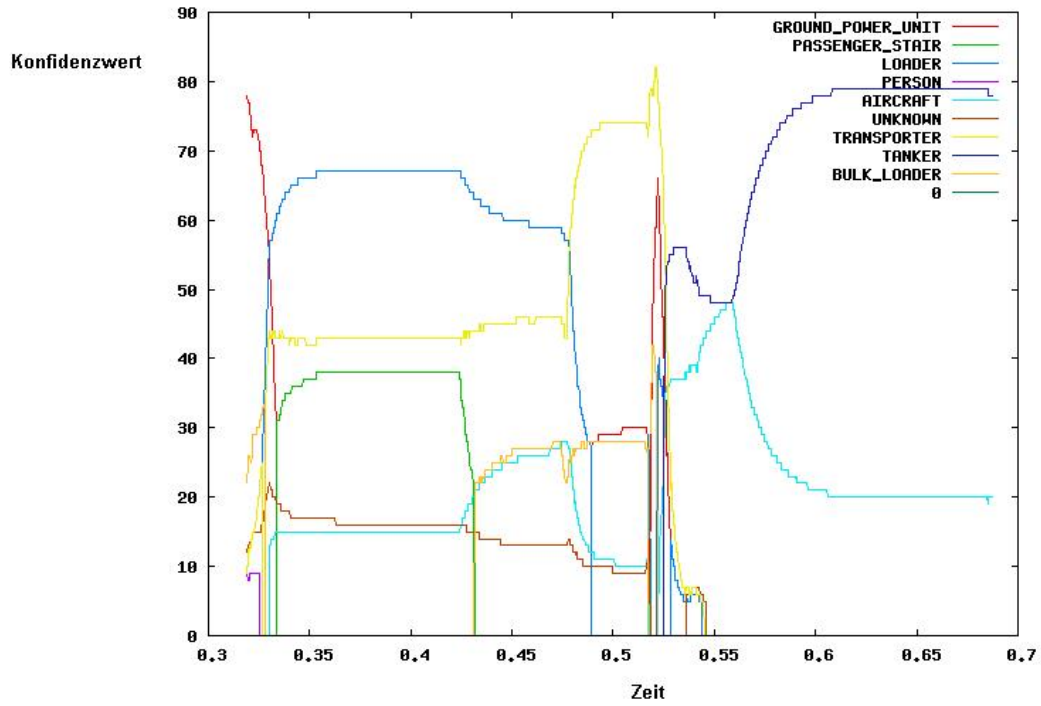


Abbildung 48: Konfidenzwerte für die Klassifizierung eines Objekts über den zeitlichen Verlauf einer primitiven Evidenz (`Vehicle-Inside-Zone`).

7.1.4 Probabilistische Modelle

Die probabilistischen zeitlichen Modelle der Vorfeldaktivitäten für die nachfolgenden Experimente wurden von Neumann [31] anhand der Statistiken von 52 annotierten Videosequenzen erstellt. Abbildung 49 zeigt eine Übersicht der Ereignisse, Zustände und deren modellierten Dauern und zeitlichen Abstände zueinander. Die beiden oberen Zahlen geben jeweils Mittelwert und Standardabweichung

der Normalverteilung an, die beiden unteren Zahlen den minimalen und maximalen Wert für das ZCN (in Minuten); dessen Constraints sind bewusst schwach gehalten, um nicht von vornherein zu viele Interpretationen auszuschließen. Negative Zahlen bedeuten, dass sich die Ereignisse bzw. Zustände überlappen können, beispielsweise ist es erlaubt, dass das `Aircraft-Positioning` bis zu zwei Minuten *vor* dem `GPU-Positioned` stattfindet. Nicht dargestellt ist das Laden und Entladen des Gepäcks im vorderen Bereich (`Unloading-FWD, Loading-FWD`), da es nahezu identisch zum Laden und Entladen des hinteren Bereichs strukturiert ist (`Unloading-AFT, Loading-AFT`). Da sich für die Präferenzmaße sehr kleine Werte ergeben, werden sie logarithmiert und sind deshalb negativ (je höher der Wert ist, desto besser ist die Bewertung der Interpretation).

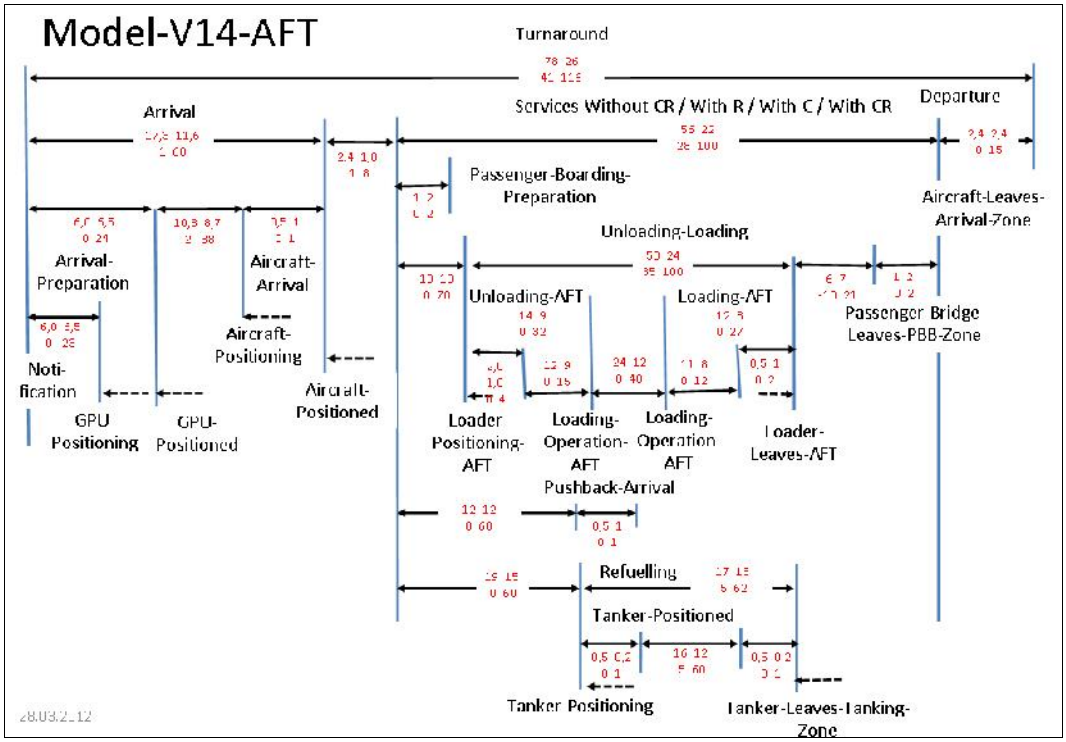


Abbildung 49: Übersicht über die probabilistische zeitliche Modellierung der Vorfeldaktivitäten (nach [31]).

7.1.5 Experimente mit künstlichen Daten

In diesem Abschnitt werden zunächst Experimente mit künstlich generierten Eingabedaten beschrieben. Die verwendete Ontologie entspricht der in Abschnitt 5.1.3 vorgestellten Struktur. Als erstes wird ein idealer Datensatz ohne Rauschen verwendet. Dieser wird in weiteren Experimenten mit einer steigenden Anzahl von Clutter-Elementen versetzt, um die Robustheit des Verfahrens zu analysieren. Alle Experimente wurden auf einem *Intel Xeon E5620* Quad-Core-Rechner mit 24 GB Ram und Windows 7 64-Bit-Betriebssystem durchgeführt.

7.1.5.1 Ein idealer Datensatz

Für dieses Experiment wurde ein im Sinne der probabilistischen Modelle (s. Abbildung 49) idealer Datensatz erstellt; er ist in Anlage C.1 dargestellt. Er enthält u.a. ein `Refuelling` und die entsprechenden Evidenzen für eine Ent- und Beladung im hinteren Bereich (`Unloading-AFT, Loading-AFT`). Das Halluzinieren von Evidenzen wurde abgeschaltet, die maximale Anzahl von Threads auf 100 gesetzt. Die Wahrscheinlichkeitsdichte für Clutter wurde in diesem Experiment näherungsweise auf den Wert 0,0001 gesetzt, da für die Modellvariante mit den meisten Blättern (mit `Refuelling` und `Unlaoding-And-Loading`) kein Clutter vorhanden ist.

In Tabelle 6 ist das Interpretationsergebnis dargestellt, Abbildung 50 zeigt einen Screenshot von SCENIOR mit der bestbewerteten `Turnaround`-Instanz. Insgesamt werden vier vollständige Abläufe gefunden. Dies ist dadurch zu erklären, dass das Auftreten von `Refuelling` optional ist und es weiterhin zulässig ist, dass das Ent- und Beladen des Gepäcks jeweils einzeln oder zusammen vorkommen kann. Ein Ablauf nur mit `Unloading-AFT` tritt nicht auf, da im Teilmodell `Services-Without-CR` das Constraint verletzt wird, dass spätestens 21 Minuten nach dem Ende des Ent- und/oder Beladens ein `Passenger-Bridge-Leaves-PBB-Zone` erfolgen muss. Die beste Bewertung erhält der Ablauf mit `Refuelling` und `Unloading-And-Loading`; zum Einen, weil das längere `Unloading-And-Loading` dem statistischen Modell von `Services` besser entspricht als der kürzere Vorgang, der nur `Loading-AFT` beinhaltet, zum Anderen, weil der Ablauf mit `Refuelling` und `Unloading-And-Loading` weniger Clutter-Elemente enthält (die Teile des `Refuelling`-Aggregats und des `Unloading-AFT`-Aggregats).

Idealer Datensatz ohne Clutter				
Threads (`MAX_NUMBER_OF_THREADS = 100`): insgesamt: 89, aktiv (max): 34, abgeschlossen: 15, gestorben: 61				
Clutter-Wahrscheinlichkeitsdichte: $\pm = 0.0001$				
Vollständige `Turnaround`-Instanzen: 4				
Laufzeit: 43 Sekunden				
Rang	1	2	3	4
Präferenzmaß	-49,35	-65,37	-82,72	-102,34
`Arrival`	x	x	x	x
`Passenger-Boarding-Prep`	x	x	x	x
`Unloading-AFT`	x	x		
`Loading-AFT`	x	x	x	x
`Refuelling`	x		x	
`Pushback-Arrival`	x	x	x	x
`Passenger-Bridge-Leaves-Z`	x	x	x	x
`Departure`	x	x	x	x

Tabelle 6: Interpretationsergebnis des idealen Datensatzes.

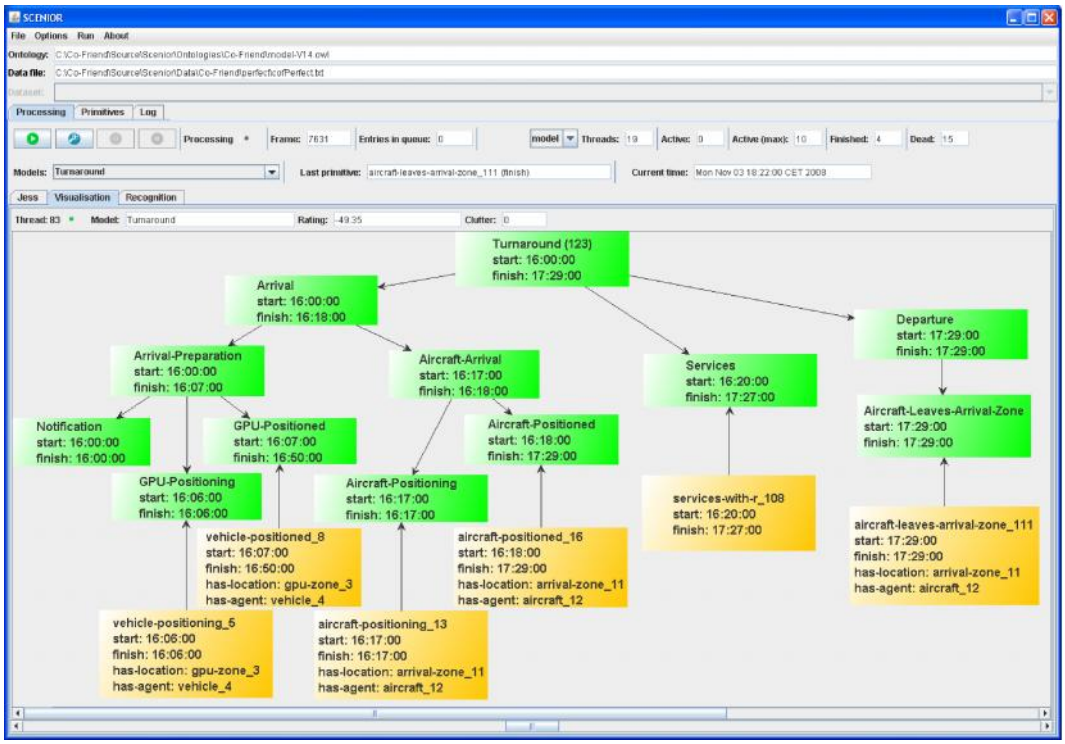


Abbildung 50: Erkannter `Turnaround` des idealen Datensatzes mit der besten Bewertung.

7.1.5.2 Datensätze mit Rauschen

Für die nachfolgenden Experimente wurde der ideale Datensatz aus Abschnitt 7.1.5.1 mit künstlichem Rauschen versetzt. Ein Rauschgenerator erzeugt dazu zufällige Clutter-Elemente der 23 verschiedenen primitiven Evidenztypen (Evidenzen von Konzepten, welche nicht in der Ontologie vorkommen, werden vorher von SCENIOR herausgefiltert und sind daher nicht relevant). Bei den punktuellen Ereignissen werden zufällige Zeitpunkte innerhalb des zeitlichen Rahmens des idealen Datensatzes erzeugt. Bei zeitlich ausgedehnten Zuständen bilden sie die Startzeitpunkte, zusätzlich wird eine zufällige Dauer zwischen einer und 30 Minuten generiert. Alle Agenten sind vom Typ `Mobile`, d.h., wie bei den Realdaten ist die Situation dahingehend erschwert, dass die Agenten entsprechend ihrer Funktion spezialisiert werden. Die IDs der Agenten werden ebenfalls zufällig ermittelt, abhängig von der Anzahl der erzeugten Elemente, d.h., bei einem Versuch mit 100 Clutter-Elementen werden IDs von elf bis 100 vergeben (die IDs von eins bis zehn sind bereits für die „echten“ Evidenzen vergeben).

Für die Clutter-Wahrscheinlichkeitsdichte kann vereinfacht ein konstanter Wert angenommen werden, oder sie kann basierend auf einer Statistik berechnet werden, welche die unterschiedlichen Evidenztypen berücksichtigt (dies kann über

den Konfigurationsschalter `USE_PRIM_STATISTIC` gesteuert werden, s. Anlage B.1). Der konstante Wert berechnet sich durch

$$\pm = \frac{n}{N}, \tag{7-2}$$

mit der Anzahl n von Clutter-Elementen beliebigen Typs und der Länge einer Abfertigung N, welche bei diesen Experimenten 81 Minuten beträgt. Für ein Experiment mit 100 Clutter-Elementen ergibt sich beispielsweise ein Wert von

$$\pm = \frac{100}{23 \cdot 81} \approx 0{,}054 \tag{7-3}$$

Dabei wird vereinfacht davon ausgegangen, dass die Clutter-Elemente zeitlich gleich verteilt sind und auch die 23 verschiedenen Evidenztypen gleich häufig auftreten (s. auch [81]). In diesem Fall berechnet sich $P_{clutter}$ (s. Formel 5-45) zu einem Zeitpunkt, bei dem sich $n_{current}$ Clutter-Elemente in einem Thread befinden, durch:

$$P_{clutter} = n_{current} \cdot \pm . \tag{7-4}$$

Im zweiten Fall wird das Vorkommen jedes Evidenztyps T_i gezählt und die Statistik in einer Datei gespeichert. Während des Interpretationsprozesses wird dann jeweils ein individuelles $\pm_i$ verwendet, abhängig vom Typ der Evidenz, die dem Clutter zugewiesen wird. Dann berechnet sich $P_{clutter}$ zu einem Zeitpunkt, bei dem sich jeweils $n_{i\_current}$ Clutter-Elemente eines Evidenztyps T_i in einem Thread befinden, durch:

$$P_{clutter} = \sum_i n_{i\_current} \cdot \pm_i . \tag{7-5}$$

Für die nachfolgenden Experimente wird jeweils eine Statistik der Clutter-Elemente verwendet. Instanzen, die keinen Clutter enthalten, werden in den Tabellen durch „x“ markiert, diejenigen mit Clutter mit „o“.

100 Clutter-Elemente

In Tabelle 7 ist das Interpretationsergebnis des Experiments mit 100 Clutter-Elementen dargestellt. Wie im Fall des idealen Datensatzes werden jeweils vier vollständige Abfertigungen erkannt, und diejenige mit `Unloading-And-Loading` und `Refuelling` liegt auf Platz eins. Dabei zeigt die Verwendung der Statistik das gleiche Ergebnis wie die Verwendung eines konstanten Werts der Clutter-Wahrscheinlichkeitsdichte. Eine genauere Untersuchung der Hypothesengraphen ergab, dass kein Clutter-Element verwendet, d.h. fälschlicherweise in die Interpretation aufgenommen wurde. Dies zeigt die gute Filterwirkung der zeitlichen Constraints und der Identitäts-Constraints; es ist unwahrscheinlich, dass diese von den zufällig erzeugten Clutter-Elementen erfüllt werden. Die Gesamtzahl der Threads steigt zwar auf 173 an, die maximale Anzahl der gleichzeitig aktiven Threads ist jedoch gegenüber dem idealen Datensatz nur um eins auf 35 gestiegen, d.h., es werden zwar viele neue Threads durch die Clutter-Elemente generiert, diese haben aber nur eine kurze Lebensdauer, da bestimmte Constraints verletzt werden.

100 Clutter-Elemente				
Threads (`MAX_NUMBER_OF_THREADS = 100`): insgesamt: 173, aktiv (max): 35, abgeschlossen: 15, gestorben: 145				
Vollständige `Turnaround`-Instanzen: 4				
Laufzeit: ca. 1,5 Minuten				
Rang	1	2	3	4
Präferenzmaß	-343,78	-347,75	-358,98	-359,95
`Arrival`	x	x	x	x
`Passenger-Boarding-Prep`	x	x	x	x
`Unloading-AFT`	x	x		
`Loading-AFT`	x	x	x	x
`Unloading-FWD`				
`Loading-FWD`				
`Refuelling`	x		x	
`Pushback-Arrival`	x	x	x	x
`Passenger-Bridge-Leaves-Z`	x	x	x	x
`Departure`	x	x	x	x

Tabelle 7: Interpretationsergebnis mit 100 Clutter-Elementen.

300 Clutter-Elemente

In Tabelle 8 ist das Interpretationsergebnis des Experiments mit 300 Clutter-Elementen dargestellt. Hier werden nun sechs vollständige Abfertigungen erkannt. Dies ergibt sich dadurch, dass ein `Loader-Positioning-AFT`, welches ein Clutter-Element ist, zusammen mit der späten `Loader-Operation-AFT` (welches eigentlich ein Beladen darstellt) ein zweites `Unloading-AFT` formt. Dieses späte Entladen liegt innerhalb des zeitlich erlaubten Abstands zum `Passenger-Bridge-Leaves-PBB-Zone`, so dass nun ein `Turnaround` nur mit `Unloading-AFT` – ohne `Loading` – erkannt wird, jeweils mit und ohne `Refuelling`.

Positiv zu vermerken ist, dass die Abfertigungen auf Rang eins bis drei keinen Clutter enthalten. Der ideale `Turnaround` mit `Refuelling` liegt zwar nur auf Rang zwei – bei den vorherigen Experimenten war er auf Rang eins – doch sein Präferenzmaß ist nur geringfügig niedriger als der von Rang eins. Dies erklärt sich folgendermaßen: generell werden Modelle mit mehr Evidenzen (z.B. eine `Service` mit `Refuelling`) durch die BCH schlechter bewertet als diejenigen mit weniger Evidenzen (z.B. ein `Service` ohne `Refuelling`), denn das Hinzufügen einer Evidenz kann das Präferenzmaß nur verschlechtern. Das wird jedoch dadurch ausgeglichen, dass ein Thread, der ein kleineres Modell repräsentiert, mehr Clutter-Elemente enthält – bei dem `Service` ohne `Refuelling` gegenüber dem mit `Refuelling` sind dies die drei Teile, aus denen sich `Refuelling` zusammensetzt. Je mehr Clutter-Elemente vorhanden sind, umso weniger wird das Auftreten von Clutter bestraft. Dies führt zu dem Effekt, dass mit steigender Anzahl der Clutter-Elemente eher kleinere Modelle bevorzugt werden. Des Weiteren ist die Bewertung von der Varianz abhängig; eine hohe Varianz bedeutet, dass die Normalverteilungsfunktion flacher ist und daher die berechneten Wahrscheinlichkeiten geringer sind. Größere Modelle haben tendenziell eine längere Dauer und höhere Varianzen.

300 Clutter-Elemente						
Threads (MAX_NUMBER_OF_THREADS = 100): insgesamt: 383, aktiv (max): 49, abgeschlossen: 20, gestorben: 350						
Vollständige Turnaround-Instanzen: 6						
Laufzeit: ca. 2,73 Minuten						
Rang	1	2	3	4	5	6
Präferenzmaß	-594,47	-596,99	-597,26	-602,37	-603,18	-606,23
Arrival	x	x	x	x	x	x
Passenger-Boarding-Prep	x	x	x	x	x	x
Unloading-AFT	x	x		o		o
Loading-AFT	x	x	x		x	
Unloading-FWD						
Loading-FWD						
Refuelling		x			x	x
Pushback-Arrival	x	x	x	x	x	x
Passenger-Bridge-Leaves-Z	x	x	x	x	x	x
Departure	x	x	x	x	x	x

Tabelle 8: Interpretationsergebnis mit 300 Clutter-Elementen.

1000 Clutter-Elemente

Tabelle 9 zeigt das Interpretationsergebnis des Experiments mit 1000 Clutter-Elementen, das bedeutet durchschnittlich ca. alle fünf Sekunden ein Störereignis. Hier werden 22 vollständige Abfertigungen erkannt. Es gibt fünf Unloading-AFT und zwei Loading-AFT-Instanzen; vier der fünf Unloading-AFT-Instanzen kombinieren sich mit den Loading-AFT-Instanzen zu acht Unloading-And-Loading-Instanzen (ein Unloading-AFT bildet mit einem Clutter-Element Loader-Positioning-AFT ein spätes Entladen und kann daher kein Unlaoding-And-Loading formen). Acht Unloading-And-Loading-AFT-Instanzen, zwei Loading-AFT-Instanzen und die späte Unloading-AFT-Instanz bilden elf Service-Without-CR-Instanzen und auch elf Service-With-R-Instanzen, insgesamt also 22 Turnaround-Instanzen. Die Abfertigungen ohne Clutter und ohne Refuelling liegen auf Rang eins und acht, die mit

Refuelling auf zwölf und 19. Angesichts der hohen Anzahl von Clutter-Elementen ist dies sicherlich noch ein recht gutes Ergebnis, zumal der Turnaround mit dem höchsten Präferenzmaß keinen Clutter enthält. Die Anzahl der maximal gleichzeitig aktiven Threads ist auf 125 gestiegen, liegt aber noch weit unter dem Limit, welches in diesem Experiment auf 200 gesetzt ist.

1000 Clutter-Elemente						
Threads (MAX_NUMBER_OF_THREADS = 200): insgesamt: 1224, aktiv (max): 125, abgeschlossen: 63, gestorben: 1148						
Vollständige Turnaround-Instanzen: 22						
Laufzeit: 16,82 Minuten						
Rang	1	2	8	12	13	19
Präferenzmaß	-658,47	-658,48	-659,71	-665,43	-665,44	-666,67
Arrival	x	x	x	x	x	x
Passenger-Boarding-Prep	x	x	x	x	x	x
Unloading-AFT	x	x		x	x	
Loading-AFT	x	o	x	x	o	x
Unloading-FWD						
Loading-FWD						
Refuelling				x	x	x
Pushback-Arrival	x	x	x	x	x	x
Passenger-Bridge-Leaves-Z	x	x	x	x	x	x
Departure	x	x	x	x	x	x

Tabelle 9: Interpretationsergebnis mit 1000 Clutter-Elementen.

1000 Clutter-Elemente, Reduzierung der Strahlbreite

Bisher wurde die maximale Anzahl der Threads (`MAX_NUMBER_OF_THREADS`) nicht erreicht, deshalb wird sie hier reduziert, um die Auswirkungen auf das Interpretationsergebnis zu beobachten. In Tabelle 10 ist das Interpretationsergebnis dargestellt. Wird die maximale Anzahl der Threads auf 70 bzw. 80 begrenzt, wird kein vollständiger `Turnaround` mehr erkannt. Dies lässt sich folgendermaßen erklären: In [81] legt Neumann dar, dass bei

$$q_i > \frac{1}{\sqrt{2®\pm_i}}, \tag{7-6}$$

wobei $\pm_i$ die Varianz eines Offsets oder einer Dauer ist, die Bewertung eines Clutter-Ereignisses immer besser ist als die modellbasierte Bewertung der BCH. Das Experiment zeigt, dass diese Bedingung bei 1000 Clutter-Elementen oft erfüllt ist und Hypothesengraphen mit vielen zugeordneten Evidenzen niedriger bewertet werden, selbst wenn sie der Statistik gut entsprechen. Wenn die maximale Anzahl der verfügbaren Threads erreicht ist, werden diese aussortiert, so dass keine vollständige Abfertigung mehr erkannt wird.

Bei einer maximalen Anzahl von 90 Threads werden acht vollständige Abfertigungen gefunden, wobei nur diejenige auf Platz eins keinen Clutter enthält (wieder ohne `Refuelling`). Bei einem Maximum von 110 Threads liegen die erkannten `Turnaround`-Instanzen ohne Clutter auf Platz eins, acht und zwölf. Dies zeigt, dass selbst bei einer hohen Clutter-Dichte gültige Interpretationen gefunden werden, solange genügend Threads zur Verfügung stehen. Zwar werden aufgrund der Bedingung aus Formel 7-6 unter diesen Umständen Threads mit vielen Evidenzen niedriger bewertet als diejenigen mit wenigen Evidenzen, doch letztere sterben zu einem gewissen Zeitpunkt, da ihr ZCN nicht mehr erfüllbar ist.

1000 Clutter-Elemente, Reduzierung der Strahlbreite						
MAX_NUMBER_OF_THREADS	70	80	90	110		
Vollständige Turnaround-Instanzen	-	-	8	13		
Laufzeit	15,04 Min.	15,5 Min.	16,04 Min.	16,73 Minuten		
Rang			1	1	8	12
Präferenzmaß			-657, 12	-657, 12	-659, 71	-664, 09
Arrival			x	x	x	x
Passenger-Boarding-Prep			x	x	x	x
Unloading-AFT			x	x		x
Loading-AFT			x	x	x	x
Unloading-FWD						
Loading-FWD						
Refuelling						x
Pushback-Arrival			x	x	x	x
Passenger-Bridge-Leaves-Z			x	x	x	x
Departure			x	x	x	x

Tabelle 10: Interpretationsergebnis mit 1000 Clutter-Elementen und Reduzierung der Strahlbreite.

100 Clutter-Elemente, aktivierte Halluzination

In diesem Experiment wird das Halluzinieren von fehlenden Evidenzen aktiviert. Dabei werden nur primitive Ereignisse (keine Zustände) in der Ontologie als halluzinierbar markiert. Dies ist dadurch begründet, dass die punktuellen Ereignisse weniger robust von der niederen Bildverarbeitung erkannt werden. Die Experimente mit realen Daten im nächsten Abschnitt werden zeigen, dass aufgrund der unzureichenden Qualität der gelieferten primitiven Evidenzen nicht auf das Halluzinieren bestimmter Evidenzen verzichtet werden kann, um eine robuste Interpretation vollständiger Abfertigungen zu erreichen.

In Tabelle 11 ist ein Teil des Interpretationsergebnisses dargestellt. Das Experiment zeigt eindrucksvoll, dass das Halluzinieren die Anzahl der generierten Threads ganz erheblich erhöht: Bei einem Limit von 200 Threads werden nun insgesamt 6186 Threads erzeugt (gegenüber 173 im Experiment ohne Halluzination), die maximale Anzahl gleichzeitig aktiver Threads liegt bei 259 (sie kann innerhalb eines Zyklus das durch `MAX_NUMBER_OF_THREADS` festgelegte Limit kurzzeitig überschreiten, s. Abschnitt 5.3.2.3) und die Anzahl vollständiger `Turnaround`-Instanzen ist von vier auf 84 angestiegen. Dies erklärt sich dadurch, dass durch das Halluzinieren wesentlich mehr Teilmodelle auf unterer Ebene instanziiert werden, und sich durch die Kombinatorik auch auf höheren Ebenen deutlich mehr instanziierte Teilmodelle ergeben. Beispielsweise reicht nun ein Clutter-Element von `Transporter-Positioned-AFT/FWD` aus, um eine Instanz von `Loading-Operation-AFT/FWD` zu bilden, da ein passendes `Transporter-Positioning-AFT/FWD` und `Transporter-Leaves-AFT` halluziniert wird. Tabelle 12 zeigt die Anzahl der jeweils vollständig instanziierten höheren Teilmodelle und Abbildung 51 die Entwicklung der Threads über die Zeit. Bereits etwa ab der Hälfte wird die maximale Anzahl der verfügbaren Threads bis zum Ende des Interpretationsprozesses voll ausgeschöpft. Auch die Laufzeit des Experiments erhöht sich erheblich, von 1,5 auf ca. 31 Minuten, liegt damit aber noch deutlich unter der Realzeit von 81 Minuten.

Erfreulicherweise befinden sich die Interpretationen ohne Clutter und ohne halluzinierte Evidenzen auf Rang eins (mit `Refuelling`) und zwei (ohne `Refuelling`). Auf Rang drei und vier befinden sich Interpretationen mit halluzinierten Evidenzen bei `Pushback-Arrival` und `Refuelling` (markiert durch „H“); auch sie enthalten keinen Clutter. Interpretationen mit halluzinierten Elementen werden automatisch dadurch niedriger bewertet, dass sie mehr Clutter-Elemente enthalten. In die BCH wird bei halluzinierten Elementen als Näherung jeweils der Mittelwert $(t_{i\,min} + t_{i\,max}) / 2$ für einen Zeitpunkt T_i eingespeist (s. auch Abschnitt 5.2.3).

100 Clutter-Elemente, aktivierte Halluzination				
Threads (MAX_NUMBER_OF_THREADS = 200): insgesamt: 6186, aktiv (max): 259, abgeschlossen: 1234, gestorben: 4915				
Vollständige Turnaround-Instanzen: 84				
Laufzeit: 31,03 Minuten				
Rang	1	2	3	4
Präferenzmaß	-343,78	-347,75	-350,46	-350,58
Arrival	x	x	x	x
Passenger-Boarding-Prep	x	x	x	x
Unloading-AFT	x	x	x	x
Loading-AFT	x	x	x	x
Unloading-FWD				
Loading-FWD				
Refuelling	x			H
Pushback-Arrival	x	x	H	x
Passenger-Bridge-Leaves-Z	x	x	x	x
Departure	x	x	x	x

Tabelle 11: Interpretationsergebnis mit 100 Clutter-Elementen und aktivierter Halluzination von Evidenzen.

Teilmodell	**Anzahl**
Unloading-AFT	21
Loading-AFT	13
Unloading-FWD	8
Loading-FWD	4
Unloading-And-Loading	274
Refuelling	7
Pushback-Arrival	7
Service-Without-CR	158
Service-With-R	617
Turnaround	84

Tabelle 12: Anzahl der instanziierten Teilmodelle.

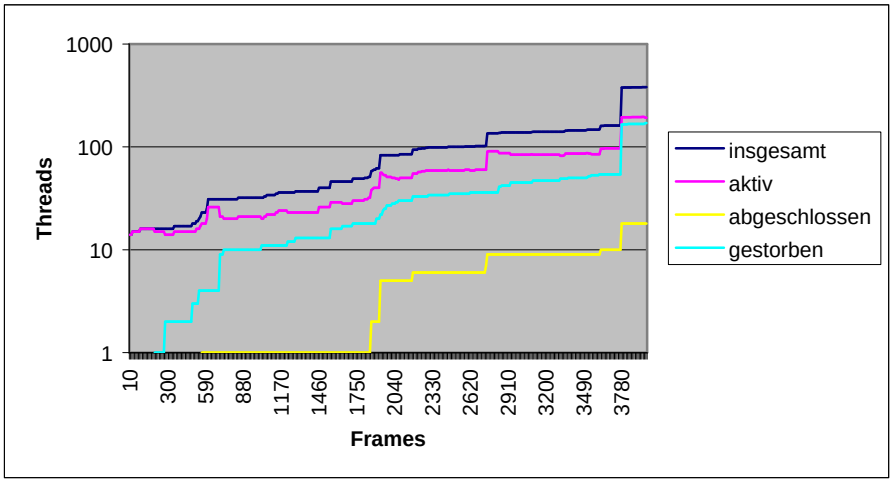


Abbildung 51: Entwicklung der Threads in Abhängigkeit der Frames.

7.1.6 Experimente mit Realdaten

Für die Experimente mit realen Daten stehen 20 annotierte Datensätze (mit der Bezeichnung cof_xy) für eine Evaluierung zur Verfügung. In Anhang C.3 ist eine Statistik aufgeführt, die zeigt, wie oft die für eine vollständige Abfertigung benötigten 23 Evidenzen – bei der gegenwärtigen Ontologie – in den jeweiligen Datensätzen vorkommen. Sie ist ein Beleg für die unzureichende Qualität der primitiven Evidenzen, wie sie von der niederen Bildverarbeitung und der Mittelschicht geliefert werden. Jede Spalte enthält mindestens eine Null, d.h., ohne Halluzination würde keine vollständige Abfertigung erkannt werden. Des Weiteren zeigt sie, dass bestimmte Evidenzen grundsätzlich nur in sehr geringer Anzahl vorhanden sind, beispielsweise `Passenger-Bridge-Positioning/Positioned/-Leaves-PBB-Zone`, während andere übermäßig häufig auftreten, wie z.B. `Aircraft-Positioned/Leaves-Arrival-Zone`. Diese Statistik wird auch für die Berechnung der Clutter-Wahrscheinlichkeitsdichten $\pm_i$ herangezogen. Die Anzahl der Clutter-Elemente in den Datensätzen ist sehr unterschiedlich: sie reicht von 29 bis 216, wenn davon ausgegangen wird, dass jeweils 23 korrekte Evidenzen enthalten sind. Eigentlich sollten für die Experimente und die Statistiken unterschiedliche Datensätze benutzt werden, doch aufgrund der geringen Anzahl verfügbarer Daten ist dies nicht möglich. Daher werden die berechneten Clutter-Wahrscheinlichkeitsdichten $\pm_i$ zufällig um +/- 10% variiert. Alle Versuchsreihen werden im Konsolenmodus (`CONSOLE_MODE = true`, s. Anlage B.1) durchgeführt, in welchem SCENIOR ohne Benutzungsoberfläche arbeitet. Dadurch ist eine wesentlich performantere Verarbeitung möglich. Wie bereits in Abschnitt 7.1.3 beschrieben, werden die Typinformationen der Agenten in den folgenden Experimenten mit realen Daten ignoriert.

7.1.6.1 Versuchsreihe mit unveränderten Datensätzen

Für diese Versuchsreihe werden die Datensätze in der Form verwendet, wie sie von der niederen Bildverarbeitung und der Mittelschicht der Co-Friend-Plattform (s. Abschnitt 7.1.2) erzeugt wurden (abgesehen von der Formatierung). Die maximale Anzahl der Threads `MAX_NUMBER_OF_THREADS` wird auf 150 gesetzt. In der Ontologie werden alle Ereignisse als `is-hallucinatable` markiert. Bei den Zuständen wird zusätzlich das Halluzinieren von `Passenger-Bridge-Positioned` (ist nur in vier Datensätzen enthalten) und `Pushback-Positioned` erlaubt, da sie nicht in allen Datensätzen vorkommen und für eine vollständige Abfertigung obligatorisch sind.

In Tabelle 13 sind die Interpretationsergebnisse dieser Versuchsreihe aufgeführt. Die Zahlen in der zweiten Zeile geben die Anzahl der gefunden `Turnaround`-Instanzen an. In 18 der 20 prozessierten Datensätze wurden mehrere vollständige Abfertigungen gefunden. Die Laufzeit der gesamten Verarbeitung betrug lediglich 1,6 Stunden; sie liegt damit weit unterhalb der Realzeit. Der gelbe Teil der Tabelle zeigt an, ob sich die erkannten Zeitintervalle mit den annotierten Intervallen überschneiden (1) oder nicht (0). Ein „F“ markiert eine falsch-positive Erkennung. Freie Felder zeigen an, dass die entsprechenden Aktionen nicht annotiert sind. Von den insgesamt 138 zu erkennenden Aktionen wurde bei 77 eine Überlappung der entsprechenden Intervalle erzielt.

Für überlappende Intervalle wird zusätzlich mit Hilfe der *normierten Kreuzkorrelation* (s. [3]) ein Maß für die Ähnlichkeit der gefundenen mit den annotierten Intervallen berechnet:

$$korr(A_i) = \frac{\min\{A\_e, N\_e\} - \max\{A\_s, N\_s\}}{\sqrt{(A\_e - A\_s)\cdot(N\_e - N\_s)}}, \tag{7-7}$$

wobei A_s und A_e den Start- und Endzeitpunkt der erkannten Aktion bezeichnen und N_s und N_e den Start- und Endzeitpunkt der annotierten Aktion. Bei punktuellen Ereignissen wird hierbei eine Zeitdauer von einer Minute gesetzt, um zu vermeiden, dass der Nenner null wird. Falls der Start- oder Endzeitpunkt der erkannten Aktion zu einem halluzinierten Ereignis oder Zustand gehört und selbst ein Intervall ist, wird näherungsweise der Mittelwert verwendet:

$$A\_s = \frac{(A\_t_{s\,min} + A\_t_{s\,max})}{2}, \quad A\_e = \frac{(A\_t_{e\,min} + A\_t_{e\,max})}{2}. \tag{7-8}$$

Die Korrelationswerte sind im blauen Teil der Tabelle dargestellt. Die kleinen Werte erklären sich z.T. dadurch, dass viele Evidenzen halluziniert werden müs-

sen und deshalb Intervalle entstehen, die sehr viel größer sind als das annotierte Intervall.

In Abbildung 53 sind beispielhaft die Interpretationsergebnisse der jeweils bestbewerteten Alternativen der Datensätze cof9 und cof18 graphisch dargestellt. Der blaue Balken stellt das annotierte Ereignis bzw. den annotierten Zustand dar, darunter ist jeweils das Ergebnis der Interpretation als dunkelgrüner Balken zu sehen. Gelbe und hellgrüne Balken stellen Intervalle dar, die durch Propagierung halluzinierter Evidenzen entstanden sind. Für den Startpunkt T_s und den Endzeitpunkt T_e ist $t_{s\min}$ bis $t_{s\max}$ in gelb, $t_{s\max}$ bis $t_{e\min}$ in dunkelgrün und $t_{e\min}$ bis $t_{e\max}$ in hellgrün dargestellt.

Eine genauere Analyse ergab, dass im Datensatz cof20 keine passende Kombination von `GPU-Positioned` und `Aircraft-Positioned` enthalten ist, welche die zeitlichen Constraints erfüllt. Die Erklärung, dass in cof65 keine Interpretation gefunden wurde, ist komplizierter. Zunächst ist festzustellen, dass der Datensatz relativ viel Clutter enthält. Dennoch werden mehrere vollständige Instanzen von `Services` gefunden, welche auch in diversen `Turnaround`-Teilmodellen zugeordnet werden können, jedoch sterben die entsprechenden Alternativen sofort wieder, da die maximale Anzahl verfügbarerer Threads ständig überschritten wird (s. Abbildung 52), und die Bewertungen aufgrund der hohen Clutter-Dichte oftmals niedriger sind, als bei den Alternativen ohne zugeordnete `Services`-Evidenz (s. Formel 7-6).

Die in Abbildung 52 dargestellte Thread-Statistik ist typisch für alle durchgeführten Experimente. Das Limit von 150 Threads wird gewöhnlich etwa in der zweiten Hälfte der Szene erreicht. Die auffälligen Stufen bei Frame 3670 und 5540 ergeben sich dadurch, dass jeweils eine Evidenz für `Passenger-Bridge-Leaves-PBB-Zone` dazu führt, dass viele `Services`-Aggregate vervollständigt werden, die dann wiederum als höhere Evidenzen zu einer Menge von teilinstanziierten `Turnaround`-Aggregaten zugeordnet werden können. Durch die Kombinatorik entstehen dabei sehr viele neue Threads.

In Tabelle 13 ist auffällig, dass falsch-positive Erkennungen nur bei `Unloading-Loading-FWD` auftreten. Sie können jedoch auch nur dort oder bei `Refuelling` auftreten, denn diese Aktionen sind optional für eine vollständige Abfertigung. Die Aktion `Unloading-Loading-AFT` ist in allen Annotationen vorhanden, daher ergibt sich dort und in den übrigen Aktionen ggf. eine „0“ und kein „F“.

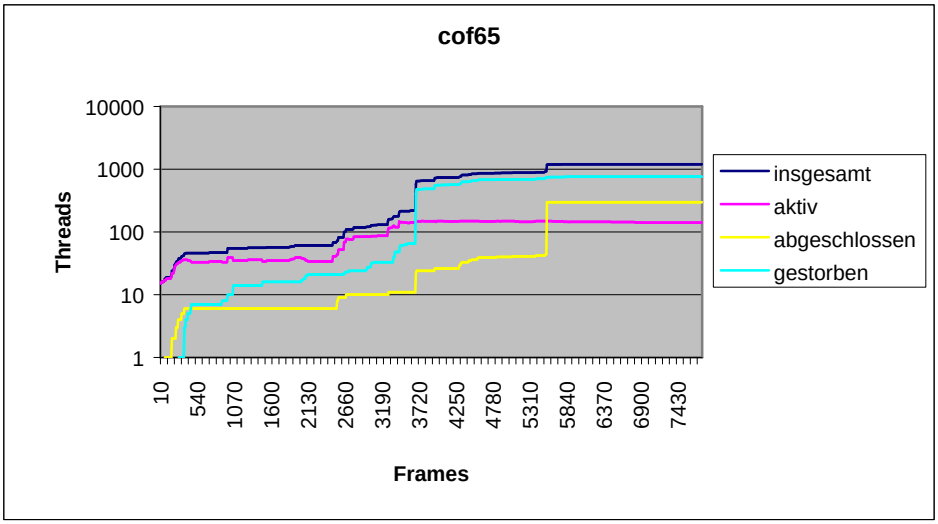
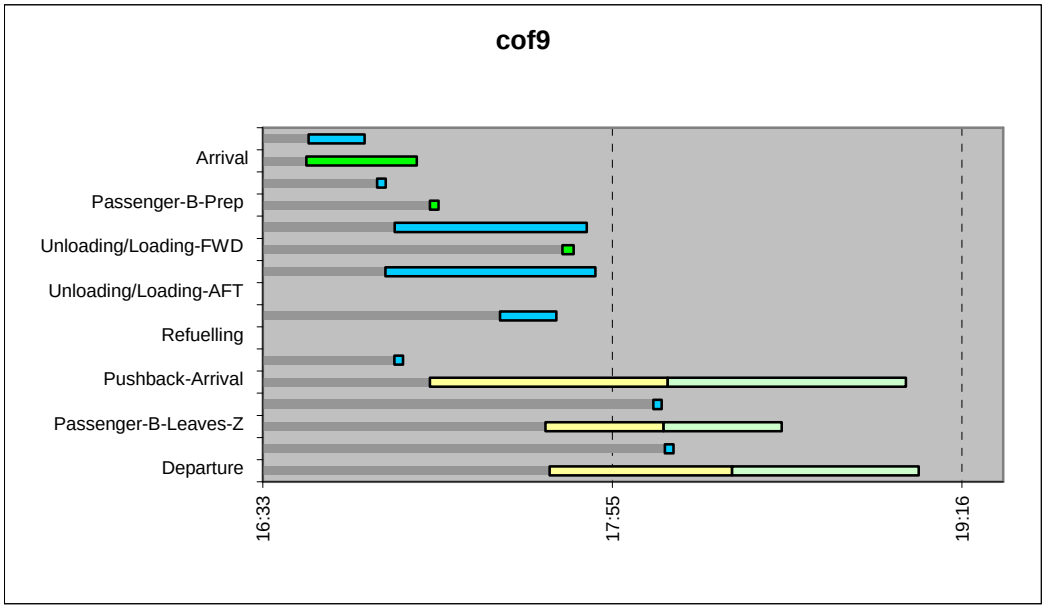


Abbildung 52: Entwicklung der Threads in Abhängigkeit der Frames.

Datensatz	**cof 01**	**cof 02**	**cof 03**	**cof 04**	**cof 05**	**cof 06**	**cof 08**	**cof 09**	**cof 16**	**cof 18**	**cof 20**	**cof 25**	**cof 29**	**cof 30**	**cof 58**	**cof 59**	**cof 62**	**cof 63**	**cof 65**	**cof 66**
#Turnaround	**12**	**46**	**7**	**29**	**18**	**11**	**53**	**40**	**15**	**24**	**0**	**43**	**42**	**28**	**36**	**26**	**29**	**60**	**0**	**29**
Arrival	1	1	1	1	1	0	1	1	0	1	0	1	1	1	1	1	1	1	0	1
Passenger-B-Prep	0	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	0	1
Unloading/Loading-FWD	F		F	F		1	0	1	F		0	F	F		0	1	F	0	0	1
Unloading/Loading-AFT	0	1	0	0	1	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0
Refuelling			0	0	0			0	0			0			0		0		0	
Pushback-Arrival	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	0
Passenger-B-Leaves-Z	1	1	0	1	0	1	0	1	1	1	0	1	1	1	1	1	1	0	0	0
Departure	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1
Arrival	0,85	0,81	0,46	0,79	0,76		0,74	0,71		0,96		0,89	0,64	0,67	0,38	0,98	0,59	0,44		0,72
Passenger-B-Prep		0,12	0,13	0,12	0,13	0,12	0,13		0,12	0,22		0,12	0,16	0,12	0,12	0,11	0,12	0,12		0,12
Unloading/Loading-FWD						0,06		0,19								0,15				0,15
Unloading/Loading-AFT		0,06			0,04		0,18			0,20				0,11	0,28					
Refuelling																				
Pushback-Arrival	0,16								0,10					0,10		0,10	0,10			
Passenger-B-Leaves-Z	0,30	0,20		0,21		0,20		0,19	0,16	0,21		0,20	0,16	0,16	0,20	0,15	0,17			
Departure	0,20	0,16	0,17	0,16	0,22	0,16	0,18	0,15	0,14	0,17		0,16	0,15	0,20	0,16	0,14	0,15			0,16

Tabelle 13: Interpretationsergebnisse der unveränderten Datensätze.



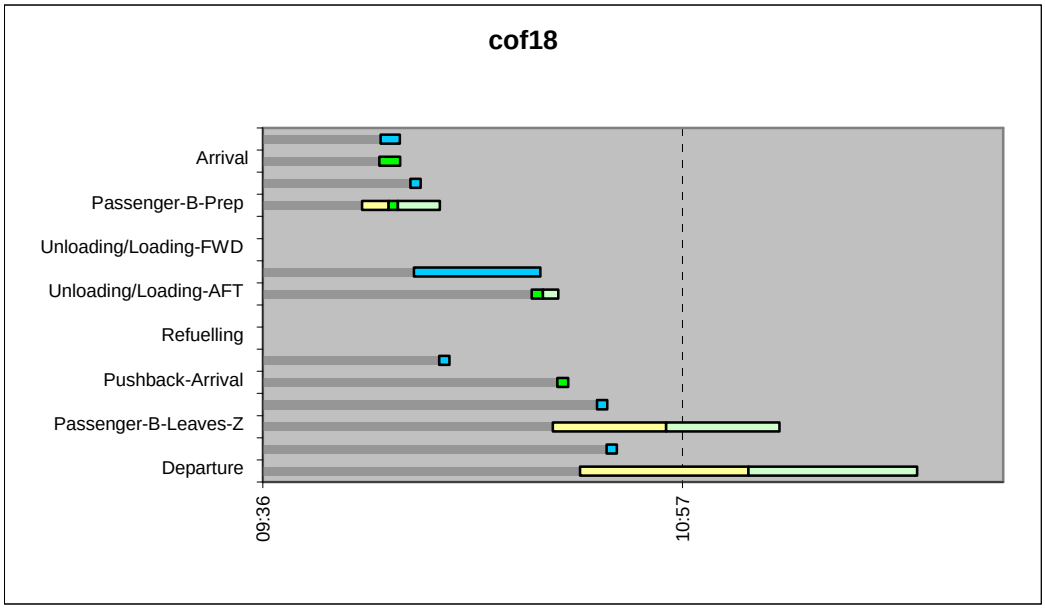


Abbildung 53: Interpretationsergebnisse der Datensätze cof9 und cof18.

7.1.6.2 Versuchsreihe mit Vorverarbeitungsschritt „Langzeit-Tracking“

Um bessere Ergebnisse zu erzielen, wird in dieser Versuchsreihe ein Vorverarbeitungsschritt durchgeführt, der ein *Langzeit-Tracking* simuliert. Bei genauerer Analyse der Datensätze hat sich gezeigt, dass es viele „zerstückelte“ Evidenzen für Zustände gibt, welche denselben Agenten (also dieselbe ID) und dieselbe Zone besitzen. Für das Konzept `Vehicle-Inside-Zone` bzw. `Vehicle-Positioned` könnte dies natürlich auch dadurch entstehen, dass ein Fahrzeug wiederholt aus der Zone heraus- und wieder in die Zone hereinfährt. Doch sind dafür die zeitlichen Unterbrechungen oftmals zu kurz, und es fehlen die entsprechenden Evidenzen für `Vehicle-Enters-Zone` und `Vehicle-Leaves-Zone`. Daher ist es wahrscheinlich, dass diese Zerstückelungen aufgrund von Verdeckung und anderen typischen Bildverarbeitungsproblemen hervorgerufen wurden. Da jedoch die IDs der Agenten und die Zonen identisch sind, hätte ein entsprechendes Langzeit-Tracking in den niederen Bildverarbeitungskomponenten oder der Mittelschicht bereits stattfinden können. Dies wird hier nun simuliert, indem in einem Vorverarbeitungsschritt die entsprechenden primitiven Evidenzen mit einem zeitlichen Abstand von bis zu einer Minute miteinander verschmolzen werden. Die Tabelle in Anlage C.4 zeigt das Ergebnis dieser Vorverarbeitung. Die durchschnittliche Clutter-Dichte ist dadurch von 107,45 auf 86,45 Elemente gesunken. Die Statistik der primitiven Evidenzen zur Berechnung der Clutter-Wahrscheinlichkeitsdichte wurde entsprechend angepasst.

In Tabelle 14 sind die Interpretationsergebnisse dieser Versuchsreihe dargestellt. Die Laufzeit der gesamten Verarbeitung betrug 1,4 Stunden. Insgesamt ist die Anzahl der sich überlappenden Intervalle von 77 der vorherigen Versuchsreihe auf 81 gestiegen. Der durchschnittliche Korrelationswert ist von 0,27 auf 0,29 gestiegen. Die Anzahl der falsch-positiv erkannten Aktionen hat sich von sieben auf acht erhöht. Für den Datensatz `cof65` wurden nun fünf Interpretationen gefunden. Insgesamt kann also eine leichte Verbesserung in der Qualität der Interpretationen registriert werden, sehr befriedigend ist das Ergebnis jedoch auch mit diesem Vorverarbeitungsschritt nicht.

Besonders ungünstig für diese und die vorherige Versuchsreihe ist die hohe Anzahl von potentiellen Evidenzen der Konzepte `GPU-Positioned` und `Aircraft-Positioned`, da die Information des Fahrzeugtyps nicht verwendet werden kann (s. Anlage C.3, C.4). Sie führt dazu, dass bereits zu Beginn der Szene sehr viele teilinstanziierte Threads des Models `Turnaround` generiert werden. Dadurch müssen zum Zeitpunkt der ersten gelieferten `Services`-Instanzen an die `Turnaround`-Threads viele Alternativen aussortiert werden, da das Limit der verfügbaren Threads aufgrund der Kombinatorik überschritten wird. Diese Situation soll durch Einfügen künstlicher Specific Vision Tasks (SVTs) in der nächsten Versuchsreihe verbessert werden.

Datensatz	cof 01	cof 02	cof 03	cof 04	cof 05	cof 06	cof 08	cof 09	cof 16	cof 18	cof 20	cof 25	cof 29	cof 30	cof 58	cof 59	cof 62	cof 63	cof 65	cof 66
#Turnaround	18	42	52	24	15	10	43	52	17	32	0	43	29	20	39	40	40	36	5	33
Arrival	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1
Passenger-B-Prep	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	1	1	0	1
Unloading/Loading-FWD		F	F	F		1	0	1	F		0	F	F	F	0	1	F	0	0	1
Unloading/Loading-AFT	1	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0
Refuelling			0	0	0			0	0			0			0		0		1	
Pushback-Arrival	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0
Passenger-B-Leaves-Z	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	0	0	0
Departure	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	1
Arrival	0,97	0,63	0,73	0,77	0,76		0,74	0,84	0,24	0,96		0,89	0,60	0,67	0,40	0,98	0,59	0,90	0,25	0,72
Passenger-B-Prep	0,12	0,12	0,12	0,12	0,13	0,12	0,13		0,13	0,12		0,12	0,17	0,12	0,12	0,11	0,11	0,12		0,12
Unloading/Loading-FWD						0,06		0,92								0,15				0,15
Unloading/Loading-AFT	0,17				0,04		0,18			0,20					0,28				0,06	
Refuelling																			0,99	
Pushback-Arrival	0,10							0,12								0,10	0,10			
Passenger-B-Leaves-Z	0,15	0,19	0,20	0,21		0,20		0,17	0,24	0,20		0,20	0,18	0,18	0,20	0,15	0,16			
Departure	0,14	0,15	0,16	0,16	0,22	0,16	0,18	0,14	0,18	0,17		0,16	0,17	0,17	0,16	0,14	0,15			0,16

Tabelle 14: Interpretationsergebnisse mit Vorverarbeitungsschritt Langzeit-Trecking.

7.1.6.3 Versuchsreihe mit Langzeit-Tracking und simulierten SVTs

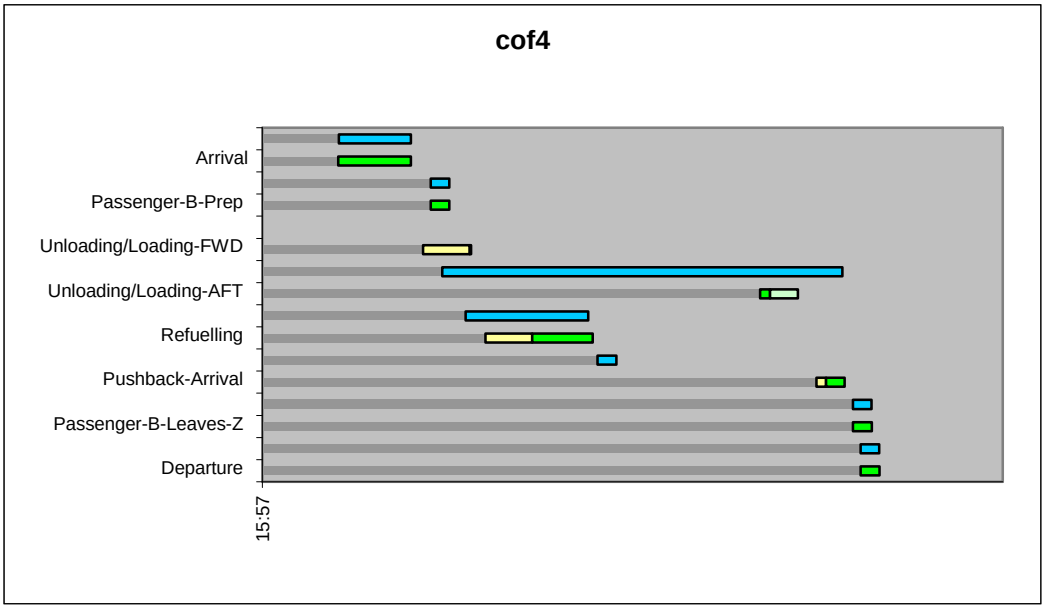
Bis zum Ende des Projekts standen keine Datensätze zur Verfügung, welche die in Abschnitt 7.1.2 beschriebenen Specific Vision Tasks (SVTs) enthalten. Daher werden den Datensätzen in dieser Versuchsreihe künstlich SVTs hinzugefügt. Die Tabellen in Anlage C.3 bzw. C.4 zeigen, welche Konzepte dafür besonders in Frage kommen: zum Einen `Aircraft-Positioning/Positioned/Leaves-Arrival-Zone`, denn entsprechende Evidenzen finden sich besonders häufig in den Datensätzen. Zum Anderen `Passenger-Bridge-Positioning /Positioned/Leaves-PBB-Zone`, denn deren Evidenzen treten nur sehr selten auf. Es waren noch weitere SVTs geplant (z.B. für `Refuelling`), doch die Datensätze sollen hier möglichst geringfügig modifiziert werden. Auf der Basis der Datensätze der vorherigen Versuchsreihe werden nun zunächst alle primitiven Evidenzen der sechs aufgeführten Konzepte entfernt. Dann wird jeweils eine Evidenz für jedes Konzept hinzugefügt, welche der Annotation perfekt entspricht. Das Halluzinieren wird für `Aircraft-Positioning/Leaves-Arrival-Zone` und `Passenger-Bridge-Positioning/Leaves-PBB-Zone` deaktiviert.

In Tabelle 15 sind die Interpretationsergebnisse dieser Versuchsreihe dargestellt. Die Laufzeit der Verarbeitung betrug insgesamt 1,1 Stunden. Die Anzahl der sich überlappenden Intervalle ist von 81 auf 96 gestiegen. Dabei handelt es sich nicht nur um die hinzugefügten SVTs; es werden nun 13 `Unloading-Loading-AFT`-Aktionen erkannt (vorher sechs) und drei `Refuelling`-Aktionen (vorher keine). Der durchschnittliche Korrelationswert ist von 0,29 auf 0,84 gestiegen – in erster Linie natürlich durch die SVTs, die einen Korrelationswert von eins bekommen. In Abbildung 54 sind beispielhaft die Interpretationsergebnisse der Datensätze cof4 und cof65 graphisch dargestellt.

Bei den Ergebnissen in Tabelle 15 fällt auf, dass für den Datensatz cof63 nun keine Interpretation mehr gefunden wird (der Datensatz cof20 wurde bereits diskutiert). Dies hat folgenden Grund: Die Abfertigung ist mit 2,23 Stunden ungewöhnlich lang. Die Zeitdauer vom Ende des Ent- und Beladens des Gepäcks (`Unloading-Loading`) bis zum Ereignis `Passenger-Bridge-Leaves-PBB-Zone` darf laut Modell höchsten 21 Minuten betragen. Dieses Constraint wird für alle gefunden `Unloading-Loading`-Aktionen verletzt, d.h., die SVT-Evidenz `Passenger-Bridge-Leaves-PBB-Zone` kommt zu spät und kann nicht für einen günstigeren Zeitpunkt halluziniert werden, wie in den vorherigen Versuchsreihen. Deshalb wird kein `Services`-Aggregat vollständig instanziiert.

Datensatz	cof 01	cof 02	cof 03	cof 04	cof 05	cof 06	cof 08	cof 09	cof 16	cof 18	cof 20	cof 25	cof 29	cof 30	cof 58	cof 59	cof 62	cof 63	cof 65	cof 66
#Turnaround	42	26	25	34	18	25	30	27	25	29	0	27	26	24	31	31	44	0	21	15
Arrival	1	1	1	1	1	1	1	1	0	1		1	1	0	1	1	1		1	1
Passenger-B-Prep	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1		1	1
Unloading/Loading-FWD	F	F	F	F		0	1	1	F						1	0	F		0	0
Unloading/Loading-AFT	0	0	0	1	1	1	0	1	1	1		1	1	1	1	1	0		1	1
Refuelling	F		0	1	1			0	0			0			0		0		1	F
Pushback-Arrival	1	0	1	0	1	0	0	1	1	0		0	0	0	1	0	1		0	0
Passenger-B-Leaves-Z	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1		1	1
Departure	1	1	1	1	1	1	1	1	1	1		1	1	1	1	1	1		1	1
Arrival	0,99	0,96	0,81	1,00	1,00	0,90	0,96	0,98		0,97		0,98	0,74		0,87	0,99	0,98		1,00	0,99
Passenger-B-Prep	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00
Unloading/Loading-FWD							0,12	0,01							0,10					
Unloading/Loading-AFT				0,18	0,11	0,09		0,18	0,12	0,20		0,21	0,55	0,10	0,28	0,17			0,20	0,18
Refuelling				0,78	0,32														0,83	
Pushback-Arrival	1,00		1,00		1,00			1,00	1,00						1,00		1,00			
Passenger-B-Leaves-Z	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00
Departure	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00	1,00	1,00	1,00	1,00		1,00	1,00

Tabelle 15: Interpretationsergebnisse mit simulierten Specific Vision Tasks.



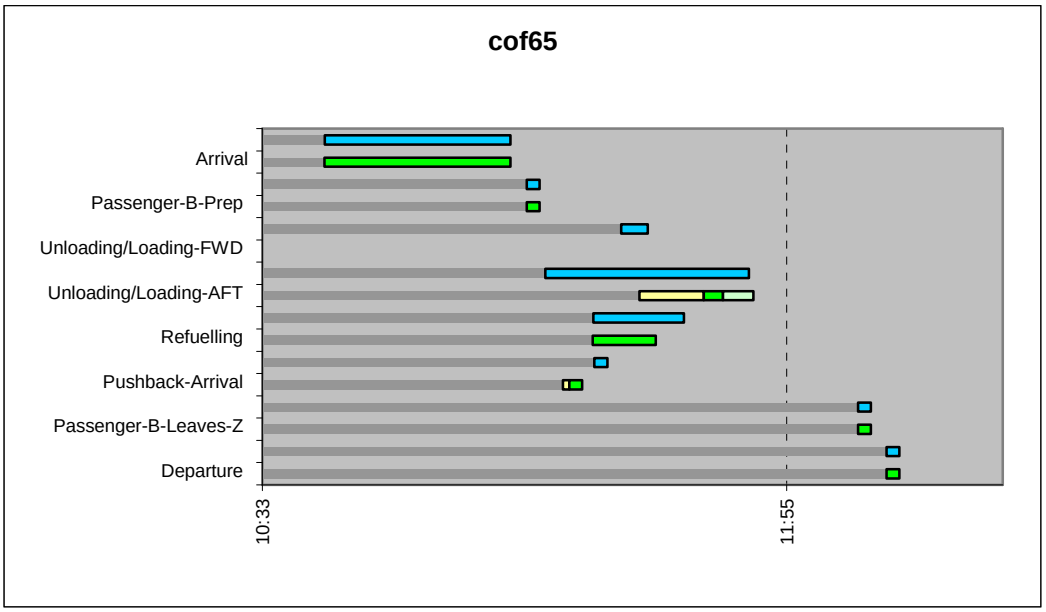


Abbildung 54: Interpretationsergebnisse der Datensätze cof4 und cof65.

Die in Abbildung 55 dargestellte Thread-Entwicklung des Datensatzes cof65 zeigt, dass durch das Einfügen der SVTs (und Löschen der entsprechenden Clutter-Elemente) das Limit der verfügbaren Threads sehr viel später erreicht wird – hier bei Frame 7120 statt bei Frame 3430 in der ersten Versuchsreihe (vgl. Abbildung 52). Dadurch wird verhindert, dass möglicherweise gute teilinstanziierte Alternativen von `Services` oder anderer Teilmodelle aufgrund der hohen Clutter-Dichte aussortiert werden, bevor sie eine vollständige `Turnaround`-Instanz bilden können. So ist es zu erklären, dass das Hinzufügen der SVTs die Ergebnisse insgesamt verbessert.

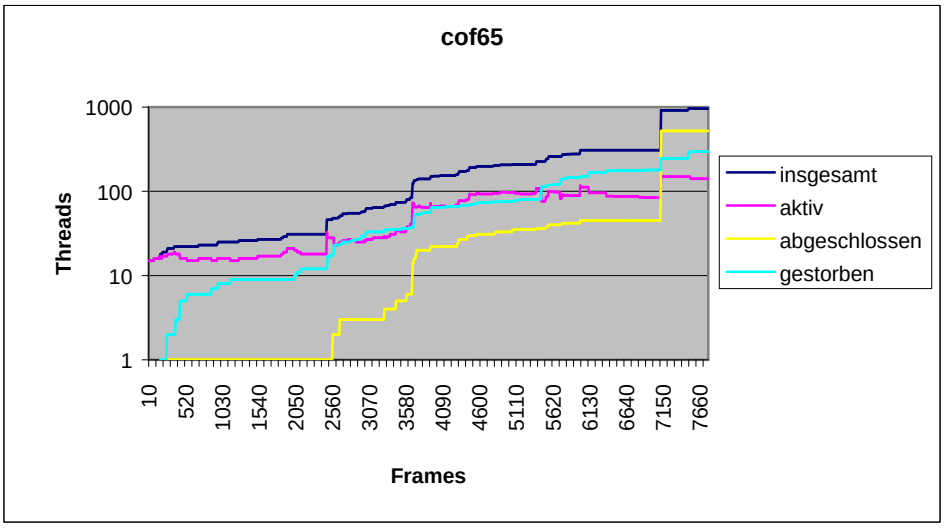


Abbildung 55: Entwicklung der Threads in Abhängigkeit der Frames.

Interpretationen auf niedrigeren Rängen

Bisher wurden jeweils nur die bestbewerteten Interpretationen jedes Durchlaufs betrachtet. Es werden jedoch in der Regel immer mehrere alternative Interpretationen für jeden Datensatz gefunden; in dieser Versuchsreihe zwischen 15 und 44. Deshalb werden nachfolgend für jeden Datensatz jeweils die Interpretationen auf Rang fünf, zehn und 15 ausgewertet. In Tabelle 16 sind die Ergebnisse dargestellt, zum Einen die Anzahl der überlappenden Intervalle, zum Anderen der durchschnittliche Korrelationswert. Dabei kann die Tendenz festgestellt werden, dass sich auf den höheren Rängen erwartungsgemäß auch die qualitativ besseren Interpretationen befinden.

Rang	**1**	**5**	**10**	**15**
Anzahl überlappender Intervalle	96	96	90	93
Durchschnittlicher Korrelationswert	0,84	0,79	0,82	0,80

Tabelle 16: Interpretationsergebnisse der Alternativen auf Rang 1, 5, 10 und 15.

Abhängigkeit von der Anzahl der maximalen Threads

In dieser Versuchsreihe wird das Interpretationsergebnis in Abhängigkeit der maximal zur Verfügung stehenden Threads untersucht. In Tabelle 17 sind die Interpretationsergebnisse dargestellt. Interessanter Weise zeigt sich bei 200 maximalen Threads durchschnittlich keine Verbesserung der Ergebnisse mehr gegenüber 150 Threads (eine Versuchsreihe mit `MAX_NUMBER_OF_THREADS = 250` lief nicht erfolgreich durch, da die Ressourcen des Rechners ausgeschöpft waren). Hier reicht offenbar eine Strahlbreite von 150 aus, um das optimale Ergebnis zu finden. Eine mögliche Erklärung ist, dass der Lösungsraum wenig oder keine lokalen Maxima hat, mit anderen Worten: gut bewertete Instanzen $\{a_1,...,a_k\}$ von Teilmodellen auf unterer Ebene führen auch zu gut bewerteten Instanzen b_i, welche $\{a_1,...,a_k\}$ als Teile haben. Dadurch wird der prinzipielle Nachteil der Strahlsuche relativiert, möglicherweise nur ein lokales Optimum zu finden.

`MAX_NUMBER_OF_THREADS`	**100**	**150**	**200**
Laufzeit (Stunden)	0,67	1,1	2,0
Anzahl überlappender Intervalle	92	96	96
Durchschnittlicher Korrelationswert	0,83	0,84	0,84

Tabelle 17: Interpretationsergebnisse in Abhängigkeit der maximal zur Verfügung stehenden Threads.

Anzahl der generierten Threads ohne Begrenzung der Strahlbreite

Abschließend wurden die Interpretationsergebnisse des Datensatzes `cof1` genauer untersucht, um zu klären, wie viele aktive Threads ohne Begrenzung der Strahlbreite entstehen würden. Dazu wurden zunächst nur die `Services`-Teilmodelle (`MAIN_HYPOTHESIS = Services`, s. Anlage B.1) analysiert. Dabei zeigt sich, dass 57 Instanzen von `Services-Without-CR` und sechs von `Refuelling` generiert werden. Diese kombinieren sich zu 342 `Services-With-R`-Instanzen,

bei maximal 200 gleichzeitig aktiven Threads, d.h., es wurden keine Alternativen verworfen. Ein Durchlauf mit `MAIN_HYPOTHESIS = Turnaround` zeigt, dass es fünf teilinstanziierte `Turnaround`-Instanzen gibt (und durch das `Notification`-Konzept werden auch keinen Neuen mehr generiert), wenn die ersten höheren `Services`-Evidenzen zur Verfügung stehen. Unter der Voraussetzung, dass alle Constraints erfüllt sind, ergibt sich dadurch eine Kombination von $5 \cdot (57 + 342) = 1995$ Threads, d.h., bei einem Interpretationsprozess mit `MAX_NUMBER_OF_THREADS = 1195` würden keine Alternativen verworfen werden. Erfreulicherweise setzt sich die bestbewertete Alternative mit `MAIN_HYPOTHESIS = Services` aus denselben primitiven Evidenzen zusammen, wie bei der besten Interpretation der obigen Versuchsreihe mit einem Maximum von 150 Threads. Zusammenfassend kann daraus der Schluss gezogen werden, dass die Dimensionierung der Strahlsuche der Domäne und der Datenqualität gut angepasst ist.

7.2 Domäne 2: Intelligente Wohnumgebungen

Ein weiteres Anwendungsgebiet für die Szeneninterpretation ist die Realisierung intelligenter Haus- und Wohnungsumgebungen (engl.: *smart home environment*, s. Abschnitt 1.2.2). In diesem Abschnitt soll anhand des *CASAS Smart-Home-Projekts* [2] und des *Living-Place-Projekts* [5] der HAW[20] Hamburg gezeigt werden, dass die in dieser Arbeit entwickelten Verfahren und das Interpretationssystem SCENIOR auch für andere Domänen anwendbar sind.

7.2.1 CASAS Smart-Home-Projekt

Das CASAS Smart-Home-Projekt [2] ist ein multidisziplinäres Forschungsprojekt der Washington State University zur Entwicklung intelligenter Wohnumgebungen. Dabei wird der Ansatz verfolgt, eine Wohnung als intelligenten Agenten zu betrachten, der in der Lage ist, das Verhalten seiner Bewohner mit Hilfe von unterschiedlichen Sensoren zu erkennen und darauf zu reagieren. Eine Anwendung ist

[20] Hochschule für Angewandte Wissenschaften.

die Beobachtung älterer Menschen, um mögliche Unfälle zu detektieren und bei Bedarf Hilfe herbeizurufen. Eine detailliertere Beschreibung des Projekts findet sich in [87].

7.2.1.1 Eingabedaten

Auf der CASAS Homepage [2] findet sich eine Vielzahl von Datensätzen. Für die Generierung dieser Datensätze wurden Mitwirkende angewiesen, bestimmte Aufgaben auszuführen. Diese umfassten u.a.:

- Befüllen eines Medikamentendosierers,
- Fernsehen,
- Pflanzen gießen,
- Telefonieren,
- Geburtstagskarte schreiben,
- Suppe zubereiten,
- Reinigen,
- Kleidung auswählen.

Die Datensätze bestehen aus Zustandsbeschreibungen von verschiedenen Sensoren, welche in einem experimentellen Apartment installiert sind. Diese Sensoren können in sieben Kategorien eingeteilt werden:

- Bewegungssensoren, welche die Bewegung von Bewohnern detektieren,
- Gegenstandssensoren, welche die An- bzw. Abwesenheit eines Gegenstands detektieren,
- Türsensoren, die detektieren, ob eine Tür geöffnet oder geschlossen ist,
- Telefonsensoren, die detektieren, ob eine Telefonat beginnt oder beendet wird,
- Wasserdrucksensoren, die den Wasserdruck verschiedener Wasserhähne messen,
- Hitzesensoren, welche z.B. die Temperatur der Herdplatten messen.

In Abbildung 56 ist eine Sensorenkarte der experimentellen Wohnumgebung dargestellt; ein „M“ bezeichnet Bewegungssensoren, die übrigen Sensoren befinden sich im Bereich der Türen und der rot markierten Bereiche. Die Sensoren der ersten vier Kategorien sind binär und können direkt durch entsprechende konzeptuelle Objekte ausgedrückt werden, beispielsweise `Open(Agent, Zone)` und `Close(Agent, Zone)` für das Ereignis, dass eine Person eine Tür öffnet bzw. schließt oder `Near(Agent, Zone)`, für den Zustand, dass sich eine Person in der Nähe eine bestimmte Zone befindet, ausgelöst durch den entsprechenden Bewegungssensor.

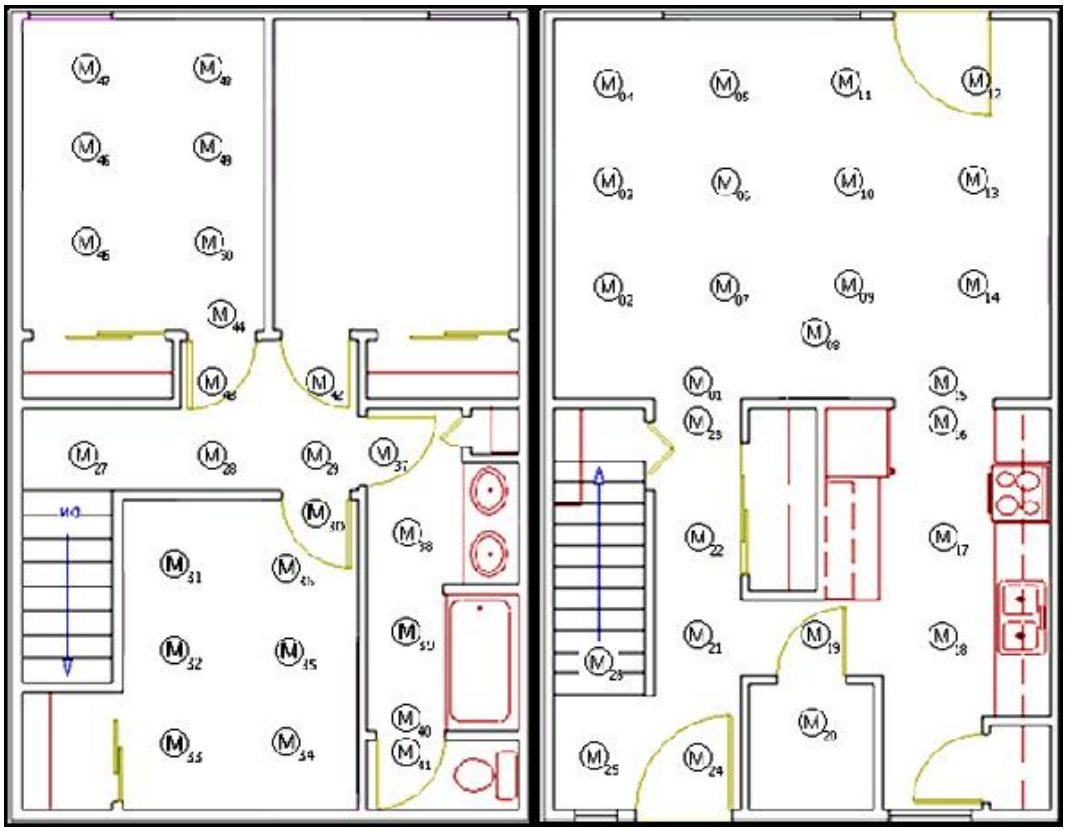


Abbildung 56: Sensorenkarte der experimentellen Wohnumgebung des Projekts CASAS (aus [87]).

Wasserdruck- und Hitzesensoren liefern Realwerte; diese können von SCENIOR momentan nicht direkt verarbeitet werden. Doch durch eine entsprechende Erweiterung des Systems könnte dies leicht realisiert werden, schließlich werden zeitliche Constraints bereits quantitativ repräsentiert und verarbeitet.

In einem Vorverarbeitungsschritt wurden die Daten zunächst in Evidenzen von Ereignissen und Zuständen überführt, welche von SCENIOR verarbeitet werden können:

original CASAS-Daten: (7-9)

```
06:06:03.04    M034 ON
06:06:06.17    M034 OFF
```

generierte Evidenz für SCENIOR:

```
06:06:03.04 - 06:06:06.17: Near(Person 114, M034)
```

Die auszuführenden Aufgaben wurden jeweils separat mit einer Person detektiert. Für die Vorverarbeitung wurden diese separaten Datensätze verwendet, mit jeweils derselben ID für die Person. Anschließend wurden die primitiven Evidenzen mehrerer Datensätze in beliebiger Reihenfolge vermischt. Der daraus entstandene Datensatz, der in diesem Beispiel verwendet wird, enthält insgesamt 211 primitive Evidenzen, wovon zehn für die hier betrachtete Aktionen „Suppe zubereiten“ benötigt werden.

7.2.1.2 Ontologie

Als Ontologie wird die bisherige Top-Level-Ontologie (s. Abschnitt 5.1.2) verwendet, die um das Konzept `Four-Sequence` $\sqsubseteq$ `Composite-Event` erweitert wurde. Analog zu `Two-Sequence` und `Three-Sequence` beschreibt es ein abstraktes Handlungsmuster, bei dem vier Ereignisse oder Zustände nacheinander stattfinden (s. auch Regel 5-6). Die in diesem Beispiel zu erkennende Aktion – das Zubereiten einer Suppe – wird durch das Aggregat `Prepare-Soup` $\sqsubseteq$ `Four-Sequence` repräsentiert. Die Aktion besteht aus dem Herausnehmen eines Artikels aus dem Küchenschrank, dem Hineinstellen eines Tellers in die Mikrowelle und dem anschließenden Herausnehmen (lediglich durch den Türsensor detektiert) und schließlich dem Zurückstellen des Artikels (beispielsweise einer Schachtel) in den Küchenschrank. Dabei müssen eine Reihe von Identitäts-Constraints erfüllt sein, beispielsweise, dass an allen Teilaktionen dieselbe Person beteiligt ist. Die kompositionelle Struktur ist in Abbildung 57 dargestellt, die SWRL-Regel für das Aggregat `Prepare-Soup` in Regel 7-10 (die zeitlichen Constraints sind in der SWRL-Regel für `Four-Sequence` definiert). Da es in dieser Domäne kein einzelnes Interpretationsziel gibt – wie das Aggregat `Turnaround` bei den Flughafenvorfeldaktivitäten – werden die zu erkennenden

Aktionen als Unterkonzepte des Domänenkonzepts `Activity` definiert, dieses wird durch `MAIN_HYPOTHESIS` = `Activity` als Dachmodell deklariert.

```
Prepare-Soup
     Take-Soup-Ingredients
          Open-Kitchen-Cupboard
          TakeI01
          Close-Kitchen-Cupboard
     Access-Microwave-Put
          Open-Microwave
          Close-Microwave
     Access-Microwave-Get
          Open-Microwave
          Close-Microwave
     Return-Soup-Ingredients
          Open-Kitchen-Cupboard
          ReturnI01
          Close-Kitchen-Cupboard
```

Abbildung 57: Kompositionelle Struktur der Aktion `Prepare-Soup`.

```
Prepare-Soup(?ps) ^                                                    (7-10)
has-part-A(?ps, ?a) ^
has-part-B(?ps, ?b) ^
has-part-C(?ps, ?c) ^
has-part-D(?ps, ?d) ^


has-agent(?a, ?a1) ^
has-agent(?b, ?a2) ^
has-agent(?c, ?a3) ^
has-agent(?d, ?a4) ^
has-location(?a, ?l1) ^
has-location(?b, ?l2) ^
has-location(?c, ?l3) ^
has-location(?d, ?l4) ^


has-part-B(?a, ?take) ^
has-part-B(?d, ?return) ^
has-location(?take, ?l5) ^
has-location(?return, ?l6)
→
equal(?a1, ?a2) ^
equal(?a2, ?a3) ^
equal(?a3, ?a4) ^
equal(?l1, ?l4) ^
equal(?l2, ?l3) ^
equal(?l5, ?l6)
```

7.2.1.3 Experiment ohne Teilmodelle

Da für diese Domäne keine probabilistischen Modelle zur Verfügung stehen, wird die BCH für das Experiment deaktiviert. Außerdem wird das Halluzinieren von Evidenzen deaktiviert. Die maximale Anzahl verfügbarer Threads wird auf 100 gesetzt. Die Ontologie enthält nur das Aggregat `Prepare-Soup` und seine Teile (gemäß Abbildung 57) als ein zusammenhängendes Modell, d.h., es gibt keine durch `is-context-free` markierten Teilmodelle.

Der Interpretationsprozess findet eine vollständige Instanziierung des Aggregats `Prepare-Soup`, welche in Abbildung 58 dargestellt ist. Insgesamt werden 105 Threads generiert, davon sterben 103, es gibt einen abgeschlossenen Thread und denjenigen ohne eine Zuweisung, welcher den Status *aktiv* behält. Die maximale Anzahl gleichzeitig aktiver Threads beträgt 32. Hier zeigt sich, dass für die Interpretation der Evidenzen, die auf weitgehend rauschfreien Sensordaten basieren, keine Halluzinierung von Evidenzen erforderlich ist und dass der Suchbaum relativ begrenzt bleibt.

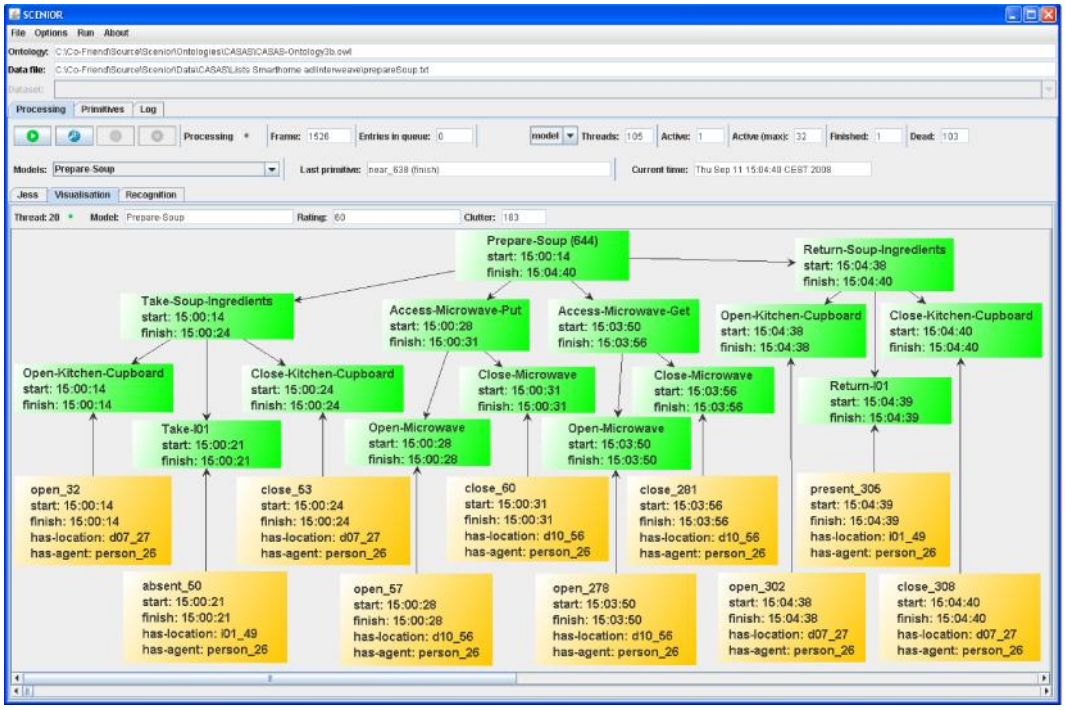


Abbildung 58: Interpretationsergebnis des CASAS-Experiments.

7.2.1.4 Experiment mit Teilmodellen

Die Domäne der intelligenten Wohnumgebungen zeichnet sich durch eine hohe Variationsvielfalt der Aktionen aus, d.h., in der Regel können bestimmte Abläufe auf unterschiedliche Arten ausgeführt werden. In der obigen Szene könnte beispielsweise der Artikel in den Schrank zurückgestellt werden, während die Mikrowelle eingeschaltet ist. In dieser Domäne sind also die Abläufe schwächer strukturiert als in der Domäne der Flugzeugvorfeldaktivitäten. Dies unterstreicht die Notwendigkeit Teilmodelle zu verwenden. Darüber hinaus gibt es in dieser Domäne eine Vielzahl von Identitäts-Constraints, die zwischen mehreren Teilmodellen bestehen; beispielsweise ist derselbe Bewohner in der Regel an allen Aktionen beteiligt. Daher ist in dieser Domäne die in Abschnitt 5.2.2.4 beschriebene Erweiterung wichtig, welche es erlaubt, Identitäts-Constraints auch zwischen verschiedenen Teilmodellen zu definieren.

Für dieses Experiment werden daher die in Tabelle 18 aufgeführten Teilmodelle definiert und in der Ontologie mit `is-context-free` markiert. Erfreulichweise liefert der Durchlauf das gleiche Endergebnis wie im obigen Experiment (s. Abbildung 59). Hier wurden nur insgesamt 74 Threads generiert, es gibt zwölf abgeschlossene Threads (die Verteilung ist in Tabelle 18 dargestellt), maximal 33 gleichzeitig aktive und 53 gestorbene Threads. Hier zeigt sich, dass die Verwendung von Teilmodellen die Anzahl der generierten Threads deutlich reduziert.

Teilmodell	**Anzahl gefundener Instanzen**
`Prepare-Soup`	1
`Take-Soup-Ingredients`	2
`Return-Soup-Ingredients`	3
`Access-Microwave-Put`	3
`Access-Microwave-Get`	3

Tabelle 18: Teilmodelle und die Anzahl der gefundene Instanzen des Interpretationsprozesses.

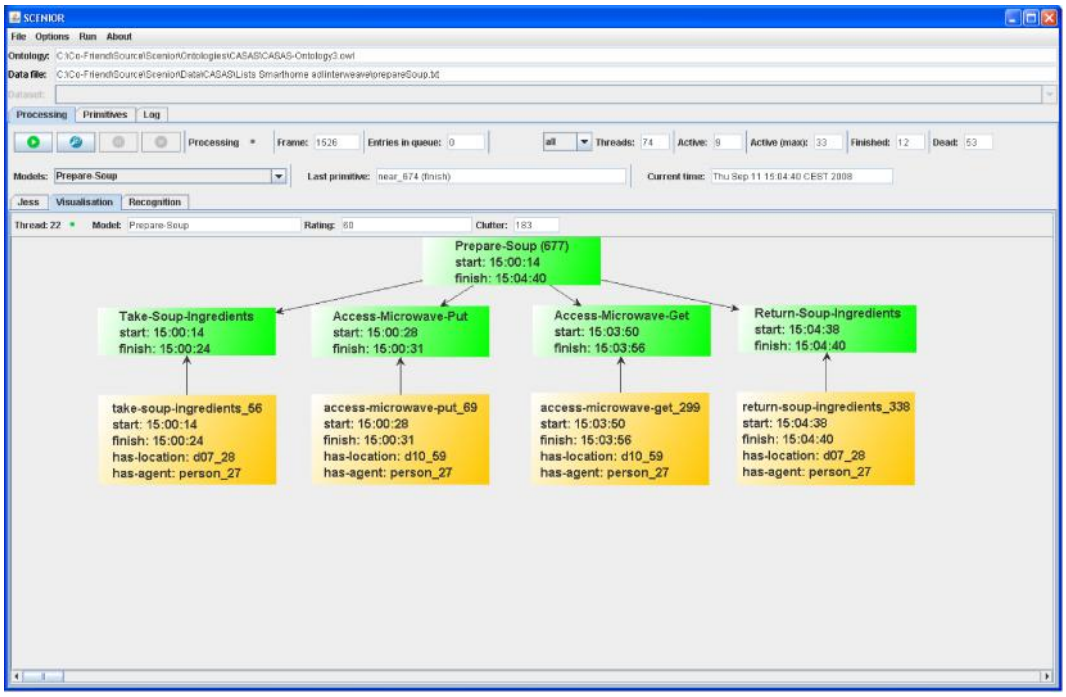


Abbildung 59: Interpretationsergebnis des CASAS-Experiments mit Teilmodellen.

7.2.2 Living-Place-Projekt

Das zweite Beispiel aus der Domäne der intelligenten Wohnumgebungen ist das Living-Place-Projekt [5] der HAW Hamburg, in dem die Möglichkeiten untersucht werden, inwieweit das Leben in einer Wohnung durch intelligente Systeme unterstützt werden kann. Ein 3D-Modell der Living-Place-Umgebung ist in Abbildung 60 dargestellt. Der Wohnbereich ist unterteilt in die Zonen *DiningArea*, *KitchenArea*, *SleepingArea*, *LivingArea* und *Bathroom*.

Der Wohnbereich ist mit verschiedenen Kameras und Sensoren unterschiedlichster Art ausgestattet. Dies sind zum Einen einfache binäre Sensoren, die beispielsweise feststellen, ob eine Tür geöffnet oder geschlossen ist, aber auch kapazitive und resistive Sensoren, welche die Annäherung von Objekten und die Krafteinwirkung auf bestimmte Materialien messen können (s. [105]). Damit wird beispielsweise detektiert, ob sich eine Person auf dem Sofa oder dem Bett befindet. Andere Sensoren registrieren, ob die Dusche oder der Fernseher aktiviert ist. Detaillierte Informationen zur Living-Place-Umgebung finden sich in [5].

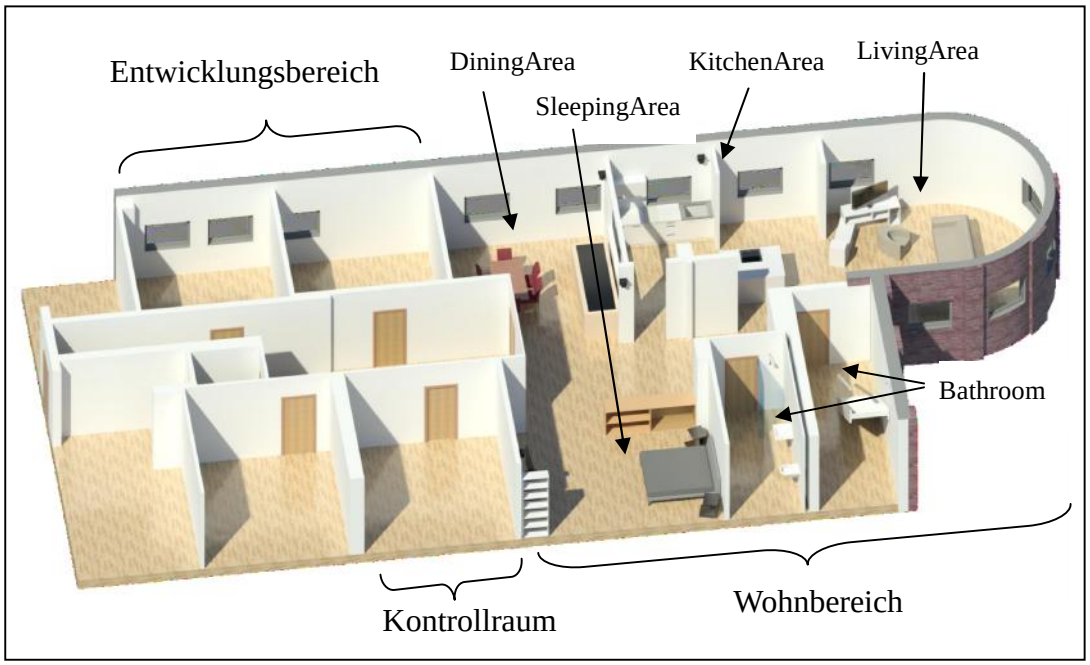


Abbildung 60: 3D-Modell der Living-Place-Umgebung der HAW Hamburg (aus [47]).

7.2.2.1 Living-Place-Plattform

Die zentrale Kommunikationsschnittstalle aller Anwendungen der Living-Place-Umgebung ist das sog. *Blackboard* (s. [47]). Auf dem Blackboard kann jede Anwendung einen Informationskanal erstellen, um Information (Ergebnisse) zu liefern und sich auf Informationskanälen anmelden, um gezielt Informationen zu empfangen. Auch die Ergebnisse der Sensoren, die die Basis für die primitiven Ereignisse und Zustände für SCENIOR darstellen, werden an das Blackboard geliefert. Für eine vollständige Integration von SCENIOR in die Living-Place-Umgebung würde das System diese Informationen über entsprechende Informationskanäle erhalten und sie, in einem noch zu implementierenden Vorverarbeitungsschritt, in geeignete primitive Evidenzen für SCENIOR übersetzen. Abgeleitete höhere Evidenzen könnten dann an das Blackboard zurückgeliefert werden, um dann wiederum als Eingabe für weitere Anwendungen zu dienen und beispielsweise bestimmte Aktionen auszulösen. Dies würde jedoch den Umfang dieser Arbeit übersteigen, daher beschränkt sich das hier dargestellte Beispiel auf eine dateigestütze Offline-Verarbeitung.

7.2.2.2 Eingabedaten

Das nachfolgende Beispiel wird anhand einer manuell erstellen Eingabedatei mit primitiven Evidenzen demonstriert, wie sie bei einer Online-Verarbeitung mit entsprechender Vorverarbeitung generiert werden könnten. Diese Eingabedatei und die im nächsten Abschnitt vorgestellte Ontologie wurde in Zusammenarbeit mit Jens Ellenberg erstellt, der sich im Rahmen seiner Masterarbeit [47] mit ontologiebasierter Aktivitätserkennung in der Living-Place-Umgebung beschäftigt hat. Dort finden sich auch weitere Informationen zur Anbindung von SCENIOR an die Living-Place-Plattform und zu den nötigen Vorverarbeitungsschritten.

Ziel des hier vorgestellten Beispiels ist es – in sehr vereinfachter Form – die typischen Aktivitäten eines Bewohners im Laufe eines Vormittags zu erkennen, repräsentiert durch das Aggregat `ResidentActionMorningActivity`, bestehend aus den Teilaktivitäten

- Aufstehen,
- Duschen,
- Frühstücken.

Die bisherige fixe Festlegung der primitiven Evidenzen auf die Eigenschaften `has-agent` und `has-location`, die sich im Rahmen des Co-Friend-Projekts etablierte, erwies sich in dieser Domäne als zu unflexibel. Beispielsweise gibt es eine Reihe von Objekten, die aktiviert oder deaktiviert werden können, und Aggregate, welche die Eigenschaft haben, einen Agenten, eine Zone und ein Objekt zu besitzen. Es ist offensichtlich, dass eine allgemeinere Definition auch für andere Domänen nützlich wäre. Deshalb wurde SCENIOR dahingehend erweitert, dass eine primitive Evidenz beliebig viele physikalische Objekte als Argumente enthalten kann, die mit dem entsprechenden Rollennamen angegeben werden (über den Konfigurationsschalter `NEW_PARSER` wird die Verwendung dieses neuen Eingabeformats festgelegt, s. Anlage B.1), z.B. kann durch

```
08:00:40.00 - 08:00:40.00: DeactivateBed(has-object: Bed 2)
```
(7-11)

das Ereignis ausgedrückt werden, dass das Bett mit der ID `Bed 2` „deaktiviert“ wurde, mit anderen Worten, sich niemand mehr darin befindet.

7.2.2.3 Ontologie

Wie bereits angedeutet, zeichnet sich die Domäne der intelligenten Wohnumgebungen durch eine hohe Variationsvielfalt der zu erkennenden Aktionen aus. Ein Beispiel dafür ist das Ereignis `ResidentActionStartBreakfast`, welches den Beginn des Frühstücks repräsentiert und die Ereignisse `ActiveCutlery` mit Zeitstempel t_1 (repräsentiert, dass das Besteck bereit liegt) und `ResidentEnterFunctionalSpaceKitchen` mit Zeitstempel t_2 (repräsentiert, dass der Bewohner die Küche betritt) als Teile hat. Nun ist es wünschenswert, den Zeitpunkt für `ResidentActionStartBreakfast` mit $max(t_1,t_2)$ zu definieren, da für die Teile keine feste Reihenfolge vorgegeben ist. Dies ist jedoch mit der konvexen Zeitpunktalgebra nicht möglich (s. Abschnitt 5.2.3). Deshalb wurden für derartige Fälle jeweils zwei Unterkonzepte definiert:

$$\begin{aligned} &\texttt{ResidentActionStartBreakfastObjectZone} \sqsubseteq \texttt{ResidentActionStartBreakfast} \\ &\texttt{ResidentActionStartBreakfastZoneObject} \sqsubseteq \texttt{ResidentActionStartBreakfast} \end{aligned} \tag{7-12}$$

Das obere Konzept repräsentiert, dass zuerst das Besteck bereit liegt und dann der Bewohner die Küche betritt, das untere entsprechend umgekehrt. Für beide Konzepte ist jeweils eine SWRL-Regel definiert, die den Zeitpunkt des Ereignisses `ResidentActionStartBreakfast` entsprechend festlegt. In Abbildung 61 ist eine mögliche kompositionelle Struktur des Aggregats `ResidentActionMorningActivity` dargestellt. Eine übersicht über die Taxonomie findet sich in Anlage A.2.

```
ResidentActionMorningActivity
    ResidentActionEndSleepLeavingZone
        ResidentLeaveFunctionalSpaceBed
    ResidentActionTakingShower
        ResidentActionStartTakingShowerObjectZone
            ActivateShower
            ResidentEnterTraditionalSpaceBathroom
        ResidentActionEndTakingShowerLeavingZone
            ResidentLeaveTraditionalSpaceBathroom
    ResidentActionBreakfast
        ResidentActionStartBreakfastZoneObject
            ActivateCutlery
            ResidentEnterFunctionalSpaceKitchen
        ResidentActionEndBreakfastDeactivateObject
            DeactivateCutlery
```

Abbildung 61: Mögliche kompositionelle Struktur des Aggregats `ResidentActionMorningActivity`.

7.2.2.4 Experiment

Der für dieses Experiment verwendete Eingabedatensatz ist in Anlage C.2 aufgeführt. Auch für dieses Experiment wurde die BCH deaktiviert, da keine probabilistischen Modelle vorliegen.

Insgesamt wurden 73 Threads generiert, davon 46 maximal gleichzeitig aktive. Am Ende des Interpretationsprozesses sind 20 Threads gestorben und 20 vollständig instanziiert; diese sind in Tabelle 19 aufgeführt. Es werden acht Instanzen von `ResidentActionMorningActivity` gefunden, eine davon ist in Abbildung 62 dargestellt. Dies erklärt sich folgendermaßen: das Aggregat `ResidentActionMorningActivity` setzt sich aus den Teilen `ResidentActionEndSleep`, `ResidentActionTakingShower` und `ResidentActionBreakfast` zusammen. Jede dieser Aktionen kann auf zwei Arten beendet werden: durch das Verlassen der entsprechenden Zone oder durch die Deaktivierung eines bestimmten Objekts. Daher ergeben sich für `ResidentActionMorningActivity` $2^3 = 8$ mögliche Instanziierungen, da jeweils beide Evidenzen im verwendeten Datensatz vorliegen. Hier wäre eine Modellierung wünschenswert, die jeweils nur eine Möglichkeit instanziiert; wenn also z.B. `ResidentActionTakingShowerEndObject` instanziiert ist, sollte unter bestimmten Bedingungen die Instanziierung von `ResidentActionTakingShowerEndZone` unterbunden werden. Ein derartiger Mechanismus ist bisher nicht realisiert.

Zusammenfassend kann gesagt werden, dass die Experimente mit den Domänen der intelligenten Wohnumgebungen vielversprechende erste Ergebnisse zeigen. Die neuen Ontologien und die Anpassungen an die Eingabedaten konnten mit relativ geringem Aufwand entwickelt werden. Dies hat gezeigt, dass sich SCENIOR für die Anwendung ganz unterschiedlicher Domänen eignet.

Teilmodell	Anzahl gefundener Instanzen
ResidentActionMorningActivity	8
ResidentActionBreakfastEndObject	1
ResidentActionBreakfastEndZone	1
ResidentActionEndBreakfastDeactivateObject	1
ResidentActionEndBreakfastLeavingZone	1
ResidentActionEndSleepDeactivateObject	1
ResidentActionEndSleepLeavingZone	1
ResidentActionEndTakingShowerDeactivateObject	1
ResidentActionEndTakingShowerLeavingZone	1
ResidentActionStartBreakfastObjectZone	0
ResidentActionStartBreakfastZoneObject	1
ResidentActionStartTakingShowerObjectZone	0
ResidentActionStartTakingShowerZoneObject	1
ResidentActionTakingShowerEndObject	1
ResidentActionTakingShowerEndZone	1

Tabelle 19: Teilmodelle des Living-Place-Experiments und die Anzahl der gefundene Instanzen des Interpretationsprozesses.

Abbildung 62: Interpretationsergebnis des Living-Place-Experiments.

8 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, einen generischen Ansatz für die Szeneninterpretation zu entwickeln, unter besonderer Berücksichtigung der Echtzeitfähigkeit. Weiterhin sollte ein Prototyp implementiert werden, um die Leistungsfähigkeit der entwickelten Methoden anhand von Domänen der realen Welt zu demonstrieren und zu testen. Aus den Erkenntnissen bestehender Ansätze wurden folgende Anforderungen an ein derartiges generisches Rahmenwerk für die Szeneninterpretation abgeleitet:

- Verwendung standardisierter Wissensrepräsentationsmethoden,
- Statische und dynamische Verarbeitung,
- Echtzeitfähigkeit,
- Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses,
- Verarbeitung unsicherer und unvollständiger Eingabedaten.

Im Folgenden wird betrachtet, inwieweit diese Anforderungen durch den vorgestellten Ansatz und das Szeneninterpretationssystem SCENIOR erfüllt werden.

Verwendung standardisierter Wissensrepräsentationsmethoden. Die Wissensbasis des hier präsentierten Ansatzes ist in der standardisierten Beschreibungssprache OWL DL und ihrer Erweitung SWRL formuliert. Dadurch können verfügbare DL-Reasoner genutzt werden, um verschiedene Konsistenzprüfungen für den OWL-Teil durchzuführen. Reasoner, die auch SWRL-Regeln in die Konsistenzprüfungen mit einbeziehen, befinden sich in der Entwicklung und sind Gegenstand der aktuellen Forschung. Lediglich die probabilistischen Modelle für die BCH sind in einem proprietären Format repräsentiert und nicht Bestandteil der OWL Wissensbasis, da es nach meinem Kenntnisstand keine effiziente Darstellungsform der benötigten Kovarianzmatritzen in OWL gibt. In SCENIOR wird die Konsistenz der Modelle in der OWL Wissensbasis mit den BCH-Modellen in der Initialisierungsphase geprüft. Für die Vermeidung von Redundanz und eine bessere Wartbarkeit der Wissensbasis wäre jedoch eine einheitliche Repräsentation in OWL wünschenswert.

Statische und dynamische Verarbeitung. Der entwickelte Ansatz und das Interpretationssystem SCENIOR sind auf die inkrementelle Verarbeitung dynamischer Eingabedatenströme ausgerichtet. Als ein innovatives Charaktermerkmal kann diesbezüglich der automatische Übersetzungsprozess der OWL Wissensbasis in ein regelbasiertes, datengetriebenes Interpretationssystem angesehen werden. Dadurch konnten die Anforderung nach standardisierten Wissensrepräsentationsmethoden mit denen einer echtzeitfähigen, dynamischen Datenverarbeitung in Einklang gebracht werden. Prinzipiell lässt sich SCENIOR jedoch auch in der gegenwärtigen Implementierung für die statische Szeneninterpretation nutzen, dies ist in Abschnitt 6.4 näher beschrieben. Für eine optimale Verarbeitung statischer Bilder müsste das System erweitert und angepasst werden.

Echtzeitfähigkeit. Die Echtzeitfähigkeit des Interpretationssystem SCENIOR wurde durch die Verwendung der performanten Regelmaschine Jess, die parallele Verarbeitung mittels Multithreadding und die Etablierung einer probabilistisch gesteuerten Strahlsuche realisiert. Die Möglichkeit, SCENIOR auch ohne Benutzungsoberfläche einsetzen zu können, beschleunigt den Interpretationsprozess zusätzlich. Die Experimente aus Kapitel 7 zeigen, dass der Interpretationsprozess durch den hier vorgestellten Ansatz und die Architektur von SCENIOR deutlich schneller als die Realzeit ist (zumindest für die in dieser Arbeit untersuchten Domänen). In der Regel bilden die Algorithmen der niederen Bildverarbeitung den limitierenden Faktor.

Trennung der Wissensbasis von den Kontrollstrukturen des Interpretationsprozesses. Durch den automatischen Übersetzungsprozess der OWL Wissensbasis in ein regelbasiertes System ist die Wissensbasis strikt von den Kontrollstrukturen des Interpretationsprozesses getrennt. Dies wäre in einem System, welches nur die Regelbasis als Wissensbasis verwenden würde, nicht der Fall. In der OWL Wissensbasis jedoch sind die Modelle in einer rein deklarativen Weise repräsentiert, während sich die Kontrollstrukturen des Interpretationsprozesses durch die automatisch generierten Regeln manifestieren. Da sich die Entwicklung der Ontologie durch einen inkrementellen Prozess mit häufigen Erweiterungen und Modifikation auszeichnet, ist eine einfache Möglichkeit zur Veränderung der Modelle wünschenswert. Dies ist durch die rein deklarative Repräsentation und standardisierte Werkzeuge gewährleistet. Die Kontrollstrukturen für den Interpretationsprozess hingegen sind relativ beständig; sie zu modifizieren würde Änderungen am Programmcode erfordern.

Verarbeitung unsicherer und unvollständiger Eingabedaten. Die anspruchsvolle Anforderung, auch unsichere und unvollständige Eingabedaten verarbeiten zu können, war für zentrale Aspekte des hier entwickelten Ansatzes maßgeblich. Die Parallelverarbeitung und die probabilistisch gesteuerte Strahlsuche beispielsweise dienen dazu, dem inhärenten Problem der Mehrdeutigkeit in der Szeneninterpretation zu begegnen. Durch unsichere und unvollständige Eingabedaten erhöht sich

die Mehrdeutigkeit. Beispielsweise sind in der Domäne der Flughafenvorfeldaktivitäten die Typinformationen der Fahrzeuge derartig unsicher, dass sie nicht genutzt werden können (s. Abschnitt 7.1.3). Das erhöht die Mehrdeutigkeit beträchtlich. Durch die massive Parallelverarbeitung können viele dieser Alternativen in Echtzeit verfolgt werden. Während sich die Szene weiter entwickelt und mehr Kontextinformation entsteht, werden dann Alternativen verworfen, die bestimmte Constraints nicht erfüllen oder durch die BCH niedrig bewertet werden. Unvollständige Eingabedaten können durch den Mechanismus des Halluzinierens von Evidenzen ergänzt werden. Es stellte sich heraus, dass diese Situation in komplexen Anwendungen der realen Welt eher die Regel als die Ausnahme ist. Das Halluzinieren ist beim derzeitigen Stand nur in sehr einfacher Weise realisiert, indem entsprechende Konzepte in der Ontologie markiert werden. Hier ist Raum für weitere Forschungsarbeit, z.B. Inferenztechniken, die auf Allgemeinwissen (engl.: *common sense*) basieren. Weiterhin zeigen die Experimente aus Abschnitt 7.1, dass der hier vorgestellte Ansatz sehr robust gegenüber zufälligem Rauschen ist, da die Constraint-Mechanismen eine gute Filterwirkung haben.

Zusammengefasst kann festgestellt werden, dass der hier präsentierte Ansatz für die echtzeitfähige Szeneninterpretation die Ziele der Arbeit zum großen Teil erfüllt. Das entwickelte Interpretationssystem SCENIOR hat den Entwicklungsstand eines Prototypen überschritten. Die Experimente aus Abschnitt 7.2 haben gezeigt, dass das System mit einem Arbeitsaufwand von ca. ein bis zwei Wochen an eine neue Domäne angepasst werden kann. Einen Großteil der Zeit nimmt dabei die Erstellung der Domänen-Ontologie ein. Die Anpassung des Eingabedatenformats ist eine geringere und übersichtliche Aufgabe.

Grundsätzlich eignet sich der hier entwickelte Ansatz und das Interpretationssystem SCENIOR für komplexe Domänen mit strukturierten Abläufen (bzw. strukturierten Bildern für die statische Verarbeitung), die sich hinreichend gut durch hierarchische Modelle beschreiben lassen. Weiterhin ist es optimiert auf die inkrementelle Verarbeitung von Evidenzen, um den Anforderungen der Echtzeitfähigkeit in vielen Domänen gerecht zu werden. Durch den Mechanismus der Teilmodelle können dabei Variationen in den Abläufen verarbeitet werden. Auch bei Domänen mit unzureichender Qualität der Eingabedaten aus den niederen Bildverarbeitungskomponenten können durch die Parallelverarbeitung und ein probabilistisches Präferenzmaß gute Ergebnisse erzielt werden. Bei sehr hoher Variation (d.h. sehr vielen Teilmodellen und initialen Threads) in Kombination mit sehr geringer Qualität der Eingabedaten, die einen großen Suchbaum erfordert, wird das System sicherlich an seine Grenzen stoßen.

Für die weitere Entwicklung des hier vorgestellten Ansatzes und des Interpretationssystems SCENIOR sind verschiedene Aspekte denkbar:

- Integration neuer Constraint-Mechanismen, beispielsweise zur Verarbeitung räumlicher Constraints.
- Einbeziehung von Common-Sense-Regeln.
- Mehr Flexibilität bei der Modellierung der Ontologie.
- Weiterentwicklung der BCH-Integration.
- Verbesserte Integration der Taxonomie in den Interpretationsprozess.

Integration neuer Constraint-Mechanismen. Für die dynamische Szeneninterpretation sind räumliche und zeitliche Constraints essentiell. Bei dem hier präsentierten Ansatz liegt der Schwerpunkt auf der Verarbeitung zeitlicher Constraints. Räumliche Constraints werden lediglich durch Reifikation räumlicher Relationen verarbeitet, wie z.B. bei dem Konzept `Vehicle-Inside-Zone`. Hier wäre die Integration eines räumlichen Constraint-Mechanismus denkbar, welcher beispielsweise auf qualitativen Kalkülen wie RCC [39] basiert oder auch quantitative Modelle einbezieht (beispielsweise, dass ein Fahrzeug in einer bestimmten Zeit nur eine bestimmte Entfernung zurücklegen kann).

Einbeziehung von Common-Sense-Regeln. Gegenwärtig werden Konzepte, deren Instanzen halluziniert werden dürfen, manuell in der Ontologie markiert. Das Halluzinieren wird durch das Constraint-Netz getriggert. Hier wären sog. Common-Sense-Regeln wünschenswert, welche Inferenztechniken realisieren, die auf Allgemeinwissen und physikalischen Modellen beruhen. Steht beispielsweise ein Fahrzeug in einer Zone, dann muss es vorher hereingefahren sein.

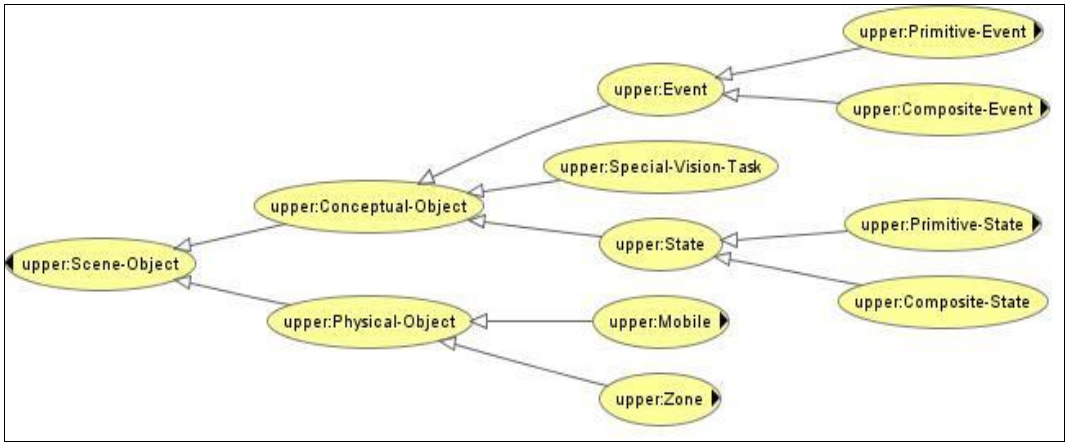
Mehr Flexibilität bei der Modellierung der Ontologie. Bisher unterliegt die Modellierung der Ontologie diversen Restriktionen: Beispielsweise ist keine Mehrfachvererbung erlaubt, ein „oder“ kann nur indirekt über die Definition alternativer Unterkonzepte definiert werden, physikalische Objekte dürfen keine weiteren Objekte als Teile haben oder über andere Eigenschaften verfügen. Hier wäre mehr Flexibilität in der Modellierung wünschenswert. Während bei der Realisierung der letzten beiden Punkte keine prinzipiellen Schwierigkeiten zu erwarten sind, ist die Umsetzung der Mehrfachvererbung ein gravierenderes Problem, da die Regelsprache Jess nur Einfachvererbung bei Templates erlaubt.

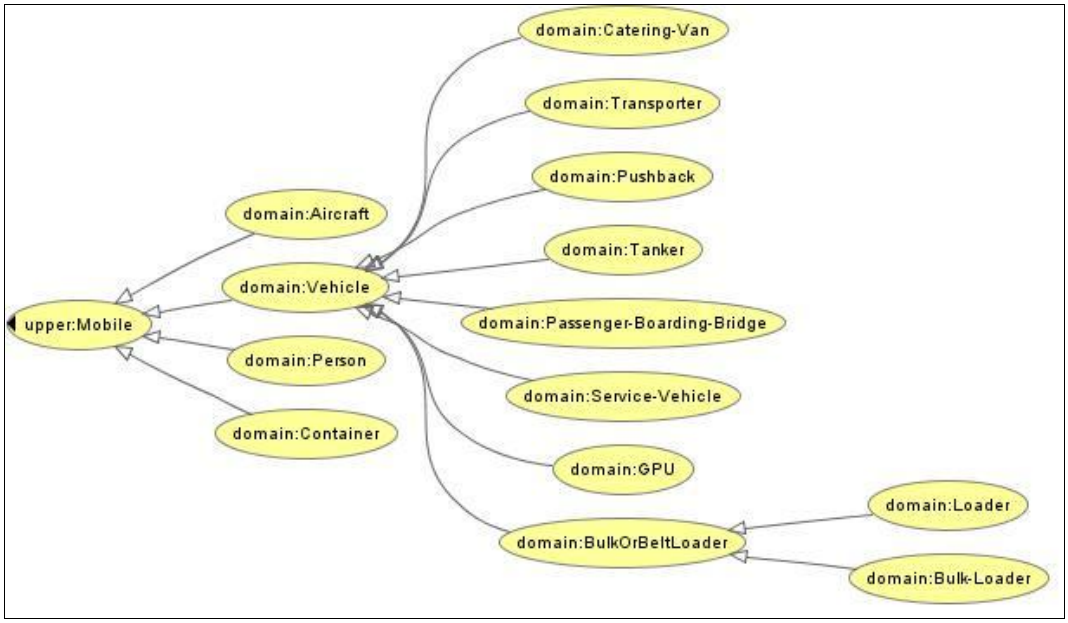
Weiterentwicklung der BCH-Integration. Die Unterteilung der Wissensbasis in Teilmodelle sollte zum gleichen Ergebnis kommen wie ein entsprechendes zusammenhängendes Modell. Dies ist bei der derzeitigen Integration der BCH nur näherungsweise der Fall. Hier ist eine Weiterentwicklung der BCH-Integration dahingehend erforderlich, dass die Bewertung von Teilmodellen bei Verwendung in einem übergeordneten Kontext entsprechend angepasst wird. Für weitere Details sei an dieser Stelle auf [33], [79], [81] und [83] verwiesen.

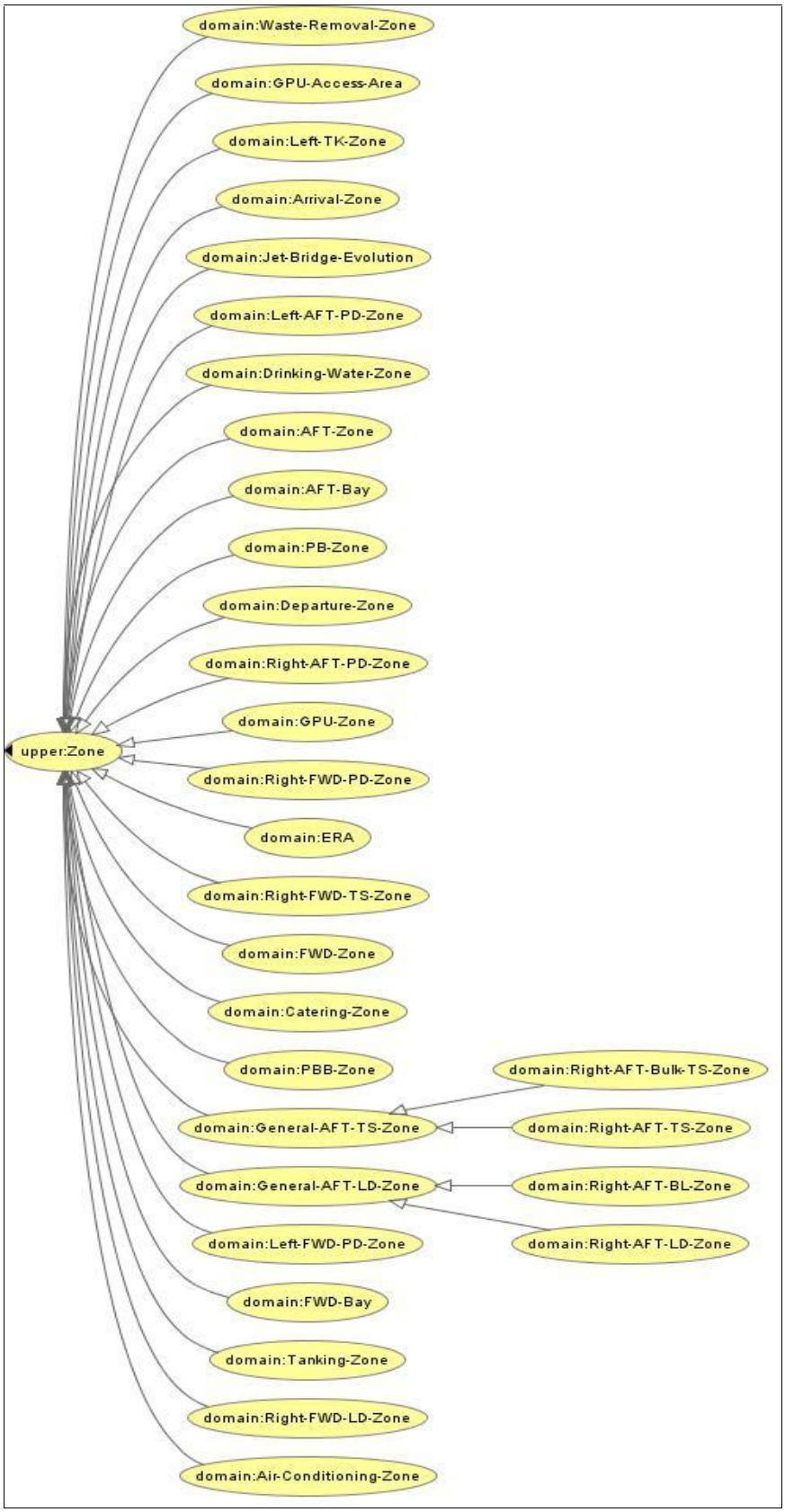
Verbesserte Integration der Taxonomie in den Interpretationsprozess. Die taxonomische Struktur der Ontologie bleibt durch den Übersetzungsprozess in Form der Template-Hierarchie in Jess erhalten. Dadurch können Instanzen durch die gegebenen Jess-Mechanismen beispielsweise zu Superkonzepten im Hypothesengraph zugewiesen werden (z.B. wird eine `Tanker-Inside-Zone`-Instanz zu `Vehicle-Inside-Zone` im Hypothesengraph zugewiesen, wenn die Constraints erfüllt sind). Umgekehrt würde eine `Mobile-Inside-Zone`-Instanz dem Clutter zugewiesen werden (und ist dann verloren), sofern sie nicht direkt spezialisiert werden kann. Eine Spezialisierung zu einem späteren Zeitpunkt, die sich durch mehr Kontextinformationen ergeben könnte, ist nicht möglich. Dies könnte evtl. dadurch erreicht werden, dass erweiterte Hypothesengraphen neben den partonomischen Strukturen auch die taxonomischen Hierarchien repräsentieren würden.

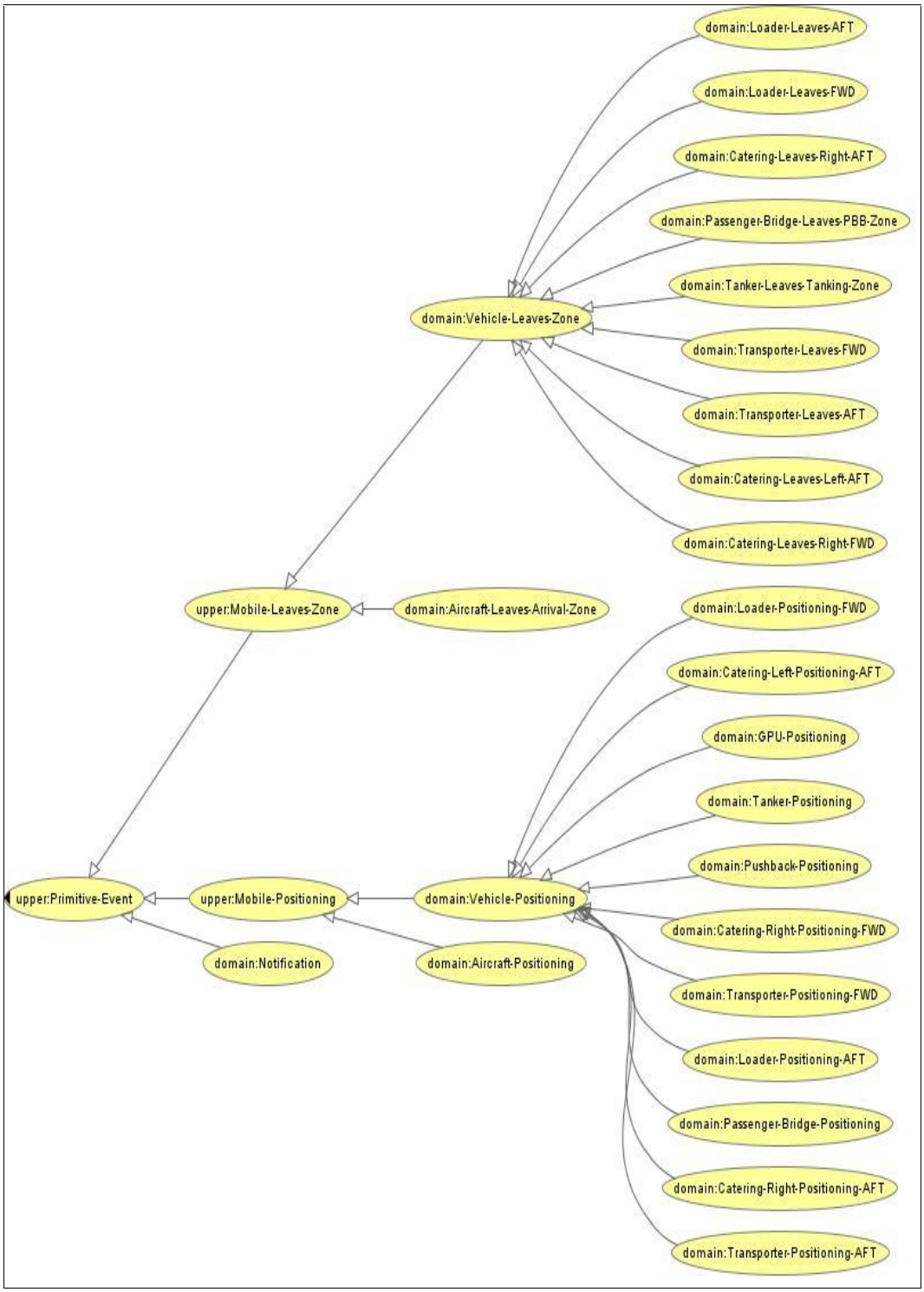
A Anlage: Ontologie

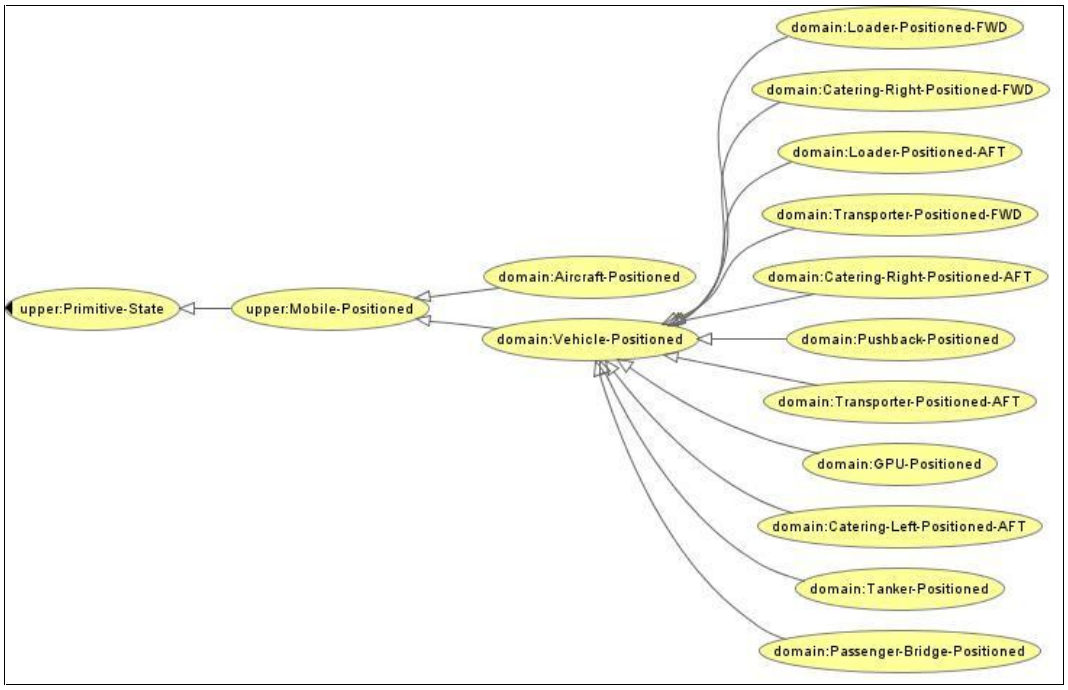
A.1 Taxonomie der Flughafenvorfeld-Domäne

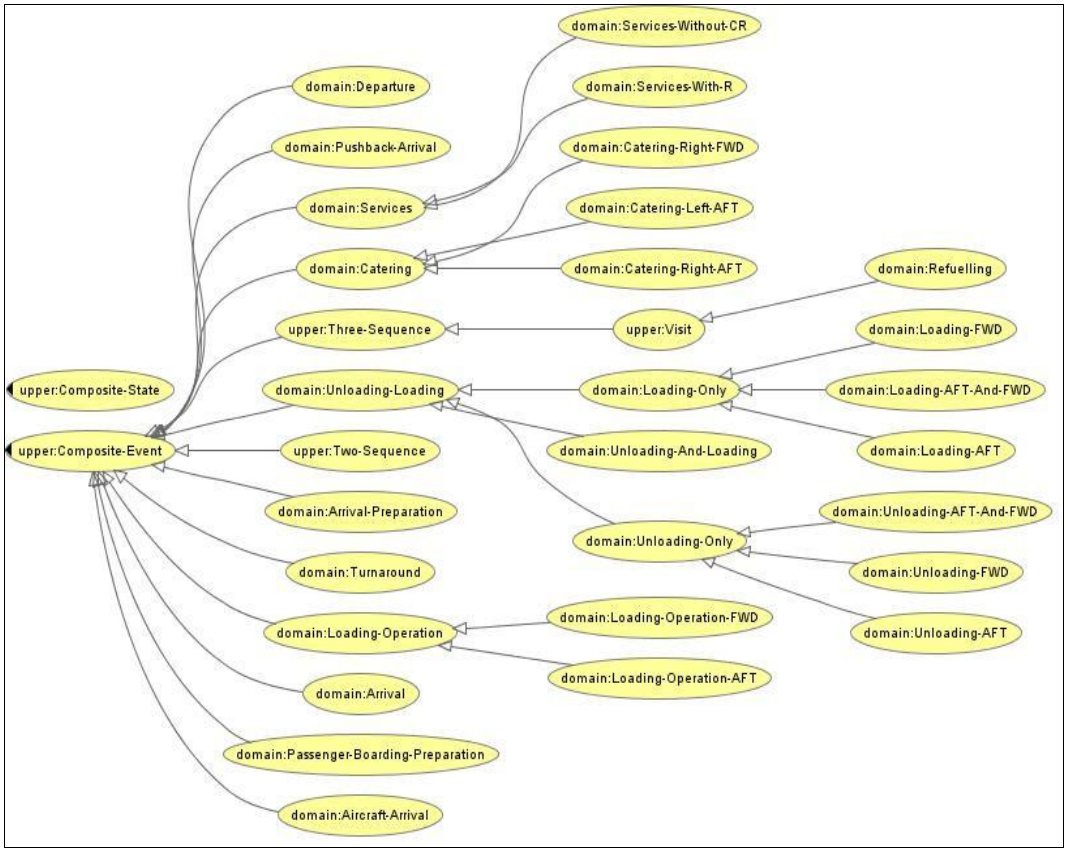




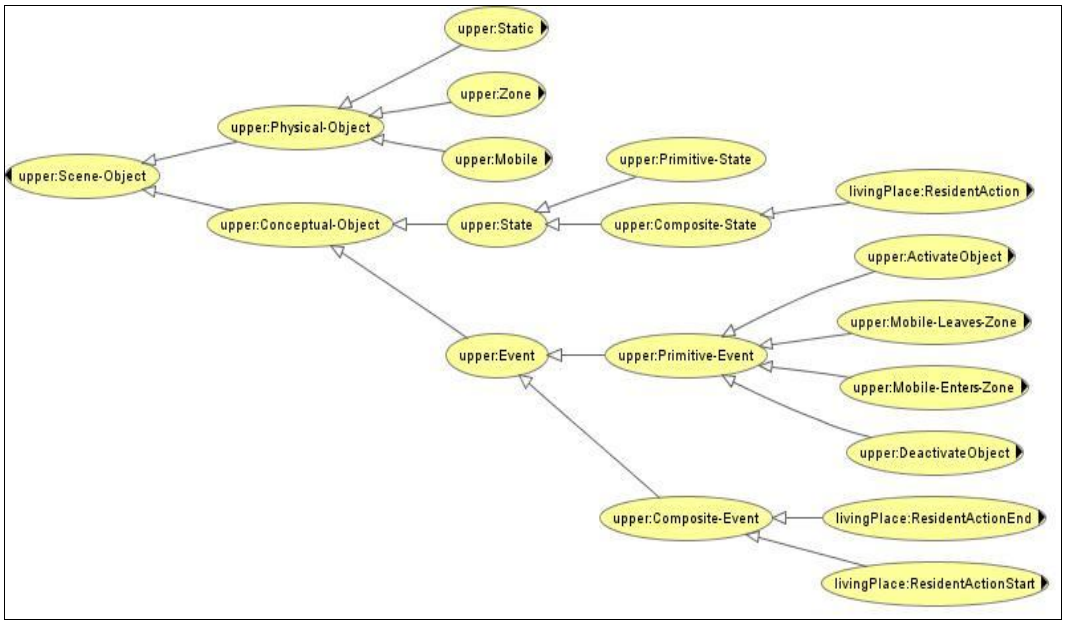

domain:Waste-Removal-Zone
domain:GPU-Access-Area
domain:Left-TK-Zone
domain:Arrival-Zone
domain:Jet-Bridge-Evolution
domain:Left-AFT-PD-Zone
domain:Drinking-Water-Zone
domain:AFT-Zone
domain:AFT-Bay
domain:PB-Zone
domain:Departure-Zone
domain:Right-AFT-PD-Zone
domain:GPU-Zone
upper:Zone
domain:Right-FWD-PD-Zone
domain:ERA
domain:Right-FWD-TS-Zone
domain:FWD-Zone
domain:Catering-Zone
domain:PBB-Zone
domain:Right-AFT-Bulk-TS-Zone
domain:General-AFT-TS-Zone
domain:Right-AFT-TS-Zone
domain:General-AFT-LD-Zone
domain:Right-AFT-BL-Zone
domain:Right-AFT-LD-Zone
domain:Left-FWD-PD-Zone
domain:FWD-Bay
domain:Tanking-Zone
domain:Right-FWD-LD-Zone
domain:Air-Conditioning-Zone

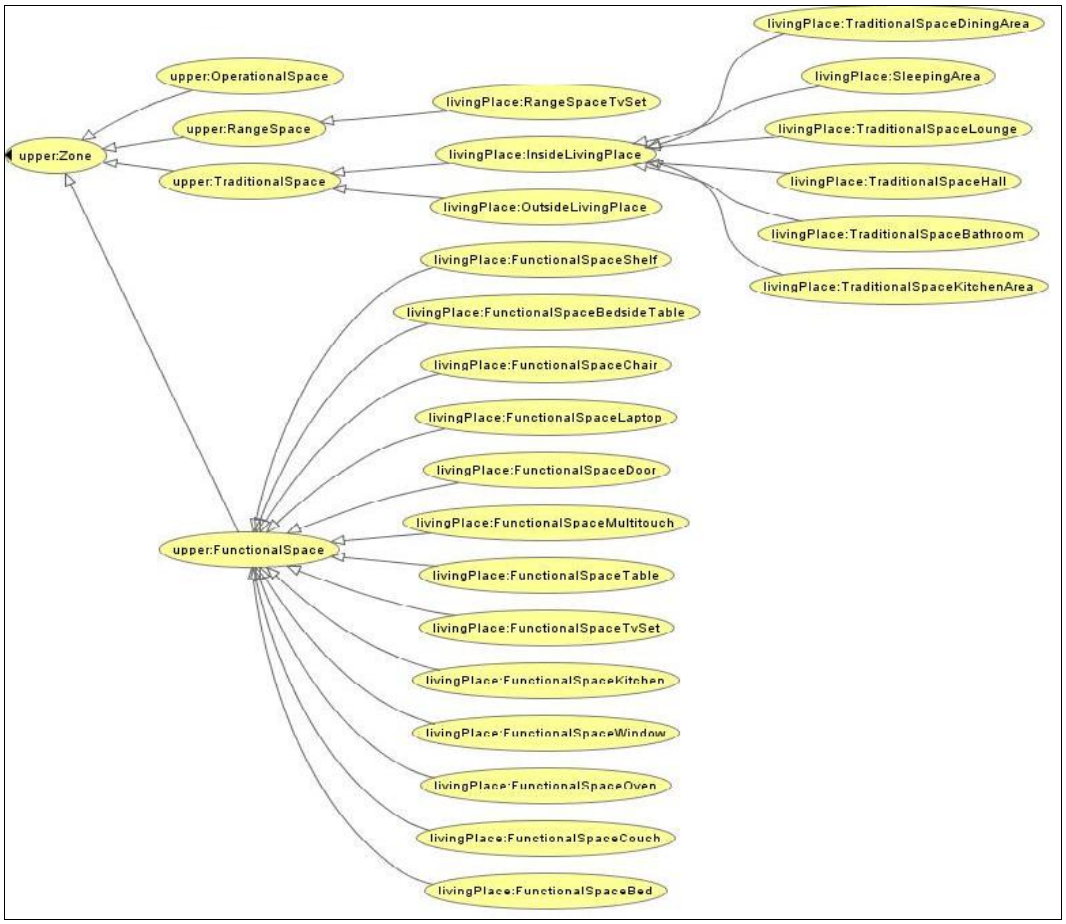

domain:Loader-Leaves-AFT
domain:Loader-Leaves-FWD
domain:Catering-Leaves-Right-AFT
domain:Passenger-Bridge-Leaves-PBB-Zone
domain:Tanker-Leaves-Tanking-Zone
domain:Vehicle-Leaves-Zone
domain:Transporter-Leaves-FWD
domain:Transporter-Leaves-AFT
domain:Catering-Leaves-Left-AFT
domain:Catering-Leaves-Right-FWD
upper:Mobile-Leaves-Zone
domain:Aircraft-Leaves-Arrival-Zone
domain:Loader-Positioning-FWD
domain:Catering-Left-Positioning-AFT
domain:GPU-Positioning
domain:Tanker-Positioning
domain:Pushback-Positioning
upper:Primitive-Event
upper:Mobile-Positioning
domain:Vehicle-Positioning
domain:Catering-Right-Positioning-FWD
domain:Notification
domain:Aircraft-Positioning
domain:Transporter-Positioning-FWD
domain:Loader-Positioning-AFT
domain:Passenger-Bridge-Positioning
domain:Catering-Right-Positioning-AFT
domain:Transporter-Positioning-AFT


domain:Loader-Positioned-FWD
domain:Catering-Right-Positioned-FWD
domain:Loader-Positioned-AFT
domain:Transporter-Positioned-FWD
domain:Aircraft-Positioned
domain:Catering-Right-Positioned-AFT
upper:Primitive-State
upper:Mobile-Positioned
domain:Vehicle-Positioned
domain:Pushback-Positioned
domain:Transporter-Positioned-AFT
domain:GPU-Positioned
domain:Catering-Left-Positioned-AFT
domain:Tanker-Positioned
domain:Passenger-Bridge-Positioned


domain:Services-Without-CR
domain:Departure
domain:Services-With-R
domain:Pushback-Arrival
domain:Catering-Right-FWD
domain:Services
domain:Catering-Left-AFT
domain:Catering
domain:Catering-Right-AFT
domain:Refuelling
upper:Three-Sequence
upper:Visit
domain:Loading-FWD
upper:Composite-State
domain:Unloading-Loading
domain:Loading-Only
domain:Loading-AFT-And-FWD
upper:Composite-Event
upper:Two-Sequence
domain:Unloading-And-Loading
domain:Loading-AFT
domain:Arrival-Preparation
domain:Unloading-AFT-And-FWD
domain:Unloading-Only
domain:Turnaround
domain:Unloading-FWD
domain:Loading-Operation
domain:Loading-Operation-FWD
domain:Unloading-AFT
domain:Loading-Operation-AFT
domain:Arrival
domain:Passenger-Boarding-Preparation
domain:Aircraft-Arrival

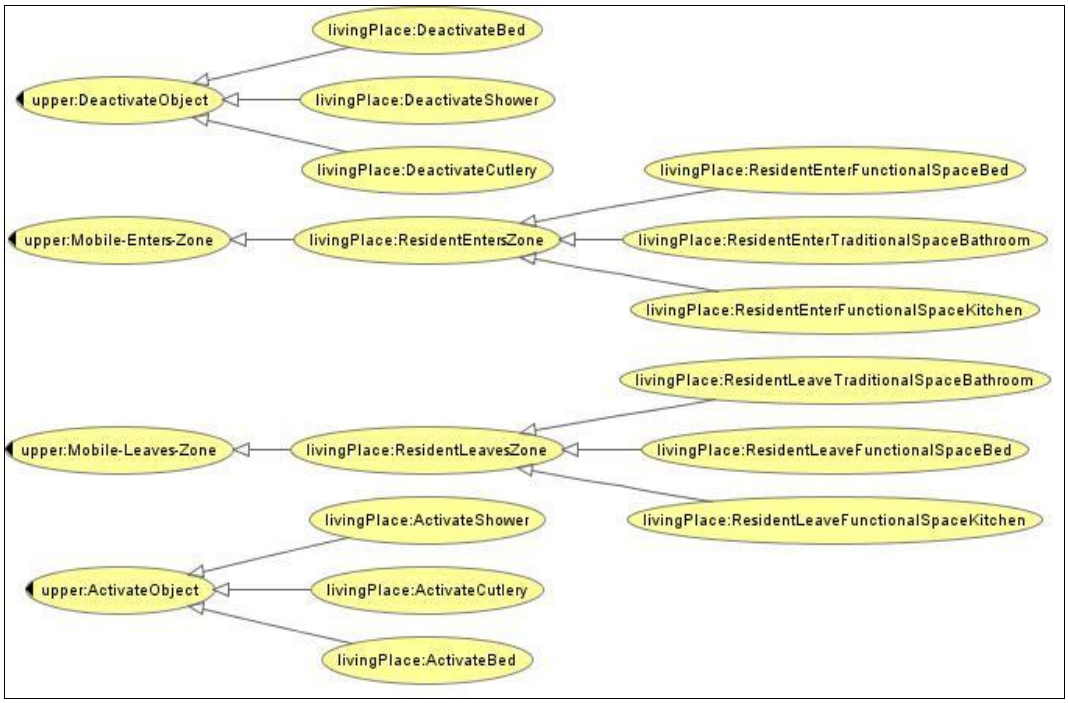
A.2 Taxonomie der Living-Place-Domäne






livingPlace:Resident
upper:Person
livingPlace:Visitor
livingPlace:Clothe
upper:Mobile
livingPlace:TvSet
livingPlace:Table
upper:EnvironmentObject
livingPlace:Chair
livingPlace:Laptop
livingPlace:BedsideTable
livingPlace:Cutlery
livingPlace:Bed
livingPlace:Shelf
livingPlace:Furniture
livingPlace:Couch
livingPlace:Multitouch
upper:Static
livingPlace:Door
livingPlace:Window
livingPlace:EnvironmentComponent
livingPlace:Basin
livingPlace:RotatingCamera
livingPlace:VideoCamera
livingPlace:VideoCamera180Deg
livingPlace:Lighting
livingPlace:Shower
livingPlace:KitchenUnit


livingPlace:ResidentActionMorningActivity
livingPlace:ResidentActionSleepEndObject
livingPlace:ResidentActionSleep
livingPlace:ResidentActionSleepEndZone
livingPlace:ResidentAction
livingPlace:ResidentActionBreakfast
livingPlace:ResidentActionBreakfastEndObject
livingPlace:ResidentActionBreakfastEndZone
livingPlace:ResidentActionTakingShower
livingPlace:ResidentActionTakingShowerEndObject
livingPlace:ResidentActionTakingShowerEndZone
livingPlace:ResidentActionEndTakingShowerLeavingZone
livingPlace:ResidentActionEndTakingShower
livingPlace:ResidentActionEndTakingShowerDeactivateObject
livingPlace:ResidentActionEndBreakfastLeavingZone
livingPlace:ResidentActionEnd
livingPlace:ResidentActionEndBreakfast
livingPlace:ResidentActionEndBreakfastDeactivateObject
livingPlace:ResidentActionEndSleep
livingPlace:ResidentActionEndSleepLeavingZone
livingPlace:ResidentActionEndSleepDeactivateObject
livingPlace:ResidentActionStartTakingShowerZoneObject
livingPlace:ResidentActionStartTakingShower
livingPlace:ResidentActionStartTakingShowerObjectZone
livingPlace:ResidentActionStartBreakfastZoneObject
livingPlace:ResidentActionStart
livingPlace:ResidentActionStartBreakfast
livingPlace:ResidentActionStartBreakfastObjectZone
livingPlace:ResidentActionStartSleep
livingPlace:ResidentActionStartSleepZoneObject
livingPlace:ResidentActionStartSleepObjectZone


livingPlace:DeactivateBed
upper:DeactivateObject
livingPlace:DeactivateShower
livingPlace:DeactivateCutlery
livingPlace:ResidentEnterFunctionalSpaceBed
upper:Mobile-Enters-Zone
livingPlace:ResidentEntersZone
livingPlace:ResidentEnterTraditionalSpaceBathroom
livingPlace:ResidentEnterFunctionalSpaceKitchen
livingPlace:ResidentLeaveTraditionalSpaceBathroom
upper:Mobile-Leaves-Zone
livingPlace:ResidentLeavesZone
livingPlace:ResidentLeaveFunctionalSpaceBed
livingPlace:ResidentLeaveFunctionalSpaceKitchen
livingPlace:ActivateShower
upper:ActivateObject
livingPlace:ActivateCutlery
livingPlace:ActivateBed

B Anlage: SCENIOR

B.1 Konfiguration

Verschiedenes	
Name	**Beschreibung**
MAIN_HYPOTHESIS	Definierte das Wurzelkonzept des Modells. Ausgehend davon werden die Hypothesengraphen generiert.
MAX_NUMBER_OF_THREADS	Maximale Anzahl der zulässigen Threads.
MAX_CLUTTER_SIZE	Maximale Anzahl der erlaubten Clutter-Elemente.
MULTITHREADING	True : alle Regelmaschinen laufen in eigenen Threads. False: die Regelmaschinen werden sequenziell abgearbeitet.
DATA_MODE	DType.DATAFILE : Eingabedaten via Datendatei. Dtype.MIDDLEWARE: Eingabedaten via Corba. Dtype.SQL : Eingabedaten via Datenbank.
SI_OUTPUT_TO_FILE	True: schreibt das Interpretationsergebnis jedes abgeschlossenen Threads in eine separate Datei, in das Verzeichnis OUTPUT_DIR.
HALLUCINATION	True : erlaubt Halluzinieren von Evidenzen. False: ignoriert is-hallucinatable-Deklarationen der Ontologie.
IGNORE_OBJECT_TYPES	True: ignoriert Objekttyp-Informationen der Evidenzen.
REALTIME	True: simuliert Realzeit, wenn Eingabedaten von Datei oder Datenbank gelesen werden.
ONLY_MAIN_CONCEPT	True: ignoriert is-context-free-Deklarationen der Ontologie, d.h., es wird initial nur ein Hypothesengraph erzeugt.
CONSOLE_MODE	True: startet SCENIOR ohne Benutzungsoberfläche (für Experimentreihen).
NEW_PARSER	True : verwendet das neue Eingabeformat, mit Angabe der Rollennamen (z.B. für Projekte: RACE, LivingPlace). False: verwendet das alte Eingabeformat (Projekt: Co-Friend).
USE_MATCHING_TABLE	True: verwendet Zuordnungstabelle (MATCHING_TABLE), falls Konzeptnamen in Eingabedaten und Ontologie nicht übereinstimmen.
USE_PRIM_STATISTIC	True: verwendet Statistik von primitiven Evidenzen (PRIM_STATISTIC) für Berechnung der Clutter-Wahrscheinlichkeitsdichte.

VIDEO_VIS	True: aktiviert Video-Visualisierung.
VIDEO_VIS_URL	URL für Video-Visualisierung.

Ein- / Ausgabedaten	
Name	**Beschreibung**
SM_CLIENT_IP SM_PORT SM_SERVERNAME SM_SERVICENAME	Konfiguration für Datenaustausch via Corba.
SQL_DATASET SQL_DATABASE_URL SQL_USERNAME SQL_PASSWORD	Konfiguration für Dateneingabe via PostreSQL Datenbank.

Zeitliches Constraint-Netz (ZCN)	
Name	**Beschreibung**
TCN_ENABLED	True: ZCN wird verwendet.
TCN_CHECK_SATISFY	True: führt in Zeitintervallen, die durch UPDATE_CURRENT_TIME gesetzt werden, eine Überprüfung der Erfüllbarkeit des ZCN durch.
UPDATE_CURRENT_TIME	Definiert die Zeitintervalle für TCN_CHECK_SATISFY in Millisekunden.

Bayes'sche Kompositionelle Hierachien (BCH)	
Name	**Beschreibung**
BCH_ENABELD	True: BCH wird verwendet.
PROB_PER_CLUTTER	Wahrscheinlichkeit für Clutter-Element.

Protokollierung	
Name	**Beschreibung**
LOG_LEVEL	Setzt die Protokollierungsstufe (Level.OFF, Level.FINE, Level.FINER, Level.FINEST, Level.WARNING).
LOG_TO_APP	True: gibt die Protokollierung im Log-Tab von SCENIOR aus.
LOG_TO_FILE	True: schreibt die Protokollierung in die Datei log.txt, in das Verzeichnis LOG_DIR.

Dateinamen und Verzeichnisse	
Name	**Beschreibung**
TEMPLATES_FILENAME	Name der Datei für Generierung der Jess-Templates.
RULES_FILENAME	Name der Datei für Generierung der Jess-Regeln.
TCN_FILENAME	Name der Datei für Generierung der ZCN Prototypen.
JCKB_DIR	Name des Verzeichnisses für Jess Wissensbasis.
DATA_FILE_DIR	Name des Verzeichnisses für Eingabedaten.
OWL_DIR	Name des Verzeichnisses für Ontologien.
BCH_DIR	Name des Verzeichnisses für BCH-Modelle.
LOG_DIR	Name des Verzeichnisses für Ausgabe einer Protokollierungsdatei.
OUTPUT_DIR	Name des Verzeichnisses für Ausgabe von Interpretationsergebnissen.
MATCHING_TABLE	Name der Zuordnungstabellendatei (nur, wenn USE_MATCHING_TABLE = true).
PRIM_STATISTIC	Name der Datei mit Statistik über primitive Evidenzen (nur, wenn USE_PRIM_STATISTIC = true).

Benutzungsoberfläche	
Name	**Beschreibung**
X_SIZE	Breite des SCENIOR-Fensters.
Y_SIZE	Höhe des SCENIOR-Fensters.

FONT_SIZE	Schriftgröße für Ausgabe-Tabs in SCENIOR.
SIMPLE_GRAPH	True: Zeigt physikalische Objekte im Hypothesengraph nicht an.
VIS_TAB RULES_TAB ACT_TAB FACTS_TAB	Schaltet die entsprechenden Ausgabe-Tabs ein bzw. aus.
PRIMITIVES_TAB	Schaltet den Primitives-Tab ein bzw. aus.
REMOVE_DEAD_THREADS	True: löscht gestorbene Threads in den Ausgabe-Tabs.
ORDER_THREADS	Ordnet Threads anhand des Präferenzmaßes in den Ausgabe-Tabs.

C Anlage: Experimente

C.1 Idealer Datensatz

```
16:00:00.00 - 16:00:00.00: Vehicle_Enters_Zone(VEHICLE 1, ERA)
16:06:00.00 - 16:06:00.00: Vehicle_Positioning(VEHICLE 2, GPU_Zone)
16:07:00.00 - 16:50:00.00: Vehicle_Positioned(VEHICLE 2, GPU_Zone)
16:17:00.00 - 16:17:00.00: Aircraft_Positioning(AIRCRAFT 3, Arrival_Zone)
16:18:00.00 - 17:29:00.00: Aircraft_Positioned(AIRCRAFT 3, Arrival_Zone)
17:29:00.00 - 17:29:00.00: Aircraft_Leaves_Arrival_Zone(AIRCRAFT 3, Arri
                           val_Zone)
16:20:00.00 - 16:20:00.00: Passseger_Bridge_Positioning (PASSENGER_BOARDING
                           _BRIDGE 4, PBB_Zone)
16:20:00.00 - 17:27:00.00: Passenger_Bridge_Positioned(PASSENGER_BOARDING
                           _BRIDGE 4, PBB_Zone)
17:27:00.00 - 17:27:00.00: Passenger_Bridge_Leaves_PBB_Zone(PASSENGER_BOARDING
                           _BRIDGE 5, PBB_Zone)
16:32:00.00 - 16:32:00.00: Vehicle_Positioning(VEHICLE 6, Right_AFT_TS_Zone)
16:32:00.00 - 16:44:00.00: Vehicle_Positioned(VEHICLE 6, Right_AFT_TS_Zone)
16:44:00.00 - 16:44:00.00: Vehicle_Leaves_Zone(VEHICLE 6, Right_AFT_TS_Zone)
17:08:00.00 - 17:08:00.00: Vehicle_Positioning(VEHICLE 7, Right_AFT_TS_Zone)
17:08:00.00 - 17:20:00.00: Vehicle_Positioned(VEHICLE 7, Right_AFT_TS_Zone)
17:20:00.00 - 17:20:00.00: Vehicle_Leaves_Zone(VEHICLE 7, Right_AFT_TS_Zone)
16:39:00.00 - 16:39:00.00: Vehicle_Positioning(VEHICLE 8, Left_TK_Zone)
16:39:00.00 - 16:56:00.00: Vehicle_Positioned(VEHICLE 8, Left_TK_Zone)
16:56:00.00 - 16:56:00.00: Vehicle_Leaves_Zone(VEHICLE 8, Left_TK_Zone)
16:32:00.00 - 16:32:00.00: Vehicle_Positioning(VEHICLE 9, PB_Zone)
16:32:00.00 - 16:44:00.00: Vehicle_Positioned(VEHICLE 9, PB_Zone)
16:30:00.00 - 16:30:00.00: Vehicle_Positioning(VEHICLE 10, Right_AFT_LD_Zone)
17:21:00.00 - 17:21:00.00: Vehicle_Leaves_Zone(VEHICLE 10, Right_AFT_LD_Zone)
```

C.2 Datensatz des Experiments der Living-Place-Domäne

```
07:44:00.00 - 07:44:00.00: ResidentEnterFunctionalSpaceBed(has-agent: Resident
                           1, has-location: FunctionalSpaceBed 8)
07:45:20.00 - 07:45:20.00: ActivateBed(has-object: Bed 2)
08:00:40.00 - 08:00:40.00: DeactivateBed(has-object: Bed 2)
08:01:00.00 - 08:01:00.00: ResidentLeaveFunctionalSpaceBed(has-agent: Resident
                           1, has-location: FunctionalSpaceBed 8)
08:01:20.00 - 08:01:20.00: ResidentEnterTraditionalSpaceBathroom(has-agent:
                           Resident 1, has-location: TraditionalSpaceBathroom
                           9)
08:02:20.00 - 08:02:20.00: ActivateShower(has-object: Shower 4)
08:07:40.00 - 08:07:40.00: DeactivateShower(has-object: Shower 4)
08:14:00.00 - 08:14:00.00: ResidentLeaveTraditionalSpaceBathroom(has-agent:
                           Resident 1, has-location: TraditionalSpaceBathroom
                           9)
08:15:00.00 - 08:15:00.00: ResidentEnterFunctionalSpaceKitchen(has-agent:
                           Resident 1, has-location: FunctionalSpaceKitchen
                           10)
08:17:20.00 - 08:17:20.00: ActivateCutlery(has-object: Cutlery 7)
08:33:40.00 - 08:33:40.00: ResidentLeaveFunctionalSpaceKitchen(has-agent:
                           Resident 1, has-location: FunctionalSpaceKitchen
                           10)
08:30:00.00 - 08:30:00.00: DeactivateCutlery(has-object: Cutlery 7)
```

C.3 Statistik der primitiven Evidenzen

Datensatz	cof 01	cof 02	cof 03	cof 04	cof 05	cof 06	cof 08	cof 09	cof 16	cof 18	cof 20	cof 25	cof 29	cof 30	cof 58	cof 59	cof 62	cof 63	cof 65	cof 66
GPU-Positioning	2	8	1	9	1	6	4	7	9	1	1	9	7	4	8	2	2	3	5	6
GPU-Positioned	4	34	6	25	13	19	32	19	23	2	4	17	20	9	17	9	6	8	10	33
Aircraft-Positioning	1	4	5	1	5	8	12	9	13	3	5	2	5	1	5	4	1	5	2	10
Aircraft-Positioned	10	38	21	28	17	22	36	54	45	11	16	31	34	18	31	8	16	8	14	35
Aircraft-Leaves-Arrival-Z	18	26	14	7	8	16	24	28	33	12	14	14	39	17	14	18	11	7	13	24
Passenger-Bridge-P	0	0	0	0	0	0	1	5	0	0	1	0	0	0	0	0	0	0	1	0
Passenger-Bridge-PD	1	0	0	0	0	0	0	5	0	0	1	0	0	0	0	0	0	0	6	0
Passenger-Bridge-L-PBB-Z	0	0	1	0	1	0	2	13	2	0	3	0	0	1	0	0	1	0	4	2
Tanker-Positioning	2	1	0	0	2	0	0	5	0	1	1	0	0	1	0	0	1	0	1	5
Tanker-Positioned	2	0	0	1	2	0	0	6	2	1	0	0	0	1	0	3	7	1	3	5
Tanker-Leaves-Tanking-Zone	2	7	2	1	3	0	0	10	2	2	2	1	0	3	1	2	2	1	11	8
Pushback-Positioning	0	3	2	0	2	2	3	1	1	3	1	0	4	3	1	0	1	0	2	3
Pushback-Positioned	0	15	4	2	3	4	22	4	7	1	2	7	21	7	3	3	2	1	8	5
Loader-Positioning-FWD	0	1	1	1	3	3	1	0	1	1	0	3	1	1	2	0	1	0	3	2
Loader-Leaves-FWD	3	7	2	5	4	5	2	4	6	1	1	7	0	9	1	1	1	0	8	4
Loader-Positioning-AFT	2	2	1	1	4	0	2	1	3	4	1	3	7	2	0	0	1	0	3	1
Loader-Leaves-AFT	2	5	4	14	4	3	4	7	20	3	3	10	33	14	10	3	2	4	8	4
Transporter-Positioning-AFT	2	2	0	2	1	2	2	5	8	0	2	0	6	6	6	2	1	2	0	5
Transporter-Positioned-AFT	2	3	2	7	2	5	7	14	12	1	2	1	8	9	15	5	3	2	1	7
Transporter-Leaves-FWD	12	13	2	7	4	9	11	25	20	2	5	4	14	16	12	9	7	2	6	14
Transporter-Positioning-FWD	0	0	0	1	2	4	0	0	6	4	0	3	4	7	4	0	1	3	2	3
Transporter-Positioned-FWD	6	3	3	2	3	12	2	3	13	12	3	3	12	10	7	1	2	2	2	4
Transporter-Leaves-AFT	7	6	3	2	2	15	4	9	13	11	3	3	12	15	12	4	1	3	5	9

C.4 Statistik der primitiven Evidenzen mit Vorverarbeitungsschritt „Langzeit-Tracking“

Datensatz	cof 01	cof 02	cof 03	cof 04	cof 05	cof 06	cof 08	cof 09	cof 16	cof 18	cof 20	cof 25	cof 29	cof 30	cof 58	cof 59	cof 62	cof 63	cof 65	cof 66
GPU-Positioning	2	8	1	9	1	6	4	7	9	1	1	8	6	3	7	2	2	3	5	6
GPU-Positioned	3	26	3	18	6	13	15	12	13	1	4	11	19	5	8	6	5	6	9	15
Aircraft-Positioning	1	4	5	1	5	8	12	9	12	3	5	2	3	1	4	3	1	5	2	10
Aircraft-Positioned	6	32	15	21	14	18	23	45	33	8	14	27	29	18	24	8	11	8	14	28
Aircraft-Leaves-Arrival-Z	18	23	14	7	8	16	24	25	30	10	14	13	35	15	13	10	11	6	13	24
Passenger-Bridge-P	0	0	0	0	0	0	1	5	0	0	1	0	0	0	0	0	0	0	1	0
Passenger-Bridge-PD	1	0	0	0	0	0	0	2	0	0	1	0	0	0	0	0	0	0	4	0
Passenger-Bridge-L-PBB-Z	0	0	1	0	1	0	2	7	2	0	3	0	0	1	0	0	1	0	4	2
Tanker-Positioning	2	1	0	0	2	0	0	5	0	1	1	0	0	1	0	0	1	0	1	5
Tanker-Positioned	2	0	0	1	1	0	0	6	1	1	0	0	0	1	0	2	3	1	2	4
Tanker-Leaves-Tanking-Zone	2	7	2	1	2	0	0	7	2	2	2	1	0	3	1	2	2	1	8	6
Pushback-Positioning	0	3	2	0	2	2	3	1	1	2	1	0	4	3	1	0	1	0	2	2
Pushback-Positioned	0	10	3	2	3	4	13	4	6	1	2	7	6	7	3	2	2	1	8	3
Loader-Positioning-FWD	0	1	1	1	3	3	1	0	1	1	0	3	1	1	2	0	1	0	3	2
Loader-Leaves-FWD	3	6	2	5	3	3	2	4	5	1	1	7	0	8	1	1	1	0	7	4
Loader-Positioning-AFT	2	2	1	1	4	0	2	1	3	3	1	3	6	2	0	0	1	0	3	1
Loader-Leaves-AFT	2	4	4	14	4	2	4	7	19	3	3	9	29	9	8	3	2	3	5	4
Transporter-Positioning-AFT	2	2	0	2	1	2	1	5	6	0	2	0	5	5	5	2	1	2	0	5
Transporter-Positioned-AFT	2	2	2	5	2	5	6	14	8	1	2	1	8	8	13	4	2	2	1	7
Transporter-Leaves-FWD	12	11	2	7	4	9	8	19	16	2	4	4	13	14	9	6	7	2	5	13
Transporter-Positioning-FWD	0	0	0	1	2	4	0	0	5	4	0	3	3	7	4	0	1	3	2	3
Transporter-Positioned-FWD	6	3	3	1	3	11	2	2	10	6	3	3	8	9	6	1	2	2	2	4
Transporter-Leaves-AFT	7	5	3	1	2	12	4	8	11	11	3	3	9	12	12	4	1	3	5	8

Abkürzungsverzeichnis

ABox .....Assertional Box
BCH .....Bayes'sche Kompositionelle Hierachien
CMV.....Colour Mean and Variance
DL .....Description Logic
EAN .....Expected Area Net
GPU.....Ground Power Unit
GSB.....Geometrische Szenenbeschreibung
GWV .....gemeinsame Wahrscheinlichkeitsverteilung
HBN .....Hierarchische Bayes-Netze
Jess .....Java Expert System Shell
KI .....Künstliche Intelligenz
KJW .....Konzeptuelle Jess Wissensbasis
LHS .....left hand side
MLN.....Markov-Logik-Netzwerke
OWL.....Web Ontology Language
PTZ.....Pan, Tilt, Zoom
RDF.....Resource Description Framework
RDFS.....Resource Description Framework Schema
RHS.....right hand side
RIF .....Rule Interchange Format
RPC .....Remote Procedure Call
RuleML .....Rule Markup Language
SAL .....Set Attributive Logic
SED .....Simple Event Detector
SVT .....Specific Vision Task
SWRL.....Semantic Web Rule language
TBox.....Terminological Box
URI.....Uniform Resource Identifier
W3C .....World Wide Web Konsortium
XML.....Extensible Markup Language
ZNS .....zeitliches Constraint-Netz

Literaturverzeichnis

[1] (o. V.): Automatische Verkehrsüberwachung. 2011. URL: http://de.wikipedia.org/wiki/Automatische_Verkehrsüberwachung. Stand: 22.07.2011.

[2] (o. V.): CASAS Smart Home Project. 2011. URL: http://ailab.wsu.edu/casas. Stand: 29.07.2011.

[3] (o. V.): Cross-correlation. 2012. URL: http://en.wikipedia.org/wiki/Cross-correlation#Normalized_cross-correlation. Stand: 18.04.2012.

[4] (o. V.): Drools – The Business Logic Integration Platform. 2011. URL: http://www.jboss.org/drools. Stand: 14.11.2011.

[5] (o. V.): HAW Hamburg. 2011. URL: http://livingplace.informatik.haw-hamburg.de/blog/ Stand: 27.04.2012.

[6] (o. V.): International Federation of Robotics. 2011. URL: http://www.ifr.org/service-robots. Stand 28.07.2011.

[7] (o. V.): Internet der Dinge. 2011. URL: http://de.wikipedia.org/wiki/Internet_der_Dinge. Stand 29.07.2011.

[8] (o. V.): Jess – the Rule Engine for the Java Platform v71p2. 2012. URL: http://www.jessrules.com/docs/71/api.html. Stand: 12.03.2012.

[9] (o. V.): Jess, the Rule Engine for the Java Platform. 2011. URL: http://www.jessrules.com/. Stand: 14.11.2011.

[10] (o. V.): Racer. 2011. URL: http://www.racer-systems.com/products/racerpro. Stand: 30.08.2011.

[11] (o. V.): Schilder-Erkennung: Opel-Eye und Blaupunkt-Gerät im Vergleich. 2011. URL: http://www.autoplenum.de/Auto/Testberichte/Schilder-Erkennung--Opel-Eye-und-Blaupunkt-Geraet-im-Vergleich-id7659.html. Stand: 21.07.2011.

[12] (o. V.): Semantisches Web. 2011. URL: http://de.wikipedia.org/wiki/ Semantisches_Web. Stand: 04.11.2011.

[13] (o. V.): SWRLJessTab. 2012. URL: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLJessTab. Stand: 06.12.2012.

[14] (o. V.): SWRLTemporalOntology. 2011. URL: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalOntology. Stand: 21.11.2011.

[15] (o. V.): The AVITRACK project. 2006. URL: http://www-sop.inria.fr/members/Francois.Bremond/topicsText/avitrackProject.html. Stand 20.03.2012.

[16] (o. V.): The Blue Brain Projekt. 2011. URL: http://bluebrain.epfl.ch. Stand: 02.08.2011.

[17] (o. V.): The Co-Friend project. 2011. URL: http://84.14.57.154/co-friend. Stand 20.03.2012.

[18] (o. V.): The Internet of Things. ITU Internet Reports, International Telecommunication Union, 2005.

[19] (o. V.): The Rule Markup Initiative. 2011. URL: http://ruleml.org/. Stand: 04.11.2011.

[20] (o. V.): Überwachungskameras auf dem Kiez werden abgeschaltet. 2011. URL: http://www.mopo.de/hamburg/panorama/ueberwachungskameras-auf-dem-kiez-werden-abgeschaltet/-/5067140/8675656/-/index.html. Stand: 22.07.2011.

[21] (o. V.): Verkehrsunfälle. 2011. URL: http://www.destatis.de/jetspeed/portal/cms/Sites/destatis/Internet/DE/Navigation/Statistiken/Verkehr/Verkehrsunfaelle/Verkehrsunfaelle.psml. Stand: 21.07.2011.

[22] (o. V.): World Wide Web Consortium. 2011. URL: http://www.w3.org/. Stand: 25.07.2011.

[23] Akka: Co-Friend Deliverable D3.2 Co-Friend PTZ-Sensor platform, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[24] Allen, J. F.; Ferguson, G.: Actions and Events in Interval Temporal Logic. Journal of Logic and Computation 4(5), 1994, S. 531 - 579.

[25] Anicic, D.; Fodor P.; Rudolph, S.; Stühmer, R.; Stojanovic N.; Studer, R.: ETALIS: Rule-Based Reasoning in Event Processing. Studies in Computational Intelligence, 2011, Volume 347/2011, S. 99 - 124.

[26] Artale, A.; Franconi, E.; Guarino, N.; Pazzi, L.: Part-Whole Relations in Object-Centered Systems: an Overview. Data and Knowledge Engineering, 20(3), 1996, S. 347 - 383.

[27] Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P. F.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2007.

[28] Badler, N. I.: Temporal Scene Analysis: Conceptual Description of Objects Movements. Report TR 80, Dep. Of Computer Science, University of Toronto, 1975.

[29] Bense, H.; Bodrow W.: Objektorientierte und regelbasierte Wissensverarbeitung. Spektrum Verlag, Heidelberg, 1995, S. 29 - 34.

[30] Binford, T. O.; Levitt, T. S.; Mann, W. B.: Bayesian inference in model-based machine vision. In Levitt et al. (Hrsg.), Uncertainty in AI(3), 1987, S. 73 - 96.

[31] Bohlken W.; Koopmann, P.; Hotz, L.; Neumann, B.: Towards Ontology-based Realtime Behaviour Interpretation. Human Behaviour Recognition Technologies: Intelligent Applications for Monitoring and Security, IGI Global. Voraussichtliche Veröffentlichung: 2013.

[32] Bohlken, W.; Neumann, B.: Generation of Rules from Ontologies for High-level Scene Interpretation. In: G. Governatori et al. (Hrsg.): Rule Interchange and Applications, Proc. International Symposium RuleML 2009, Springer, LNCS 5858, 2009, S. 93 - 107.

[33] Bohlken, W.; Neumann, B.; Hotz, L.; Koopmann, P.: Ontology-Based Realtime Activity Monitoring Using Beam Search. In: Crowley, J.L. et al. (Hrsg.): ICVS 2011, Springer, Heidelberg, LNCS 6962, 2011, S. 112 - 121.

[34] Borth, M.: Wissensgewinnung auf Bayes-Mengen. Dissertation. Fakultät für Informatik, Universität Ulm, 2004.

[35] Burg, J. F. M.: Linguistic Instruments in Requirements Engineering. IOS Press, 1997.

[36] Cavanagh, P.: Top-down processing in vision. In R.A. Wilson & F.C. Keil, the MIT encyclopedia of the cognitive science, Cambridge, 1999, S. 844 - 845.

[37] Chau, D. P.; Brémond, F.; Thonnat, M; Corvee, E.: Robust Mobile Object Tracking Based on Multiple Feature Similarity and Trajectory Filtering. The International Conference on Computer Vision Theory and Applications (VISAPP), 2011, S. 569 - 574.

[38] Co-Friend partners CSL, UNIVLEEDS, READING, INRIA: Deliverable D1.3b Scene Interpretation and Reasoning Methods, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[39] Cohn, A. G.; Bennet, B.; Gooday, J.; Gotts, N.: Representation and reasoning with qualitative spatial relations about regions. In: Stock, O.: Temporal and spatial reasoning, Kluwer, 1997.

[40] Cohn, A. G.; Dubba, K.; Greenall, J.; Koopmann, P.; Neumann, B.; Patino L.; Sridhar K.: Co-Friend Deliverable D4.3 Flexibility and Learning Capacities Evaluation, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[41] Cohn, A. G.; Magee, D.; Galata, A.; Hogg, D.; Hazarika, S.: Towards an architecture for cognitive vision using qualitative spatio-temporal representations and abduction. In: Freska et al. (Hrsg.), Spatial Cognition III, S. 232 - 248.

[42] Cucchiara, R.; Piccardi M.; Mello, P.: Image Analysis and Rule-Based Reasoning for a Traffic Monitoring System. IEEE Transactions on Intelligent Transportation Systems, Vol. 1, No. 2, 2000, S. 758 - 763.

[43] Davis, R.; Buchanan, G.; Shortliffe, E. H.: Production Systems as a Representation for a Knowledge-Based Consultation Program. In Artifical Intelligence 8, 1977, S. 15 - 45.

[44] Duda, R. O.; Gaschnig, J.; Hart, P. E.: Model design in the PROSPECTOR consultant system for mineral exploration. In: Michie, 1979.

[45] Dworschak, M.: Die Antwortmaschine. 2009. URL: http://www.spiegel.de /spiegel/print/d-65330443.html. Stand 25.07.2011.

[46] Ebbinghaus, H.-D.; Flum, J.; Thomas, W.: Einführung in die mathematische Logik. Spektrum Akademischer Verlag, 2001.

[47] Ellenberg, J.: Ontologiebasierte Aktivitätserkennung im Smart Home Kontext. Master Thesis, HAW Hamburg, 2011.

[48] Erikson, H.: Using JessTab to integrate Protégé and Jess. IEEE Intelligent Systems, 18(2), 2003, S. 43 - 50.

[49] Evans, M.: Co-Friend Deliverable D2.2 Robust Detection and Multiview Tracking, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[50] Evans, M.: Co-Friend Deliverable D2.3 Video & GNSS Multimodal Data Fusion, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[51] Fox, M.; Long, D.: An Extension to PDDL for Expressing Temporal Planning Domains. In Journal of Artificial Intelligence Research 20, 2003, S. 61 - 124.

[52] Friedman-Hill, E.: Jess in Action: Java Rule-Based Systems. Manning, Greenwich, 2003.

[53] Fusier, F.; Valentin, V.; Brémond, F.; Thonnat, M.; Borg, M., Thirde, D.; Ferryman, J.: Video understanding for complex activity recognition. Machine Vision and Applications, 18(3), Springer, 2007, S. 167 - 188.

[54] Gaines, B.: Using Explicit Ontologies in Knowledge-based System Development. In International Journal of Human-Computer Systems (46), 1997.

[55] Gries, O.; Möller, R.; Nafissi, A; Rosenfeld, M.; Sokolski, K.; Wessel, M.: A probabilistic abduction engine for media interpretation. Proc. Fourth Int. Conf. on Web reasoning and rule systems, 2010, S. 182 - 194.

[56] Gruber, T. R.: A translation approach to portable ontologies. In Knowledge Acquisition, 5(2), 1993, S. 199 - 220.

[57] Gruninger, M.; Lee, J.: Ontology - applications and design. Comm. ACM 45(2), 2002, S. 39 - 41.

[58] Grütter, R.: Semantic Web zur Unterstützung von Wissensgemeinschaften. Oldenbourg Wissensschaftsverlag GmbH, 2008.

[59] Guarino, N.: Semantic Matching: Formal Ontology Distinctions for Information Organisation, Extraction and Integration. In M. T. Pazienza (Hrsg.), Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology. Springer Verlag, 1997, S. 139 - 170.

[60] Guarino, N.; Giarette, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N.J.I. Mars (Hrsg.), Towards Very Large Knowledge Bases, IOS Press, 1995.

[61] Gyftodimos, E.; Flach, P. A.: Hierarchical Bayesian Networks: A probabilistic reasoning model for structured domains. In de Jong, E.; Oates, T. (Hrsg.), Proc. Workshop on Development of Representations, ICML, 2002, S. 23 - 30.

[62] Hawkins, J.: Die Zukunft der Intelligenz. Rowohlt Taschenbuch Verlag. Reinbek bei Hamburg. 2006.

[63] Hitzler, P.; Krötzsch, M; Rudolph, S.; Sure, Y.: Semantic Web. Springer Verlag Berlin, Heidelberg, 2008.

[64] Hotz L.; Neumann, B.: Scene Interpretation as a Configuration Task. Künstliche Intelligenz, BöttcherIT Verlag, Bremen, 2005, S. 59 - 65.

[65] Hotz, L.; Neumann, B.; Terzić, K.: High-level Expectations for Low-level Image Processing. Proc. KI-2008, Springer, S. 87 - 94.

[66] Hu, Y.-J.; Yeh C.-L.; Challenges for Rule Systems on the Web. In: G. Governatori et al. (Hrsg.): Rule Interchange and Applications, Proc. International Symposium RuleML 2009, Springer, LNCS 5858, 2009, S. 4 - 16.

[67] Ijsselmuiden, J.; Stiefelhagen, R.: Towards High-Level Human Activity Recognition through Computer Vision and Temporal Logic. In Proc. KI 2010: Advances in artificial intelligence, Springer, 2010, S. 426 - 435.

[68] Karimi V.: Semantic Web Rule Language (SWRL), 2008. URL: http://www.cs.uwaterloo.ca/~gweddell/cs848/Vahid.pdf. Stand: 07.11.2011.

[69] Kifer, M.: Rule Interchange Format: The Framework. In Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems, Springer-Verlag Berlin, Heidelberg, 2008.

[70] Koivunen, M.-R.; Miller, E.: W3C Semantic Web Activity. Proceedings of the Semantic Web Kick-off Seminar, Finland, Universität Helsinki, 2001.

[71] Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer, 2006.

[72] Litzenberger, G.: Industrieroboter – Schlüsselkomponente für die Automation. 2008. URL: http://www.openautomation.de/898-0-industrieroboter--schluesselkomponente-fuer-die-automation.html

[73] Luger, G.: Künstliche Intelligenz – Strategien zur Lösung komplexer Probleme. 4. Aufl., Pearson Studium, München, 2001.

[74] Luyken, R.: Big Brother ist wirklich ein Brite. 2007. URL: http://www.zeit.de/2007/03/Big-Brother. Stand: 22.07.2011.

[75] Matsuyama, T.; Hwang, V. S.: SIGMA – A Knowledge-Based Aerial Image Understanding System. Plenum Press, New York, 1990.

[76] Möller, R.; Neumann, B.: Ontology-Based Reasoning Techniques for Multimedia Interpretation and Retrieval. In: Y. Kompatsiaris, P. Hobson (Hrsg.): Semantic Multimedia and Ontologies: Theory and Applications, Springer, 2008, S. 55 - 98.

[77] Mumford, D.; Zhu, S.-C.: A stochastic grammar of images. Now Publishers, 2007.

[78] N. Guarino: Formal Ontology and Information Systems. In: Proc. FOIS'98, Trento (Italy), 1998, S. 3 - 15.

[79] Neumann, B.: Bayesian Compositional Hierarchies – A Probabilistic Structure for Scene Interpretation. Technischer Report FBI-HH-B-282/08, Department Informatik, Universität Hamburg, 2008.

[80] Neumann, B.: Natural Language Description of Time-Varying Scenes. In: D. Waltz (ed.), Semantic Structures, Lawrence Erlbaum, 1989, S. 167 - 206.

[81] Neumann, B.; Bohlken W.; Koopmann, P.; Hotz, L.; Fraile R.: Co-Friend Deliverable D1.3c Scene Interpretation and Reasoning Methods, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2011.

[82] Neumann, B.; Moeller, R.: On Scene Interpretation with Description Logic. In: Cognitive Vision Systems, H.-H. Nagel und H. Christensen (Hrsg.), Springer, LNCS 3948, 2006, S. 247 - 275.

[83] Neumann, B.; Terzić, K.: Context-based Probabilistic Scene Interpretation. In: Proc. IFIP AI-2010, Brisbane, Sept. 2010, S. 155 - 164.

[84] Neumann, B.; Weiss, T.: Navigating through logic-based scene models for high-level scene interpretations. In Proc. 3rd Int. Conf. On Computer Vision Systems (ICVS-2003), Springer, 2003, S. 212 - 222.

[85] Patino, L.; Brémond, F.; Evans, M.; Shahrokni A.; Ferryman J.: Video Activity Extraction and Reporting with Incremental Unsupervised Learning. Proc. of 7th IEEE International Conference on Advanced Video and Signal-based Surveillance, 2010.

[86] Priesterjahn, S.: Online Limitation and Adaption in Modern Computer Games. Dissertation, Universität Paderborn, 2007.

[87] Rashidi, P.; Cook, C.: Keeping the resident in the loop: Adapting the smart home to the user. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 39(5), 2009, S. 949 - 959.

[88] Reiter, R.; Mackworth, A.: The Logic of Depiction. TR 87-13, Dept. Computer Science, Univ. of British Columbia, Vancouver, Canada, 1987.

[89] Riboni, D.; Bettini, C.: COSAR: hybrid reasoning for context-aware activity recognition. Personal and Ubiquitous Computing 15, 2011, S. 271 - 289.

[90] Richardson, M.; Domingos, P.: Markov Logic Networks. Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195-250, USA, 2006.

[91] Rimey, D.; Brown, C.: Control of Selective Perception Using Bayes Nets and Decision Theory. Dissertation, TR 468, University of Rochester, USA, 1993.

[92] Russel, S.; Norvig P.: Künstliche Intelligenz – Ein moderner Ansatz. 2. Aufl., Pearson Studium, München, 2004.

[93] Sartre, J.-P.: Das Sein und das Nichts. Versuch einer phänomenologischen Ontologie, Rowohlt Verlag, Reinbek bei Hamburg, 1991, S. 42.

[94] Schmidt-Schauß, M.; Smolka, G.: Attributive concept descriptions with complements. Artifical Intelligence, 48(1), 1991, S. 1 - 26.

[95] Schöning, U.: Logik für Informatiker. Spektrum Akademischer Verlag, 5 edition, 2000.

[96] Shahrokni, A.; Offroy G.: Specific Vision Tasks. Co-Friend Deliverable D2.4, URL: http://kogs-www.informatik.uni-hamburg.de/projekte/Co-Friend-HH.html, 2010.

[97] Shalloway, A.; Trott, J. R.: Entwurfsmuster verstehen. Mitp-Verlag Bonn, ISBN 3-8266-1345-7, 2003.

[98] Shanahan, M.: Perception as abduction: Turning sensor data into meaningful representation. Cognitive Science, 29, 2005, S. 103 - 134.

[99] Siegwart, R.; Nourbakhsh, I.; Scaramuzza, D.: Introduction to Autonomous Mobile Robots. 2. Aufl., MIT Press, Massachusetts, 2011, S. 1 - 34.

[100] Sridhar, M.; Cohn, A. G.; Hogg, D. C.: Unsupervised Learning of Event Class from Video. Proc. of the Twenty-Fourth AAAI Conference on Artifical Intelligence, 2010.

[101] Trefflich, B.: Videogestützte Überwachung der Fahreraufmerksamkeit und Adaption von Fahrerassistenzsystemen. Dissertation, Technische Universität Ilmenau, 2010.

[102] Tsotsos, J. K.; Mylopoulos, J.; Covvey, H. D.; Zucker, S. W.: A Framework for Visual Motion Understanding. IEEE PAMI-2, 1980, S. 563 - 573.

[103] Vila, L.: A Survey on Temporal Reasoning in Artifical Intelligence. AI Communications 7(1), 1994, S. 4 - 28.

[104] Wand, Y.: A Proposal for a Formal Model of Objects. In W. Kim und F. H. Lochovsky (Hrsg.), Object-Oriented Concepts, Databases and Applications. Addision Wesley, Reading, 1989, S. 537 - 559.

[105] Weiss, A.-K.: Smart Textiles: Entwicklung textiler Sensoren für intelligente Umgebungen am Beispiel eines Sofas. Bachelorarbeit, HAW Hamburg, 2011.

[106] Wojtczyk, M.: A New Model To Design Software Architectures For Mobile Service Robots. Dissertation, Technische Universität München, 2010.

[107] Wolfram, S.: Wolfram Alpha. 2011. URL: http://www.wolframalpha.com. Stand: 25.07.2011.

[108] Wren, C. R.; Azarbayejani, A.; Darrell, T.; Pentland, A. P.: Pfinder: real-time tracking of the human body. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(7), 1997, S. 780 - 785.

[109] Zimmer, F.-M.; Neumann, B.: Incremental Recognition of Multi-object Behaviour Using Hierarchical Probabilistic Models. Proc. of the 5th Workshop on Behaviour Monitoring and Interpretation, BMI'10, CEUR-WS.org, ISSN 1613-0073, 2010, S. 8 - 21.

[110] Zivkovic, Z.: Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recognition Letters, 2006.

Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 21.06.2012 Unterschrift