Dienstekomposition in intelligenten Umgebungen basierend auf KI-Planung

Dissertation

zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

an der Fakultät für Informatik und Elektrotechnik

der Universität Rostock

eingereicht von
Dipl.-Ing. Florian Marquardt
geboren am 1. März 1980 in Magdeburg

Verteidigung:
Rostock, 15. Juni 2010

Gutachter:
Prof. Dr. rer. nat. habil. Adelinde M. Uhrmacher, Universität Rostock
Prof. Dr. rer. nat. Birgitta König-Ries, Universität Jena
Prof. Dr.-Ing. Thomas Kirste, Universität Rostock

urn:nbn:de:gbv:28-diss2010-0148-9

„In der Logik kam es mir wunderlich vor, dass ich diejenigen Geistesoperationen, die ich von Jugend auf mit der größten Bequemlichkeit verrichtete, so auseinanderzerren, vereinzeln, gleichsam zerstören sollte, um den rechten Gebrauch derselben einzusehen."

Johann Wolfgang von Goethe, Dichtung und Wahrheit

Danksagung

Die vorliegende Dissertation markiert den Endpunkt meiner Forschungstätigkeit an der Arbeitsgruppe Modellierung und Simulation am Institut für Informatik der Fakultät für Informatik und Elektrotechnik der Universität Rostock. Sie entstand im Rahmen des von der Deutschen Forschungsgesellschaft geförderten Graduiertenkollegs MuSAMA.

Mit dieser Danksagung möchte ich mich bei all jenen bedanken, die an der Vollendung der vorliegenden Arbeit in welcher Art und Weise auch immer beteiligt waren.
Der Dank gilt insbesondere und in erster Linie meiner Betreuerin Frau Prof. Dr. rer. nat. habil. Adelinde M. Uhrmacher. Weiterer Dank gilt Herrn Prof. Dr.-Ing. Thomas Kirste als Sprecher des Graduiertenkollegs sowie Frau Prof. Dr. rer. nat. Birgitta König-Ries für hilfreiche und engagierte Kritik an der Arbeit.
Ich danke den Wissenschaftlern, mit denen ich mich per Email zu inhaltlichen Fragen austauschen konnte oder auf Konferenzen interessante Gespräche führen durfte.
Für teils inhaltliche, teils moralische Unterstüzung gilt mein Dank den Mitarbeitern der Arbeitsgruppe Modellierung und Simulation, den Stipendiaten der Graduiertenkollegs MuSAMA und dIEM oSiRiS und weiteren netten und hilfsbereiten Personen aus dem Umfeld der Universität Rostock.
Danke Mattis, Roland, Jan, Carsten, Mathias, Sigrun, Nadja, Christoph, Christiane, Maik, Christian, Henry, Stefan, Petra, Andrea, Hansi, Orianne, Dagmar, Dortje, Wenke.

Ich bedanke mich für ihre ständige Unterstützung bei meiner Familie, bei Sophia.
Ohne euch wäre diese Arbeit nicht entstanden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

B2B Business-to-Business

BPEL4WS Business Process Execution Language for Web Services

CORBA Common Object Request Broker Architecture

DAML DARPA Agent Markup Language

DAML-S DAML-Services

DARPA Defense Advanced Research Projects Agency

DCOM Distributed Component Object Model

DIANE Dienste in ad-hoc Netzen

DPWS Device Profile for Web Services

DSD DIANE Service Description

EAI Enterprise Application Integration

eBNF Erweiterte-Backus-Naur-Form

ebXML Electronic Business using XML

EJB Enterprise Java Beans

ERP Enterprise Resource Planning

GRK Graduiertenkolleg

HTML Hypertext Markup Language

HTN Hierarchical Task Network

ICAPS International Conference on Automated Planning and Scheduling

ICKEPS International Competition on Knowledge Engineering for Planning and Scheduling

IOPE Input Output Precondition Effect

IPC International Planning Competition

KIF Knowledge Interchange Format

KQML Knowledge Query and Manipulation Language

MuSAMA Multimodale Smarte Appliance Ensembles für mobile Applikationen

OSGi Open Services Gateway initiative

OWL Web Ontology Language

OWL-S OWL for Web Services

PE Precondition and Effect

PDDL Planning Domain Description Language

POP Partial Order Plan

QoS Quality of Service

REST Representational State Transfer

RDF Resource Description Framework

RPC Remote Procedure Call

SAWSDL Semantic Annotations for WSDL

SOA Service Oriented Architecture

SOUPA Standard Ontology for Ubiquitous and Pervasive Applications

STRIPS Stanford Research Institute Problem Solver

SWRL Semantic Web Rule Language

TCP/IP Transmission Control Protocol/Internet Protocol

TOP Total Order Plan

UDDI Universal Description, Discovery and Integration

UI User Interface

UML Unified Markup Language

UPnP Universal Plug and Play

URI Unified Ressource Identifier

WADL Web Application Description Language

WSDL Web Service Description Language

WSML Web Service Modelling Language

WSMO Web Service Modelling Ontology

WSMX Web Service Execution Environment

WSRL Web Service Request Language

W3C World Wide Web Consortium

XML Extended Markup Language

XSD XML Schema Definition

Kapitel 1

Einleitung

1.1 Motivation

Seit jeher wird mit Hilfe der Technik versucht, den Lebens- und Arbeitsalltag einfacher und komfortabler zu gestalten. Im Laufe der Zeit wurden immer wieder technologische Entwicklungen eingesetzt, um wiederkehrende, alltägliche Tätigkeiten zu vereinfachen oder zu ersetzen. So hat sich durch die Verwendung von elektronischen Geräten im Haushalt der Lebensstandard der Menschen drastisch erhöht. In den letzten Dekaden kam dabei immer mehr computergestützte Technik zum Einsatz. Angefangen von einfachen Küchengeräten über Unterhaltungselektronik bis hin zu kompletten Raum- und Haussteuerungen erleichtern mittlerweile eine erhebliche Anzahl computergestützter Geräte den Alltag.
Mit jedem neuen Gerät werden aber auch neue Anforderungen an dessen Benutzer gestellt. Neue Kompetenzen sind erforderlich, um die Fähigkeiten des Gerätes zweckmäßig und sicher einzusetzen. Und so wie neue Technologie Arbeit erleichtern soll, führt sie auch immer wieder dazu, dass es Probleme bei der Verwendung dieser Technologie gibt, die im schlimmsten Fall dazu führt, dass der Lebenskomfort sinkt anstatt zu steigen. Ziel bei der Entwicklung neuer Technologien muss daher immer auch die Anwenderfreundlichkeit und ein ausgewogenes Verhältnis zwischen Nutzen und Aufwand sein.
Die Komplexität der Anwendung neuer Technologie steigt für den Benutzer noch stärker, wenn mehrere Geräte zur Umsetzung einer Funktionalität eingesetzt werden müssen oder oft wechselnde Geräte zu verwenden sind.
Intelligente Umgebungen erlauben es, dieses dynamische Zusammenspiel der Geräte zu automatisieren und so die Komplexität des Systems vor dem Nutzer zu verbergen. Damit besteht die Chance die zunehmende Heterogenität und Dynamik der Geräte beherrschbar zu machen und so im Effekt die Lebensqualität zu erhöhen.

1.2 Intelligente Umgebungen

Intelligente Umgebungen bezeichnen räumlich begrenzte Ansammlungen von Geräten, die über eigene Berechnungsressourcen verfügen und miteinander vernetzt sind. Die Zusammensetzung der Geräte ist dynamisch und kann sich über die Zeit ändern. Des Weiteren verfügen intelligente Umgebungen über sensorische Möglichkeiten die Umwelt zu beobachten. Sie sind räumlich begrenzt und unterstützen einen Nutzer in seinen Handlungen darin.
Kirste (2006) beschreibt Smart Environments[1] als:

> [...] physical spaces that are able to react to the activities of users, in a way that assists the users in achieving their objectives in this environment.

Nach dieser Definition verfolgen Nutzer in einer intelligenten Umgebung ein bestimmtes Ziel, welches von der Umgebung erkannt werden kann. Anhand des Zieles ist die Umgebung in der Lage adäquat zu reagieren und dem Nutzer so Assistenz anzubieten.
Wie in vielen wissenschaftlichen Gebieten gibt es auch im Bereich der Intelligenten Umgebungen eine Vielzahl von Begrifflichkeiten, die sich über die Zeit entwickelt haben und teilweise ähnliche und überlappende Sachverhalte beschreiben. Eine klare Trennung der Themengebiete ist oftmals nicht möglich. Im Folgenden soll versucht werden, verschiedene für diese Arbeit relevante Begriffe zu nennen, deren Gemeinsamkeiten und Unterschiede zu klären und herauszuarbeiten, welche Aspekte für die vorliegende Arbeit von Bedeutung sind und zu welchen Visionen sie beitragen können.

Ubiquitous Computing Der Term *Ubiquitous Computing*[2] beschreibt die Allgegenwärtigkeit von Rechenkapazitäten in allen Lebenslagen.
Der Begriff wird von der Mehrzahl der Autoren auf Mark Weisers Aufsatz „The computer for the 21st century“ zurückgeführt (Weiser, 1991). Hierin beschreibt Weiser die Vision von Technologie, die nahtlos in Alltagsgegenstände übergeht und so nicht mehr direkt wahrgenommen wird.

> The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.

Der Nutzer bedient beim Ubiquitous Computing nicht mehr einzelnen Geräte, um Funktionalitäten zu erhalten, sondern die Geräte sind ganz in das intuitive

[1] In dieser Arbeit wird die in englischsprachigen Veröffentlichungen verwendete Bezeichnung *Smart Environment* mit “Intelligente Umgebung” übersetzt.
[2] ubiquitous = allgegenwärtig

Handeln des Nutzers integriert, ohne dass dieser zwangsläufig davon Kenntnis hat.
Seit Weisers Vision differenzierte sich das Gebiet und eine Fokussierung auf verschiedene Teilaspekte der Vision fand statt, was zu neuen Begriffen und Schwerpunkten führte.

Pervasive Computing Eine eher technologische Betrachtung wird durch *Pervasive Computing*[3] beschrieben.
Die Informatik ist mit der sich ständig vergrößernden Rechen- und Speicherkapazität und der damit einhergehenden stetig ansteigenden Komplexität von Hard- und Software konfrontiert. Diese steigende Komplexität zieht einen Gewinn an Leistungsfähigkeit nach sich, der in Kombination mit voranschreitender Miniaturisierung dazu führt, dass heute ein Großteil der Menschheit ständig von Klein- und Kleinstcomputern umgeben ist. Ein Beispiel dafür ist die bereits heute existierende Durchdringung des Alltags durch Mobiltelefone.[4]
Zwar wird auch im Pervasive Computing Weisers Vision verfolgt, allerdings liegt der Fokus verstärkt auf der Interoperabilität[5] und dem reibungslosen Zusammenspiel verschiedener Geräte. Chen u. a. (2004) betonen dabei die räumlich und zeitlich unbegrenzte Verfügbarkeit von Informationen.

> In the pervasive computing vision, computer systems will seamlessly integrate into the life of everyday users, providing them with services and information in an „anywhere, anytime“ fashion.

Neben der Interoperablilität der Geräte wird sich im Rahmen des Pervasive Computing auch der Auswertung der gesammelten Daten gewidmet. Satyanarayanan (2001) betont unter anderem die Bereitstellung von sogenannten *Smart Spaces* und deren Unsichtbarkeit. Diese „intelligenten Räume“ zeichnen sich durch die Verfügbarkeit von Sensoren und der Möglichkeit von einfachen Stimulus-Response-Reaktionen aus. Sobald die vorhandene Technik einen Benutzer nur noch minimal von seiner eigentlichen Tätigkeit ablenkt, ist Unsichtbarkeit oder auch Unaufdringlichkeit erreicht.

[3] pervasive = durchdringend
[4] Nach einer Erhebung der Vereinten Nationen (United Nations Conference on Trade and Development (UNCTAD), 2007) gab es 2006 weltweit 3,3 Mrd. aktive Mobiltelefone. Im Vergleich zu 2004 verdoppelte sich diese Anzahl. Da der Trend zur mobilen Telefonie ungebrochen ist, kann davon ausgegangen werden, dass die Zahl der heute aktiven Mobiltelefone signifikant über dem Niveau von 2006 liegen wird.
[5] Interoperabilität wird als die Fähigkeit von zwei oder mehr Systemen oder Komponenten angesehen, Informationen auszutauschen und diese Informationen zu verwenden (IEEE, 1990).

Ambient Intelligence Während die Umgebungen des Pervasive Computing zwar mit Sensoren ausgestattet sind, die gewonnenen Daten sammeln und analysieren können sowie weiterhin über einfach Stimulus-Response-Regeln auf Änderungen der Umwelt reagieren können, geht die Vision der *Ambient Intelligence*[6] einen Schritt weiter und beschreibt auch komplexere, intelligente Reaktionen.[7]
Aarts und Encarnação (2006) fassen die wesentlichen Merkmale von Ambient Intelligence wie folgt zusammen:

> Ambient Intelligence refers to electronic environments that are sensitive and responsive to the presence of people.

Die Autoren stellen damit die intelligenten Reaktionen auf Menschen in den Vordergrund. Eine weitere Beschreibung, in der einerseits der unaufdringliche Charakter als auch die nötige Beherrschbarkeit zur Geltung kommt, geben Friedewald u. a. (2005):

> It (the environment) would respond in a seamless, unobtrusive and often invisible way, nevertheless remaining under the control of humans.

Allen Beschreibungen gemein ist das Ziel, dem Menschen mit Hilfe der Funktionalität verschiedenster Geräte unaufdringliche Unterstützung anzubieten. Wesentlicher Anspruch an eine neue Technologie in intelligenten Umgebungen muss daher immer Unaufdringlichkeit sein. Dem Nutzer soll geholfen werden, jedoch soll er dabei so wenig wie möglich gestört werden. Weiterhin impliziert die Forderung nach Unaufdringlichkeit in Umgebungen mit heterogenen und wechselnden Geräten eine Entlastung des Nutzers vom Wissen der Funktionalitäten der einzelnen Geräte.

1.3 Semantic Web

Neben den intelligenten Umgebungen sind im Rahmen dieser Arbeit zusätzlich die Fortschritte des *Semantic Web* von Interesse. Dieses ist, wie auch Ubiquitous Computing, Pervasive Computing und Ambient Intelligence eine Vision, die noch immer weit davon entfernt ist, realisiert zu sein. Initiativen

[6] ambient = umgebend

[7] Die Frage, ob die Reaktion einer technischen Umgebung wahrhaft als „intelligent“ bezeichnet werden kann, berührt die Frage nach der generellen Möglichkeit von intelligenten Maschinen. Diese philosophische Kernfrage der Informatik wird auch die Vision der Ambient Intelligence nicht beantworten können. Dennoch stellen sie einen Fortschritt gegenüber reinen stimuluse-response Aktionen dar.

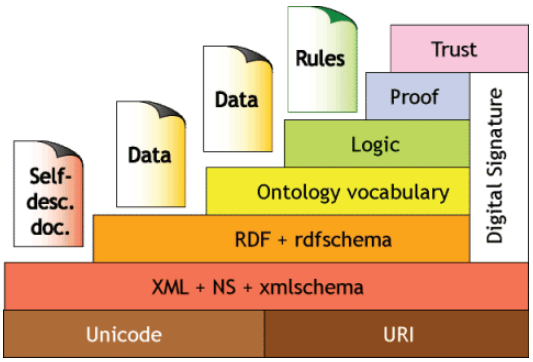

Abbildung 1.1: Schichten des Semantic Web (nach Berners-Lee (2000))

des Semantic Web beschäftigen sich mit den Möglichkeiten der semantischen Beschreibung von und dem Zugriff auf Dienste, die über Netzwerke (zumeist das Internet) zur Verfügung gestellt werden. Ein Dienst zeichnet sich dadurch aus, dass er eine bestimmte Funktionalität über eine wohldefinierte Schnittstelle zur Verfügung stellt. Dienste sind in erster Linie für den Gebrauch durch andere Software vorgesehen. Die Verwendung durch einen Nutzer erfolgt über zusätzliche Schnittstellen wie GUIs aber nicht direkt über die Dienstschnittstelle. Anders als bei intelligenten Umgebungen wird beim Semantic Web die räumliche Komponente vernachlässigt. Dienste sind verfügbar oder nicht, wo sie lokalisiert sind, ist sekundär. Auch die dedizierte Unterstützung des Nutzers ist kein vordergründiges Ziel des Semantic Web, vielmehr wird ein automatisierter und sinnvoller Informationsaustausch zwischen Maschinen bzw. deren Diensten angestrebt.

Der Begriff Semantic Web geht auf einen Artikel von Tim Berners-Lee aus dem Jahr 2000 zurück (Berners-Lee, 2000). Zu dieser Zeit hatte das Internet begonnen sich zu etablieren und unter anderem der Wunsch nach automatischer Durchsuchbarkeit des Netzes nach mehr als nur Schlagworten führte zu Bestrebungen, die verfügbaren Informationen sowie deren Bedeutung, deren Semantik, maschinenlesbar zu machen (Fensel u. a., 2003). Die einzelnen Ebenen der Informationsverarbeitung, denen sich das Semantic Web widmet, sind in Abbildung 1.1 dargestellt.

Beginnend mit der untersten entstanden für die dargestellten Ebenen elaborierte Beschreibungssprachen und Verfahren zu Verarbeitung der hinterlegten Informationen. Mit der Annotation von Information mit Metadaten und deren Beschreibung mittels Ontologien, ermöglicht es das Semantic Web, umfassende Schlussfolgerungen aus bestehendem Wissen zu ziehen und beispielsweise mit KI-Methoden daraus neues Wissen zu generieren.

Bestimmte Funktionalitäten lassen sich erst durch die Kombination von verschiedenen anderen Diensten erreichen. Um diese Dienste zu identifizieren

und sinnvoll miteinander zu kombinieren, können automatische Kompositionsmethoden eingesetzt werden. Entsprechende Forschungsarbeit zur automatischen Komposition von Diensten ist häufig im Gebiet des *Semantic Web* angesiedelt. Die Automatische Dienstekomposition kann der „Logic“-Schicht in Abbildung 1.1 zugeordnet werden. Es ist zu erkennen, dass sie über der „Ontology“-Schicht steht und daher auf Ontologien angewiesen ist.
Die Verwendung der Sprachen und Methoden des Semantic Web in intelligenten Umgebungen ermöglicht es, die von der Vision der Ambient Intelligence geforderte Intelligenz in die Geräte zu bringen. In einer intelligenten Umgebung in der Geräte und Dienste versteckt sind (d.h. sowohl durchdringend als auch allgegenwärtig), weiß der Nutzer unter Umständen gar nicht, dass bestimmte Dienste verfügbar sind. Die automatische Dienstekomposition kann diese Dienste für ihn nutzbar machen, ohne dass er es bemerkt.

These 1 *Intelligente Umgebungen besitzen einen räumlichen Bezug und erfordern Unaufdringlichkeit gegenüber dem Nutzer, was sie von anderen Anwendungsgebieten von Dienstekomposition, wie dem Semantic Web, unterscheidet.*

1.4 MuSAMA

Das Graduiertenkolleg (GRK) Multimodale Smarte Appliance Ensembles für mobile Applikationen (MuSAMA) ist ein an der Universität Rostock angesiedeltes, von der DFG gefördertes Graduiertenkolleg. MuSAMA befasst sich mit Methoden und Werkzeugen zur Realisierung intelligenter Umgebungen. Als spezieller Anwendungsfall wird ein intelligenter Besprechungs- oder Konferenzraum betrachtet. Das GRK ist dazu in die vier in Abbildung 1.2 dargestellten Forschungsschwerpunkte unterteilt.
Die vorliegende Arbeit untersucht die im Forschungsschwerpunkt 3 angesiedelte Strategiesynthese in intelligenten Umgebungen. Ziel der Strategiesynthese ist es, Nutzerassistenz durch Gerätekooperation zu erreichen. Die einzelnen Stufen der Nutzerassistenz sind in Abbildung 1.3 dargestellt.
In einer intelligenten Umgebung sind viele Geräte vorhanden, die verschiedene Dienste anbieten. Viele dieser Dienste sind dem Nutzer unter Umständen nicht bewusst, da sie allgegenwärtig und durchdringend also für den Nutzer transparent zur Verfügung stehen. Dennoch muss der Nutzer diese Dienste in Anspruch nehmen um seine Ziele zu erreichen. Ausgehend von den Zielen des Nutzers ist eine Intentionsanalyse in der Lage, daraus Zielbeschreibungen zu

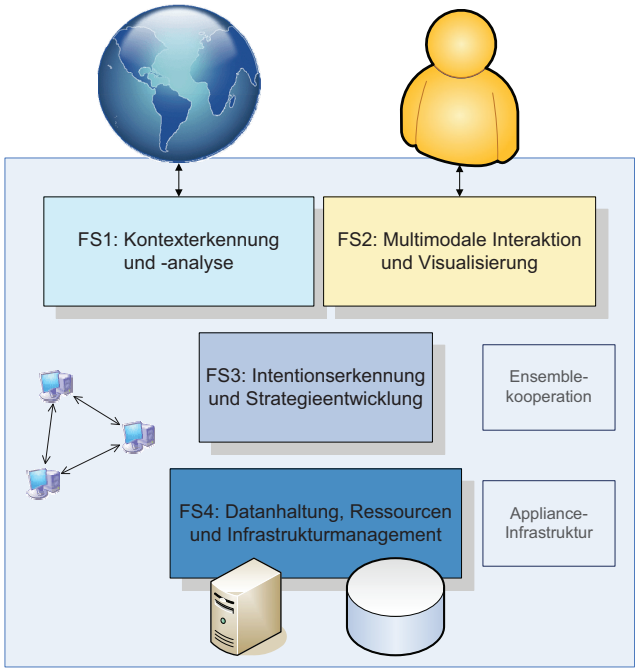

Abbildung 1.2: Forschungsschwerpunkte des GRK MuSAMA.

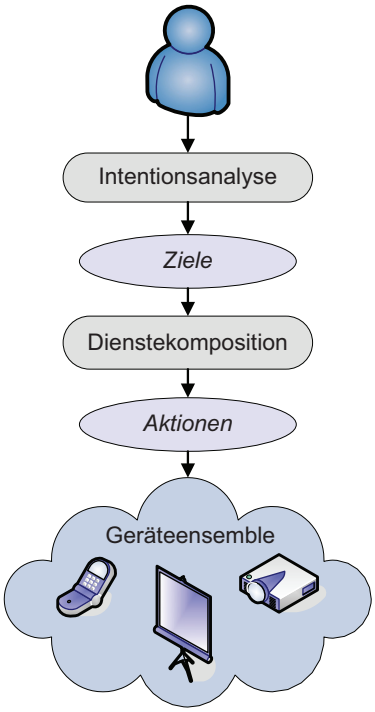

Abbildung 1.3: Nutzerassistenz in einer intelligenten Umgebung

inferieren. Anhand dieser Ziele und Informationen über die Struktur der Umgebung kann die Dienstekomposition eine Reihe von Diensten identifizieren, die zur Erfüllung des gesuchten Ziels benötigt werden. Diese Aktionen werden schließlich in der Umgebung ausgeführt und realisieren so die gewünschte Nutzerassistenz. Auf diese Art und Weise hat der Nutzer Dienste für seine Zwecke verwendet, von deren Existenz er nichts weiß und genau das ist ein Teilaspekt der Vision von intelligenten Umgebungen.
Es wird davon ausgegangen, dass die Geräte der intelligenten Umgebung ihre Funktionalität als Dienste anbieten, so dass die Strategiesynthese durch eine automatisierte Komposition von Diensten realisiert werden kann. Dabei gibt es verschiedene Möglichkeiten diese Komposition zu erreichen. Im Rahmen dieser Arbeit wird sich auf KI-Planung als Werkzeug zur Dienstekomposition konzentriert. Eine umfassende Erläuterung der Möglichkeiten von KI-Planung und eine Betrachtung anderer Methoden zur Komposition wird in den Kapiteln 2 und 3 gegeben.

1.5 Problembeschreibung

Im Rahmen dieser Arbeit sollen die Möglichkeiten der automatischen Komposition von Diensten in intelligenten Umgebungen untersucht werden.
Wesentliche Anforderungen an die Komposition von Diensten in intelligenten Umgebungen sind

- Unaufdringlichkeit
- Fähigkeit zur Berücksichtigung von Ad-hoc-Umgebungen
- Ressourcenschonung

Um der Anforderung an Unaufdringlichkeit zu genügen, muss die Komposition schnell und automatisch, also möglichst ohne Interaktion seitens des Nutzers, erfolgen.
Daneben muss die generelle Tauglichkeit von KI-Planung als Kompositionsmethode in intelligenten Umgebungen untersucht werden. Dazu sollen folgende Punkte betrachtet werden:

- Algorithmen
- Mögliche Sprachen zur Beschreibung
- Modellierung in diesen Sprachen
- Effizienz der Algorithmen

Es ist zu klären, welche Algorithmen zur Verwendung in Frage kommen, welche Sprachen zur Problembeschreibung verwendet werden können und welche Implikationen die Wahl des Algorithmus und der Beschreibungssprache auf die Modellierung der Kompositionsprobleme hat. Welche Facetten der Realität sind abbildbar und welche sind es nicht?
Planung stammt aus dem Bereich der KI und verwendet daher teilweise unterschiedliche Werkzeuge und Sprachen als sie in intelligenten Umgebungen oder im Semantic Web verwendet werden. Im Rahmen der Arbeit ist zu klären, inwieweit Methoden der verschieden Gebiete genutzt werden können.
Letztlich sollen die gewonnenen Erkenntnisse in einer Architektur zusammengefasst werden und die Anwendbarkeit einer solchen Architektur durch eine prototypische Umsetzung von Kernfunktionen der Architektur untermauert werden.

Kapitel 2

Dienstekomposition

2.1 Komponierte Dienste

In einer intelligenten Umgebung stehen verschiedene Geräte mit unterschiedlichen Fähigkeiten zur Verfügung. Diese Fähigkeiten werden über ein verbindendes Netzwerk als Dienste angeboten. Nicht alle Wünsche des Nutzers lassen sich von einem Gerät oder einem Dienst allein erfüllen. Oft ist das Zusammenspiel mehrerer Geräte und Dienste erforderlich. Dieses Zusammenspiel kann durch eine Komposition vorhandener Dienste erreicht werden. Dabei wird durch eine Verknüpfung von Diensten ein komponierter und höherwertiger Dienst zur Verfügung gestellt.
Komposition von Diensten basiert auf den Prinzipien der Service Oriented Architecture (SOA) (Dustdar und Schreiner, 2005). Diese sollen zunächst kurz erläutert werden. Im Anschluss wird ein allgemeines Kompositionsmodell eingeführt, an dem sich im weiteren Verlauf der Arbeit orientiert wird. In der Literatur finden sich verschiedene Methoden zur Dienstekomposition. Sie werden im Abschnitt 2.4 vorgestellt und auf ihre Eignung für den Einsatz in intelligenten Umgebungen anhand des vorgestellten Kompositionsmodells untersucht.

2.2 Serviceorientierte Architekturen

Eine Softwareapplikation besteht grundlegend aus Daten und einer Programmlogik, die diese Daten verarbeitet. Im einfachsten Fall befinden sich beide Teile auf dem selben Rechner. In verteilten Systemen (Tanenbaum, 2003) ist es möglich, die Daten und die Programmlogik einer Applikation voneinander zu trennen. Im nächsten Entwicklungsschritt wurde auch die Programmlogik mit Hilfe netzwerkbasierter Komponenten wie Remote Procedure Call (RPC) auf verschiedene Rechner verteilt. Mit der Einführung von einheitlichen Schnittstellenbeschreibungen von Diensten wird es darüber hinaus möglich, die verteilten Komponenten lose gekoppelt nutzbar zu machen. Durch diese lose gekoppelte Verteilung können Netzwerkeffekte und damit ökonomischer Mehrwert entstehen (Liebowitz und Margolis, 1994). Netzwerkeffekte bilden eine Hauptmotivation für die Etablierung Serviceorientierter Architekturen in der Industrie.
Wesentliche Ziele von SOAs sind Transparenz, Wiederverwendbarkeit und Komponierbarkeit von Prozessen. Hierzu werden die Prozesse in Dienste gekapselt und von ihrem Anbieter (*provider*) in einer Registrierung (*registry*) publiziert (*published*). Dazu müssen die Schnittstellen der Dienste in einer standardisierten Sprache beschrieben sein. Ein Konsument (*consumer*) kann einen benötigten Dienst dann über die Registrierung suchen (*find*). Mit den

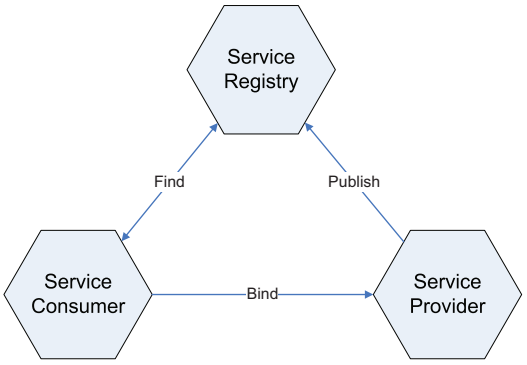

Abbildung 2.1: Komponenten einer Serviceorientierten Architektur und ihr Zusammenspiel

Informationen der Registrierung ist es dem Konsumenten möglich, den Dienst in eigenen Applikationen einzubinden (*bind*) und schließlich zu verwenden (Erl, 2006). In Abbildung 2.1 ist dieses Zusammenspiel dargestellt.

Wenn ein angeforderter Dienst nicht in der Registrierung vorhanden ist, kann versucht werden, die geforderte Funktionalität aus anderen Diensten unter Zuhilfenahme einer Dienstekomposition bereitzustellen. Direkt implementierte Dienste werden atomar genannt. Resultate einer Komposition heißen komponierte Dienste (Alonso u. a., 2003). Der Zugriff auf atomare als auch komponierte Dienste ist für einen Dienstkonsumenten in einer SOA transparent.

Auch wenn SOA ein Konzept ist und keine Angaben über die Art der verwendeten Dienste macht, werden häufig Webservices[1] verwendet, um SOAs zu realisieren. Webservices stellen über das Internet verfügbare Applikationen dar. Sie sind nachrichtenbasiert. Die ausgetauschten Nachrichten werden überwiegend mit Hilfe von SOAP beschrieben.[2]

Neben SOAP existieren noch weitere Möglichkeiten, um Webservices anzusprechen, wie die sogenannten Representational State Transfer (REST)-basierten Webservices. Dabei werden HTTP-GET-Methoden verwendet, um Anfragen an einen Dienst zu übertragen. Auf eine Verwendung von SOAP-Envelopes wird verzichtet. Der große Vorteil dabei ist der geringe Datenüberschuss im Verhältnis zu den Nutzdaten (overhead). Allerdings ist die Länge einer GET-Anfrage beschränkt und abhängig von der Implementierung des jeweiligen Webservers. Eine standardisierte Mindestlänge, die von jedem Webserver un-

[1] In dieser Arbeit wird die zusammengesetzte eingedeutschte Schreibweise „Webservice“ verwendet. In der Literatur finden sich auch weitere Schreibweisen, wie zum Beispiel „Web Service“ oder „Web-Service“.

[2] Während das Akronym SOAP in Box u. a. (2000) noch als Simple Object Access Protocol bezeichnet wurde, wird in den Entwürfen zur Version 1.2 (vgl. (W3C, 2007a)) keine Ausschreibung des Akronyms mehr verwendet.

terstützt wird, gibt es nicht (Fielding u. a., 1999). Daher ist es mit dieser Methode u.U. problematisch, sehr lange Anfragen an Dienste zu formulieren.

2.3 Allgemeines Kompositionsmodell

Es gibt eine große Zahl an Übersichtspapieren, die sich mit teilweise verschiedenen Schwerpunkten der Dienstekomposition befassen (Milanovic und Malek, 2004; Rao und Su, 2004; Dustdar und Schreiner, 2005; Hull und Su, 2005; Küster u. a., 2005; Fluegge u. a., 2006; ter Beek u. a., 2006; Alamri u. a., 2006; Bronsted u. a., 2007; Agarwal u. a., 2008). Dabei werden unterschiedliche Termini verwendet. Es ist nötig, die im weiteren Verlauf der Arbeit verwendeten Begriffe für einen einheitlichen Gebrauch in Bezug auf die Komposition von Diensten zu definieren.

Definition 1 (Prozessmodell) *Ein Prozessmodell ist die formale Beschreibung, wie und wann Dienste einer Komposition ausgeführt werden. Es kann als gerichteter, in der Regel azyklischer Graph, bestehend aus miteinander verknüpften Dienstbeschreibungen, angesehen werden. Knoten dieses Graphes stellen Aufrufe des jeweiligen Dienstes dar. Die Kanten repräsentieren deren zeitliche Abfolge.*

Definition 2 (Aktionssequenz) *Eine Aktionssequenz ist eine Instanz eines Prozessmodells. Dabei sind den einzelnen Dienstebeschreibungen reale Dienste zugeordnet. Eine Aktionssequenz ist mit Hilfe einer entsprechenden Steuerungskomponente direkt ausführbar.*

Mit den Definitionen des Prozessmodells und der Aktionssequenz wird eine klare Trennlinie zwischen Strategiesynthese und Aktionsausführung gezogen (vgl. Abbildung 1.3). In der Literatur wird im Zusammenhang mit Kompositionen häufig die Bezeichnung Plan verwendet (Ponnekanti und Fox, 2002). Allerdings ist diese Bezeichnung hinsichtlich der hier gemachten Unterscheidung zwischen Prozessmodell und Aktionssequenz nicht vorteilhaft. Ein Plan im Sinne der KI-Planung (siehe Sektion 3.1) ist lediglich eine Möglichkeit der Darstellung des Prozessmodells.
Im weiteren Verlauf dieses Abschnittes wird ausgehend von Beschreibungen in der Literatur ein allgemeines Kompositionsmodell entworfen, das alle wesentlichen Facetten von Dienstekompositionen in intelligenten Umgebungen abdeckt. Das Kompositionsmodell umfasst zunächst die verschiedenen Kompositionsphasen. Weiterhin sind der Zeitpunkt der Kompositionsphasen und die Frage, ob Nutzerinteraktion stattfindet, von Interesse. Schließlich ist die Formalisierung der verwendeten Dienste, des Prozessmodells und der Aktionssequenz ein wesentliches Merkmal des Kompositionsmodells.

2.3.1 Kompositionsphasen

In Fluegge u. a. (2006) werden vier Phasen der Komposition unterschieden: 1. Generierung eines Prozessmodells, 2. Auffinden der vorhandenen Dienste, 3. Publizieren des komponierten Dienstes und 4. Management des Datenflusses während der Ausführung der Komposition. In der ersten Phase wird das Prozessmodell erstellt (*composition*). Dabei kann ausgehend von vorhandenen Dienstbeschreibungen und einer Zielvorgabe eine automatische Kompositionsmethode das Modell erstellen. Auch eine manuelle Erstellung ohne weiteres Vorwissen ist denkbar. Das Prozessmodell kann abhängig von der Kompositionsmethode sehr unterschiedlich strukturiert sein. In jedem Fall wird es eine zeitliche Abfolge von Dienstaufrufen enthalten. Die Generierung des Prozessmodells steht im Rahmen dieser Arbeit im Mittelpunkt der Betrachtungen. In der zweiten Phase, dem Auffinden der vorhandenen Dienste (*discovery*), werden ausgehend von den im Prozessmodell enthaltenen Dienstbeschreibungen, reale Dienste gesucht, auf die diese Beschreibungen zutreffen. Wie die Mehrzahl der Veröffentlichungen zum Thema Dienstekomposition, wird auch bei Fluegge u. a. (2006) die Komposition von Webservices betrachtet. Dabei ist die angenommene Gesamtzahl der verfügbaren Dienste sehr viel größer als im Bereich der intelligenten Umgebungen. Ausgehend von einer Umfrage (Stand 2007) (Marquardt u. a., 2008) ist für die Anzahl der beteiligten Dienste im Bereich der intelligenten Umgebungen mit einer Größenordnung im zweistelligen Bereich zu rechnen (siehe Abschnitt 5.1.1). Obwohl davon auszugehen ist, dass diese Anzahl steigt (Papazoglou u. a., 2006), wird die Gesamtzahl der verfügbaren Dienste in einer intelligenten Umgebung auch zukünftig erheblich kleiner sein, als die angenommenen Dienstemengen für die Komposition von Webservices im Internet. Aus diesem Grund ist es für die Komposition in intelligenten Umgebungen möglich und sinnvoll, den *discovery*-Schritt vor der Generierung des Prozessmodells auszuführen, um das Prozessmodell nur auf Grundlage der verfügbaren Dienste zu entwerfen. Um den komponierten Dienst ausführen zu können, muss er verfügbar gemacht werden (*deployment*). Dazu müssen die real vorhandenen Dienste in das Prozessmodell eingebunden werden und der Dienst braucht eine physikalische Repräsentation (beispielsweise über einen Unified Ressource Identifier (URI)). Dies geschieht in der Regel innerhalb eines sogenannten Servicecontainers. Soll der Dienst ausgeführt werden, wird die Adresse des Servicecontainers, ergänzt um den Namen des Dienstes, als Endpunkt verwendet. Auch der letzte Punkt des allgemeinen Kompositionsmodells von Fluegge u. a. (2006), Management des Datenflusses (*management*), ist der Anwendungsdomäne der Webservices geschuldet. Die Dauer eines Businessprozesses innerhalb einer Komposition kann sich mitunter auf mehrere Tage erstrecken. Des Weiteren nimmt auch der vorange-

gangene Kompositionsprozess (2.) Zeit in Anspruch. In dieser Zeit kann sich die während des discovery-Schrittes (1.) ermittelte Verfügbarkeit der an der Komposition beteiligten Dienste ändern. Um darauf reagieren zu können, muss während der Ausführung der Komposition eine Managementkomponente den aktuellen Zustand der beteiligten Dienste überwachen und eingreifen, falls es zu Inkonsistenzen oder Konflikten während der Ausführung kommt.
Srivastava und Koehler (2004) unterteilen eine Komposition in drei Phasen. 1. Workflow Komposition, 2. Workflow Instanziierung und 3. Ausführung des komponierten Dienstes. Das Ergebnis der ersten Phase nennen sie „abstract workflow“, was mit dem hier beschriebenen Prozessmodell gleichzusetzen ist. In der zweiten Phase wird daraus ein „executable workflow“, in dieser Arbeit Aktionssequenz genannt, generiert.
Das in Rao und Su (2004) beschriebene Kompositionsmodell besteht aus fünf Schritten; 1. Dienstrepräsentation, 2. Übersetzung, 3. Prozessgenerierung, 4. Evaluierung und 5. Ausführung. Die ersten beiden Schritte sind eine differenziertere Beschreibung des ersten Schrittes von Fluegge u. a. (2006). Die aufgeführte Evaluierung (4.) ist bei Fluegge u. a. (2006) nicht genannt und erlaubt eine Auswahl des gefundenen Prozessmodells nach nicht-funktionalen Gesichtspunkten, wie Kosten, Dauer, Qualität (Quality of Service (QoS)). Wenngleich der Schritt (*management*) von Fluegge u. a. (2006) die Ausführung des komponierten Dienstes impliziert, ist die Bezeichnung „Ausführung“ (*execution*) von Rao und Su (2004) passender.
Hull und Su (2005) unterscheiden während einer Komposition lediglich die zwei Phasen: Synthese und Ausführung. Bei der Synthese wird das Prozessmodell, bestehend aus formalen Dienstbeschreibungen, erstellt. Durch ein Abgleichen (*matching*) der Dienstbeschreibungen mit den realen verfügbaren Diensten, wird das Prozessmodell ausführbar gemacht.
Es ist zu sehen, dass die Unterscheidung in Generierung des Prozessmodells und dessen anschließende Instanziierung von allen Autoren in irgendeiner Form berücksichtigt wird. Diese beiden Punkte bilden daher den Kern einer Komposition.
Als Grundlage für die vorliegende Arbeit soll eine leicht abgewandelte Form der Beschreibung von Fluegge u. a. (2006) verwendet werden, da auch die dort beschriebenen Punkte *discovery* und *execution* im weiteren Verlauf der Arbeit angesprochen werden.

1. Auffinden der vorhandenen Dienste (*discovery*)

2. Generierung eines Prozessmodells (*composition*)

3. Instanziierung des Prozessmodells (*deployment*)

4. Ausführung des komponierten Dienstes (*execution*)

In der ersten Phase wird festgestellt, welche Dienste in der Umgebung vorhanden sind. In dieser Phase wird zudem die Intention des Nutzers und der gegenwärtige Zustand der Umgebung ermittelt. Während der zweiten Phase wird das Prozessmodell erstellt. In der anschließenden *deployment*-Phase wird das Prozessmodell mit Hilfe der im ersten Schritt gefundenen Dienste instanziiert. Die so erzeugte Aktionssequenz kann dann im letzten Schritt ausgeführt werden.
Hull und Su (2005) bezeichnen „the automated discovery, composition, enactment and monitoring“ als vordergründiges und langfristiges Ziel der Forschung auf dem Gebiet der Dienstekomposition und beschreiben damit vergleichbare Kompositionsphasen.
Im Rahmen dieser Arbeit wird sich vor allem auf die Generierung des Prozessmodells und weniger auf die Ausführung der Aktionssequenz konzentriert.

2.3.2 Tempus und Modus

In der Literatur werden verschiedene Merkmale zur zeitlichen Klassifikation von Kompositionsmethoden verwendet. Fluegge u. a. (2006) gehen dabei auf den gesamten Kompositionsprozess ein. Sie schlagen eine zweidimensionale Klassifikation von Kompositionsmethoden vor und orientieren sich am Zeitpunkt der Instanziierung sowie an der Art der Erzeugung des Prozessmodells. Kompositionsverfahren, deren Prozessmodell automatisch erstellt und spät instanziiert wird, werden dynamisch genannt. Verfahren mit manuell erstellten und früh instanziierten Prozessmodellen werden als statisch bezeichnet. Alle Mischformen dazwischen bezeichnen Fluegge u. a. (2006) als semi-dynamische Komposition. Diese Einteilung ist zu einseitig, da Verfahren, bei denen das Prozessmodell noch zur Laufzeit manuell manipuliert werden kann (Casati und Shan, 2001), nach dieser Klassifikation unter die statischen Kompositionsverfahren fallen, obwohl sie ein hohes Maß an Dynamik besitzen.
Eine weitere Klassifikation für Kompositionsmethoden wird von ter Beek u. a. (2006) beschrieben. Auch sie unterscheiden zunächst zwischen statischen und dynamischen Kompositionsmodellen, wobei sich die Bedeutung von statisch und dynamisch zu der von Fluegge u. a. (2006) unterscheidet. Statische Komposition wird in Orchestrierung und Choreographie unterschieden. Dabei wird bei der Orchestrierung eine zentrale Koordinationskomponente eingeführt, die den Informationsaustausch zwischen den Diensten organisiert und überwacht. Eine Choreographie verzichtet auf eine zentrale Komponente und definiert Regeln für den dezentralen Datenaustausch zwischen den Diensten. Ter Beek u. a. (2006) gehen sowohl für Orchestrierung als auch für die Choreographie

davon aus, dass das Prozessmodell zur Laufzeit bekannt sein muss. Es ist festzuhalten, dass diese strikte Einordnung der Orchestrierung und der Choreographie zu den statischen Kompositionsmethoden bei ter Beek u. a. (2006) auf architektonischen Merkmalen beruht. Ob eine Kompositionsmethode statisch oder dynamisch ist, hängt jedoch wesentlich vom Zeitpunkt der Generierung sowie vom Zeitpunkt der Instanziierung des Prozessmodells ab, nicht aber unmittelbar von der verwendeten Architektur. Eine genauere Betrachtung der architektonischen Anforderungen von Komposition wird in Abschnitt 6.1 gegeben eine zusätzliche Beschreibung von Choreographie und Orchestrierung befindet sich in Abschnitt 6.2.
Auch Klusch (2008) unterscheidet statische und dynamische Kompositionsmethoden. Methoden, die Generierung und Ausführung der Komposition strikt voneinander trennen, werden hier statisch genannt. Kompositionsmethoden, bei denen die Phasen der Plangenerierung und -ausführung der Komposition verschränkt sind, nennt Klusch (2008) dynamisch. Wesentliches Argument für eine solche Verschränkung ist die damit gewonnene Flexibilität, auf Änderungen in der Dienstelandschaft bzw. auf nicht deterministische Rückgaben von Diensteaufrufen eingehen zu können.
Im folgenden werden zwei Dimensionen (Tempus und Modus) zur Einordnung von Kompositionsmethoden definiert.

Tempus Aufgrund der Unterscheidungen zwischen Prozessmodell und Aktionssequenz werden im Rahmen dieser Arbeit die Zeitpunkte deren Erstellung betrachtet. Mögliche Zeitpunkte beginnen mit dem Entwurf des Systems und reichen bis zum Zeitpunkt der Ausführung des Systems. Je nach Zeitpunkt wird zwischen **dynamischer** und **statischer** Komposition unterschieden. Eine Komposition ist um so dynamischer, je näher zur Ausführung das Prozessmodell bzw. die Aktionssequenz erstellt wird. Diese Definition unterscheidet sich zu der von Fluegge u. a. (2006), da dort bei der Einordnung in statische und dynamische Verfahren nicht zwischen der Erstellung von Prozessmodell und Aktionssequenz unterschieden wird. Dynamik entscheidet maßgeblich über die möglichen Einsatzgebiete der Kompositionsmethode. Intelligente Umgebungen mit Gerätekonstellationen, die nicht vorhersehbar sind und stark fluktuieren können, verlangen eine dynamische Synthese des Prozessmodells sowie eine dynamische Instanziierung des Prozessmodells. Dabei bindet eine dynamische Instanziierung Dienste erst direkt vor ihrer Ausführung in das Prozessmodell ein. Je mehr Zeit zwischen dem Finden eines Dienstes und dessen wirklicher Verwendung vergeht, desto größer ist die Gefahr, dass sich die Verfügbarkeit des Dienstes ändert. Da sich die Verfügbarkeit von Diensten innerhalb einer Umgebung jederzeit ändern kann, ist eine späte also dynamische Instanziierung für Kompositionen in intelligenten Umgebungen vorteilhaft.

Modus Der temporalen kann zusätzlich noch eine modale Klassifikation hinzugefügt werden. Modal meint hier die Art und Weise, wie Entscheidungen während der Generierung des Prozessmodells bzw. dessen Instanziierung getroffen werden. Mögliche Modi sind **automatisch** oder **manuell**. Ist eine Interaktion seitens des Nutzers notwendig, heißt eine Kompositionsmethode manuell, kommt sie ohne direkte Hilfe eines Nutzers aus, heißt sie automatisch. Neben den beiden Extremen gibt es Mischformen (semi-automatisch), bei denen dem Nutzer beispielsweise Vorschläge gemacht werden oder der Nutzer nur bei unsicheren Entscheidungen eingreifen muss.

Es ist zu beachten, dass eine manuelle Erzeugung des Prozessmodells nicht zwangsläufig statisch sein muss. Kann das Prozessmodell beispielsweise manuell noch zur Laufzeit des Systems verändert werden, handelt es sich um eine dynamische Komposition, obwohl die Änderungen durch einen Benutzer vorgenommen werden. Genauso ist es denkbar, dass zur Laufzeit bereits mehrere Prozessmodelle instanziiert sind und während der Laufzeit dynamisch eine dieser Instanzen zur Ausführung ausgewählt wird. Die Komposition wäre damit statisch (in beiderlei Hinsicht), eine automatische Auswahl der Aktionssequenz wäre dennoch möglich.

Um die gewünschte Unaufdringlichkeit in einer intelligenten Umgebung zu erreichen, sollte der Modus einer Komposition möglichst automatisch sein. Anderenfalls wäre der Nutzer gezwungen, in irgendeiner Art und Weise bewusst in den Kompositionsprozess einzugreifen.

2.3.3 Formalisierung

Neben den Phasen sowie Tempus und Modus einer Komposition muss weiterhin die verwendete Formalisierung der Daten innerhalb der einzelnen Kompositionsschritte betrachtet werden. In Alonso u. a. (2003) wird Komposition stärker über die verwendeten Formalisierungen beschrieben. Es werden die nachstehend aufgelisteten Aspekte genannt, zu denen jeweils eine Sprache erwähnt wird, die die entsprechende Funktionalität anbietet. Eine genauere Betrachtung der Sprachen, die für diese Arbeit von Interesse sind, folgt im Kapitel 3.3.

1. Das Komponentenmodell, welches die Art der zu komponierenden Elemente beschreibt (z.B. Web Service Description Language (WSDL)).

2. Das Orchestrierungsmodell, welches die Sprache definiert, mit der das Prozessmodell beschrieben wird (z.B. Business Process Execution Language for Web Services (BPEL4WS)).

3. Das Daten- und Datenzugriffsmodell, in dem die verwendeten Datentypen definiert werden (z.B. Extended Markup Language (XML)-Schema).

4. Das Dienstauswahlmodell, in dem *discovery, binding* und *matching* der Komponenten beschrieben werden (z.B. Universal Description, Discovery and Integration (UDDI), Electronic Business using XML (ebXML) Registry).

5. Das Transaktionsmodell, welches für die Beschreibung des Datenflusses zur Ausführung der Komposition verwendet wird (z.B. WS-Transaction).

6. Die Fehlerbehandlung, die regelt, wie auf mögliche Fehler bei der Dienstausführung reagiert werden kann.

Auch hier lassen sich die vier für eine Komposition als wesentlich herausgearbeiteten Phasen *discovery, composition, deployment* und *execution* wiederfinden. Die nötigen Formalisierungen für die 2. Phase, der Generierung des Prozessmodells, werden durch das Komponentenmodell und das Orchestrierungsmodell definiert und sind für diese Arbeit von Interesse. Der vierte Punkt, Dienstauswahlmodell, ist für die erste Kompositionsphase (*discovery*) wesentlich. Auch wenn es Überschneidungen gibt, konzentrieren sich die übrigen Punkte überwiegend auf Probleme während der Ausführung von Kompositionen (*execution*) und werden nicht näher erläutert, der interessierte Leser wird auf Alonso u. a. (2003, S.256ff) verwiesen.
In der Literatur finden sich weitere Einordnungen von Kompositionsmethoden. Milanovic und Malek (2004) nennen vier Anforderungen an Kompositionsmethoden. Die Erreichbarkeit von Diensten (Konnektivität), die Beschreibung nichtfunktionaler Qualitätsmerkmale, die Korrektheit der Komposition und die Skalierbarkeit. Im Rahmen dieser Arbeit wird die Erreichbarkeit der Dienste vorausgesetzt. Nichtfunktionale Qualitätsmerkmale (QoS) von Diensten werden im Rahmen dieser Arbeit nicht vorrangig betrachtet. Werden Algorithmen zur Komposition verwendet, die nachweislich korrekt arbeiten, kann davon ausgegangen werden, dass auch die resultierenden Prozessmodelle korrekt sind. Eine Betrachtung hinsichtlich der Korrektheit der Komposition ist zwar immer noch von der Korrektheit der verwendeten Beschreibungssprachen und der Korrektheit und Vollständigkeit der Beschreibung abhängig. So finden sich in der Literatur sogar Aussagen, die aufgrund der großen Komplexität eine vollständige Korrektheit von Kompositionen als generell nicht erreichbar erachten und lediglich Annäherungen für realistisch halten (Klusch, 2008). Dennoch wird im Rahmen dieser Arbeit von einer eindeutigen und korrekten Kompositionsmethode ausgegangen.
Die letzte von Milanovic und Malek (2004) genannte Anforderung, die Skalierbarkeit der verwendeten Methoden, wird im Kapitel 5 näher betrachtet. In

ihrer Veröffentlichung beschränken sich die Autoren weiterhin auf Kompositionsmethoden, bei denen das Prozessmodell zur Laufzeit bereits vorhanden ist und erwähnen die Möglichkeit der automatischen Komposition lediglich für Modelchecking- und Finite-State-Machine-basierte Ansätze.
Agarwal u. a. (2008) orientieren sich bei ihrer Einordnung stark an der verwendeten Architektur. Sie unterscheiden zwischen überlappenden, monolithischen, abgestuften und template-basierten Kompositionen. Im Abschnitt 6.1, der die in dieser Arbeit vorgestellte Architektur beschreibt, wird näher auf die Einordnung von Agarwal u. a. (2008) eingegangen.

2.3.3.1 Komponentenmodell

Das Komponentenmodell beschreibt die Art und Weise, wie die Elemente, die zu komponieren sind, formalisiert werden. Komponenten sind in diesem Fall die Dienste der intelligenten Umgebung. Es gibt verschiedene Möglichkeiten, Dienste zu beschreiben. Hull und Su (2005) unterscheiden die drei Arten nachrichten-, aktivitäts- oder ereignisbasierte Beschreibung. Klein u. a. (2005) unterscheiden nachrichten- und zustandsbasierte Dienstbeschreibungen. Da der überwiegende Anteil der Veröffentlichungen, die sich mit der Komposition von Diensten beschäftigt aus dem Umfeld der nachrichtenbasierten Webservices stammt, ist es sinnvoll, sich zunächst mit dieser Art der Beschreibungen zu beschäftigen.
Der de-facto Standard zur Beschreibung eines Webservices ist WSDL (W3C, 2007b). WSDL ist nachrichtenbasiert und beschreibt die Schnittstelle des Dienstes und damit die Syntax der Nachrichten, die der Dienst austauscht. Die Implementierung des Dienstes bleibt so für den Konsumenten, dem lediglich die Schnittstellenbeschreibung bekannt ist, transparent (siehe Abschnitt 2.2). Eine rein syntaktische Dienstbeschreibung ist für eine Komposition jedoch nicht ausreichend, da diese darüber hinaus auf Semantik angewiesen ist (Sirin u. a., 2003; Benatallah u. a., 2003; Srivastava und Koehler, 2003; ter Beek u. a., 2006). Eine semantische Erweiterung von WSDL um Verknüpfungen zu Ontologien leistet z.B. Semantic Annotations for WSDL (SAWSDL) (Kopecký u. a., 2007). Dabei werden die Namen von Ports und die verwendeten Datentypen mit Ontologien verknüpft. Diese einfache semantische Anreicherung ist jedoch noch nicht ausreichend. Um umfassende automatische Komposition zu ermöglichen, ist es nötig, neben den Schnittstellen auch das Verhalten und die Funktionsweise des Dienstes zu formalisieren. Dafür wurden im Bereich der Webservices umfangreichere Beschreibungssprachen, wie OWL for Web Services (OWL-S) (Martin u. a., 2004), Web Service Modelling Ontology (WSMO) (Lausen u. a., 2005) oder DIANE Service Description (DSD) (Klein, 2004) entwickelt. Sie bieten neben der semantischen Beschrei-

bung der Schnittstellen die Möglichkeit, Prozessmodelle, Vorbedingungen und Effekte von Diensten zu hinterlegen, so kann beschrieben werden, welche Funktionalität der Dienst hat, wie diese Funktionalität erreicht wird und welche Auswirkungen das auf seine Umgebung hat. Im Abschnitt 3.3 wird detaillierter auf die Merkmale der verschiedenen Sprachen zur Dienstbeschreibung eingegangen.
Neben der Formalisierung von Diensten in dedizierten Beschreibungssprachen werden auch andere Ansätze verwendet, die meist aus pragmatischen Gründen gewählt werden, da die jeweils verwendete Kompositionstechnik inhärent einen bestimmten Formalismus unterstützt. Für eine reale Anwendung ist es in solchen Fällen nötig, einen Übersetzer zu verwenden, der die Dienstbeschreibungen in die jeweils vom Kompositionsalgorithmus verwendete Repräsentation umwandelt.
Einer solcher Ansätze ist die Formalisierung auf Grundlage von (Beschreibungs-)Logiken. Dabei werden die Komponenten mit Hilfe der Elemente der entsprechenden Logik formuliert. Meist wird von einer zustandsbasierten Sicht auf die Dienste ausgegangen. Eine Komposition wird durch Anwenden von Schlussregeln aus der entsprechenden Logik erreicht. Um vom resultierenden Prozessmodell zu einer ausführbaren Verkettung von Diensten zu gelangen, muss der resultierende Schluss in Dienstaufrufe und Beschreibungen des Datenflusses zwischen den Diensten übersetzt werden (Klein u. a., 2005). Eine grundlegende Einschränkung von allen logikbasierten Komponentenmodellen ist die Beschränkung auf Variablen, die im überwiegenden Fall lediglich die Wahrheitswerte **true** und **false** annehmen können. Sobald Zahlen und arithmetische Operationen benötigt werden, ist eine Formalisierung der Welt durch Wahrheitswerte unhandlich.
Eine einfache logikbasierte Art der Modellierung ist die Beschreibung von Diensten mit Hilfe von Hornklauseln (Ponnekanti und Fox, 2002). Die Komposition erfolgt durch aussagenlogisches Schließen. Der resultierende Schluss entspricht der Komposition der Dienste. Wenngleich diese Art der Formalisierung sehr elegant ist, ist die Formulierung aller für eine Komposition wesentlichen Informationen über einen Dienst als Hornklausel unintuitiv und unübersichtlich. Es müsste ein Übersetzer zum Einsatz kommen, der in der Lage ist, Beschreibungen von Diensten in der jeweiligen Beschreibungssprache automatisiert in Hornklauseln zu übersetzen. Dabei müsste berücksichtigt werden, dass die meisten Beschreibungssprachen eine größere Ausdrucksmächtigkeit als Hornklauseln haben. Ein solcher Übersetzer konnte in der Literatur nicht gefunden werden.
Neben der Aussagenlogik werden weitere Logiken verwendet. Kompositionsverfahren, die auf KI-Planung beruhen (Carman u. a., 2003; Peer, 2005), verwenden zumeist Planning Domain Description Language (PDDL) (Ghallab

u. a., 1998) (siehe Abschnitt 3.1) als Modellierungssprache. Sie lässt mit Erweiterungen prädikatenlogische Ausdrücke zu. Eine detaillierte Beschreibung zur Verwendung von PDDL für die Formalisierung von Diensten wird im Abschnitt 3.1.4 gegeben.
In Hamadi und Benatallah (2003); Zhang u. a. (2004) werden Petri-Netz-basierte Modelle zur Repräsentation von Webservices vorgestellt. Petri-Netze sind bipartite Graphen bestehend aus Stellen und Transitionen. Stellen repräsentieren Dienste. Transitionen beschreiben die Ein- und Ausgänge eines Dienstes. Sind alle Plätze eines Dienstes besetzt, wird er ausgeführt (die Transition feuert) und die Ausgänge des Dienstes werden mit Markern besetzt. Durch deren wohldefinierte formale Semantik (Reisig und Rozenberg, 1998) ermöglichen es Petri-Netze, dead-locks zu erkennen oder die Erreichbarkeit des Zielzustandes automatisch zu verifizieren (Mecella u. a., 2002). Hamadi und Benatallah (2003) definieren eine Algebra für die häufigsten Kompositionspatterns (z.B. Sequenz, Alternative, Iteration, Parallele Ausführung). Ein konkreter automatisierter Ansatz zur Komposition von Diensten mit Hilfe dieser Darstellung wird nicht gegeben. Vielmehr dienen Petri-Netz-basierte Prozessmodelle zur Verifikation der Funktionalität von Kompositionen, beispielsweise durch das Entdecken von Inkonsistenzen, nicht aber zur automatisierten Erstellung von Prozessmodellen.
In McIlraith (2002); Sheshagiri u. a. (2003); Wu u. a. (2003); Sirin u. a. (2004); Amigoni u. a. (2005) sind kleine Teile des Orchestrierungsmodells im Komponentenmodell enthalten. Diese Verfahren stellen eine Synthese zwischen schnittstellenbasierter Darstellung und der Darstellung als Prozess (Sirin u. a., 2004) dar. Dabei wird den reinen Dienstbeschreibungen domänenspezifisches Kontrollwissen beigefügt, das als Teil eines Prozessmodells, in dem der beschriebene Dienst vorkommen kann, zu verstehen ist. Zur Formulierung dieser Prozessmodellteile werden beispielsweise OWL-S Prozessbeschreibungen oder Hierarchical Task Network (HTN)-Methoden (siehe Abschnitt 3.1.3) verwendet.

2.3.3.2 Orchestrierungsmodell

Im Orchestrierungsmodell ist die Formalisierung des Prozessmodells festgelegt. Prozessmodelle können durch verschiedene Formalismen, wie beispielsweise state charts (Wodtke und Weikum, 1997; Thoene u. a., 2003), Petri-Netze (van der Aalst, 1998; Mecella u. a., 2002), graphische Aktivitätshierarchien (Cass u. a., 2000) oder als Plan im Sinne der KI-Planung (Peer, 2005) dargestellt werden. Jeder dieser Formalismen bringt spezifische Vor- und Nachteile mit sich. Die Möglichkeiten der verschiedenen Formalismen zur automatischen Erzeugung des Prozessmodells sind für diese Arbeit interessant und

sollen für einige Methoden genauer betrachtet werden.
Eine Möglichkeit der Beschreibung des Orchestrierungsmodells bieten Unified Markup Language (UML) State Charts. Ein Vorteil bei der Verwendung von UML ist dessen breite Akzeptanz aufgrund der Standardisierung durch die ISO (aktuelle Version 1.4.2 (ISO, 2005)). Aufgrund des hohen Abstraktionslevels und der unscharfen Formalisierung (O'Keefe, 2006) ist eine direkte und automatisierte Übersetzung eines UML State Charts oder Aktivitätsdiagramms in eine Aktionssequenz allerdings nicht ohne Weiteres möglich. UML dient vorrangig der visuellen Darstellung und dem manuellen Entwurf von Prozessmodellen. Ähnliches gilt für die Darstellung in Aktivitätshierarchien (Cass u. a., 2000) mit dem Unterschied, dass es sich dabei im Gegensatz zu UML nicht um eine standardisierte Darstellung handelt. Sowohl UML State Charts als auch Aktivitätshierarchien können nicht im benötigten Umfang automatisch erzeugt, sondern müssen manuell erstellt werden.
Eine weit verbreitete Methode zur Formalisierung des Prozessmodells sind workflowbasierte Beschreibungssprachen (van der Aalst, 1998) wie BPEL4WS (Andrews u. a., 2003). Darin ist eine große Anzahl an Mustern (patterns) (z.B. Sequence, Parallel Split, Synchronization) zur Beschreibung des Daten- und Kontrollflusses einer Komposition definiert (van der Aalst u. a., 2003). Durch diese Vielfalt ist es möglich, sehr elaboriertes Verhalten auszudrücken. Gleichzeitig wird es mit wachsender Komplexität immer schwieriger automatische Verfahren zur Erzeugung von Prozessmodellen zu entwickeln. Wird die Komplexität vom Orchestrierungsmodell in das Komponentenmodell verlagert, wird die automatische Generierung einfacher. Allerdings steigen dann die Anfordungen an die Modellierung der Komponenten. In der Realität ist ein Großteil der workflowbasierten Kompositionsbemühungen aus der Literatur als statisch in Bezug auf die Erstellung des Prozessmodells anzusehen (Fluegge u. a., 2006).
Kommen Methoden der KI-Planung bei der Komposition zum Einsatz, entspricht das Ergebnis der Komposition (also das Prozessmodell) dem Ergebnis der verwendeten Planungsmethode. Die einfachste Variante ist hier ein geordneter Plan (Total Order Plan (TOP)), der einer strikt sequenziellen Ausführung der Dienste entspricht. Eine Erweiterung dazu ist ein partiell geordneter Plan (Partial Order Plan (POP)). Hier sind auch parallele Ausführungen von Diensten möglich. Mit konditionalen Plänen (McDermott, 2002; Martínez und Lespérance, 2004) ist es möglich, Bedingungen für die Ausführung von Aktionen im Plan zu formulieren. Ausgehend von diesen Bedingungen können auch alternative Pfade im Plan vorhanden sein.
In Srivastava und Koehler (2004) argumentieren die Autoren, dass die Komposition von Webservices nicht als einmalige Plansynthese mit explizit vorgegebenen Zielen verstanden werden kann. Sie beschreiben Webservicekomposition

als kontinuierlichen Prozess der Manipulation komplexer Workflowbeschreibungen, was es erforderlich macht, Probleme der Prozessmodellgenerierung, der Ausführung, der Optimierung und der Wartung zu lösen, um so die Ziele der Komposition inkrementell zu verfeinern und sie schließlich lösbar zu machen. Um diesem Problem zu begegnen, ist eine Trennung von Synthese und Instanziierung nötig, nur so können die erwähnten Verfeinerungen durch Reinitialisierung oder erneuter Synthese erreicht werden. Verschränkte Verfahren, die Plansynthese und Ausführung miteinander verbinden, arbeiten intern in gleicher Weise (Agarwal u. a., 2008).
Eine Kompositionsmethode, die in Ad-hoc-Umgebungen arbeiten soll, wird in Fällen, bei denen das Prozessmodell a priori nicht bekannt ist, idealerweise auch neue Prozessmodelle generieren können. Es kommt daher für die Dienstekomposition in intelligenten Umgebungen lediglich eine dynamische Kompositionsmethode in Frage. Da nur auf das in den Komponenten enthaltene Wissen zurückgegriffen werden kann, muss sämtliche zur Komposition benötigte Semantik und Logik im Komponentenmodell hinterlegt sein. Wird KI-Planung zur dynamischen Generierung des Prozessmodells verwendet, ergibt sich daraus, dass als Orchestrierungsmodell Pläne verwendet werden können. Im Gegensatz zu Workflowbeschreibungen haben Pläne jedoch eine wesentlich schwächere Ausdrucksmächtigkeit. Allerdings lassen sich auch mit der überschaubaren Semantik von Plänen (Sequenz, Parallelität und ggf. Konditionen) viele Szenarien aus der Praxis abbilden (Marquardt und Uhrmacher, 2009b) (siehe Kapitel 4). Der entscheidende Vorteil von Plänen ist die Tatsache, dass sie mit KI-Planern automatisch generiert werden können. Workflows müssen manuell erzeugt werden und sind so für einen unaufdringlichen Einsatz in intelligenten Umgebungen nicht geeignet. Die Erkenntnis, dass die Verwendung von Workflows zur Dienstekomposition nur dann möglich ist, wenn das Prozessmodell bekannt ist, mit Planung jedoch neue Prozessmodelle gefunden werden können, kann mit folgender aphoristischer Sentenz zusammengefasst werden:
Workflows kennen den Weg und suchen die Mittel.
Planer kennen die Mittel und suchen den Weg.
In einer intelligenten Umgebung ist es im Gegensatz zum Internet sehr viel einfacher möglich, alle „Mittel“ (Dienste) zu kennen und den unbekannten „Weg“ (Prozessmodell) zu suchen und so eine sowohl dynamische als auch automatische Komposition zu ermöglichen.

These 2 *KI-Planer unterstützen eine dynamische und automatische Komposition von Diensten, da sie neue Dienste miteinbeziehen können und so in der Lage sind, neue Prozessmodelle zu erstellen.*

2.4 Kompositionsmethoden in der Anwendung

Im Folgenden werden verschieden Projekte bzw. Veröffentlichungen, die sich mit der Komposition von Diensten beschäftigen, untersucht. Es wird beschrieben, wann und auf welche Art und Weise das Prozessmodell generiert und instanziiert wird. Weiterhin wird das jeweils verwendete Komponenten- und Orchestrierungsmodell beschrieben. Zunächst wird die Komposition von Webservices betrachtet, da diese im überwiegenden Teil der Veröffentlichung zum Thema Dienstekompositon im Mittelpunkt steht.
Der für diese Arbeit relevante Unterschied zwischen Umgebungen bestehend aus Webservices und intelligenten Umgebungen ist die Anzahl der darin verfügbaren Dienste. Aus der räumlichen Abgrenzung intelligenter Umgebungen gegenüber Webservice-Umgebungen ergibt sich auch eine unterschiedliche Anzahl zu betrachtender Dienste. Sie ist in Webservice-Umgebungen sehr hoch. Hier können weder alle verfügbaren Dienste aufgelistet werden[3], noch kann der Zustand der Dienste umfassend überwacht werden. Im Gegensatz dazu ist die Anzahl der Dienste einer intelligenten Umgebung begrenzt und überschaubar. Eine Quantifizierung der Größe gegenwärtiger intelligenter Umgebungen wird in Abschnitt 5.1.1 gegeben. Durch die weitaus kleinere Anzahl an Diensten ist auch eine umfassende Überwachung ihrer Zustände (beispielsweise über broadcasts) möglich. Wird in intelligenten Umgebungen der Zugriff auf das Internet und somit den Zugriff auf Webservices zu, wird die gemachte Unterscheidung hinfällig, da dann auch in einer intelligenten Umgebung alle im Internet verfügbaren Webservices erreichbar sind und die Trennlinie zwischen beiden Umgebungen verwischt. Im Rahmen dieser Arbeit wird lediglich die Komposition von Diensten in abgeschlossenen intelligenten Umgebungen betrachtet.

2.4.1 In Webservice-Umgebungen

Das System **eFlow** (Casati und Shan, 2001) verwendet für die Beschreibung der Prozessmodelle Flussdiagramme, ähnlich den UML-Aktivitätsdiagrammen. Sie bestehen aus Dienst-, Entscheidungs- und Ereignisknoten. Die Kanten können zusätzlich mit Übergangsbedingungen versehen werden. Die Prozessmodelle werden zwar manuell erzeugt, dennoch ist eFlow dynamisch, da Änderungen des Prozessmodells zur Laufzeit vorgenommen werden können. Durch die Verwendung von generischen Dienstknoten ist das System auch hinsichtlich der Erstellung der Aktionssequenz dynamisch. Dienste und das Prozessmodell werden in eFlow in einem properitären XML-Format beschrieben. Darin sind

[3] Genauso wenig, wie es sinnvoll ist, alle verfügbaren Webseiten im Netz aufzulisten.

die Ein- und Ausgänge des Dienstes, die URI sowie Informationen über die Kosten und die Bezahlung des Dienstes enthalten.
Aggarwal u. a. (2004) nutzen die **METEOR-S** Plattform zur semantischen Annotation von Webservices (Patil u. a., 2004), um Dienste zu komponieren. Die Definition des Prozessmodells erfolgt a priori mit Hilfe von BPEL4WS-Prozessen. Als Komponentenmodell werden Service-Templates eingeführt, die ein spätes Binden der Dienste ermöglichen. Der Fokus der Arbeit liegt auf der Berücksichtigung von Qualitätsmerkmalen, um passende Dienste für ein gegebenes Prozessmodell zu finden. Die beschriebene Kompositionsmethode ist somit statisch und manuell bezüglich der Erzeugung des Prozessmodells. Die Instanziierung erfolgt aber dynamisch und automatisch.
In **SWORD** (Ponnekanti und Fox, 2002) werden einfache Hornklauseln verwendet, um Dienste zu beschreiben. Die Dienste werden dabei durch ihre Ein- und Ausgaben sowie durch ihre Abhängigkeiten von anderen Diensten (von den Autoren „bedingte Ein- und Ausgaben“ genannt) definiert. Das Prozessmodell wird dann durch logisches Schließen generiert. In einem anschließenden Schritt wird das generierte Prozessmodell als Graph visualisiert. Der Nutzer erhält so die Möglichkeit in die Generierung einzugreifen. Auf Anforderung des Nutzers wird schließlich eine Aktionssequenz erzeugt und ausgeführt. Die Kompositionsmethode ist dynamisch in Bezug auf die Generierung als auch die Instanziierung des Prozessmodells. Sowohl Prozessmodell als auch Aktionssequenzen werden automatisch generiert.
In McIlraith (2002) wird eine Methode zur Komposition von Webservices vorgestellt, die auf **Golog**, einer Programmiersprache basierend auf dem Situationskalkül (Levesque u. a., 1997), aufbaut. Dienste werden hier mit Hilfe der Beschreibungssprache DAML-Services (DAML-S) (Burstein u. a., 2002), einem Vorgänger von OWL-S (siehe Abschnitt 3.3), beschrieben, wobei neben den Ein- und Ausgängen auch Vorbedingungen und Effekte berücksichtigt werden. Weiterhin werden generische high-level-Prozeduren, ebenfalls in DAML-S, hinterlegt. Im Kompositionsprozess wird aus diesen Prozeduren das Prozessmodell generiert.
Sheshagiri u. a. (2003) verwenden einen **STRIPS**-basierten Planer zur Generierung von Prozessmodellen. Die verwendeten Dienste sind als DAML-S-Beschreibungen gegeben. Darin sind die Vorbedingungen und Effekte der Dienste beschrieben. Es wird ein eigener Planer verwendet. Das Prozessmodell dieser Kompositionsmethode ist daher ein einfacher Plan. Angaben über die Leistungsfähigkeit des verwendeten Planers werden nicht gemacht. Die Erzeugung des Prozessmodells ist dynamisch und automatisch. Die Instanziierung ist nicht Gegenstand der Arbeit.
Auch im **CASCOM** Projekt (Klusch und Zhing, 2008; Blankenburg u. a., 2008) werden klassische KI-Planer zur Komposition von Diensten verwen-

det. Anwendungsgebiet der CASCOM-Architektur ist die mobile Notfallassistenz. Die vorgestellte Architektur verwendet Dienstebeschreibungen in OWL-S. Auch für die Instanziierung des Prozessmodells werden die Möglichkeiten von OWL-S verwendet. Die Generierung wie auch die Instanziierung des Prozessmodells erfolgen dynamisch und automatisch. Das Projekt CASCOM ist die derzeit aktuellste und umfassendste Architektur zur Komposition von Webservices.

Rao und Su (2004) verwenden **lineare Logik** (LL) zur Komposition. Die Modellierung von Diensten mit Hilfe der linearen Logik hat den Vorteil, dass Ressourcen direkt ausgedrückt werden können, ein Aspekt, der bei der Modellierung von realen Systemen und damit auch intelligenten Umgebungen von Bedeutung ist (siehe Abschnitt 4.2.4). Die Generierung des Prozessmodells basiert auf der Idee der Softwarekonstruktion durch logische Beweise (Manna und Waldinger, 1992). Die Dienste liegen als DAML-S-Beschreibungen vor, die in LL-Axiome umgewandelt werden. Die Nutzeranfrage ist als LL-Term gegeben. Aus diesen Eingangsdaten kann ein Beweis erzeugt werden, sofern eine Komposition möglich ist. Dieser Beweis entspricht dann der gewünschten Komposition. Das erzeugte Prozessmodell wird mit Hilfe von DAML-S dargestellt. Synthese als auch Instanziierung des Prozessmodells erfolgen dynamisch und automatisch.

In Zheng und Yan (2008) wird eine Methode zur Komposition von Webservices vorgestellt, die auf einer adaptierten Variante des **GRAPHPLAN**-Algorithmus beruht. Dabei wird die Komposition basierend auf den Schnittstellenbeschreibungen in WSDL erstellt. Allerdings werden nur Ein- und Ausgänge sowie Fehlermeldungen zur funktionalen Beschreibung der Dienste verwendet. Die realen Vorbedingungen und Auswirkungen eines Dienstes auf seine Umgebung wurden nicht berücksichtigt. Durch die fehlende Betrachtung der Semantik der Dienste ist dieses Verfahren anfällig für semantisch ungeeignete Kompositionen. Die Korrektheit der automatisch generierten Kompositionen ist fraglich.

Zur Komposition von Webservices wurden auch Planer basierend auf Modelchecking eingesetzt (Traverso und Pistore, 2004; Yu und Reiff-Marganiec, 2006). Dieser Ansatz kann effektiv mit Nichtdeterminismus, unvollständiger Beobachtbarkeit und komplexen Zielen umgehen. Dabei wird ein logikbasiertes Komponenten- sowie Orchestrierungsmodell verwendet. Es ließen sich hierzu jedoch keine Aussagen über Laufzeitverhalten und ggf. zu beachtende Anforderungen und Bedingungen für die Modellierung entsprechender Probleme finden.

Bei der Untersuchung von Methoden zur Komposition von Webservices fällt auf, dass das Prozessmodell größtenteils nicht dynamisch erzeugt wird. Schwerpunkt der Forschung ist meist die dynamische Instanziierung des Prozessmo-

dells. Ursachen dafür sind insbesondere die hohen Anforderungen an Vorhersagbarkeit, Verlässlichkeit und Sicherheit von Diensten in den dominierenden Anwendungsgebieten Enterprise Resource Planning (ERP), Enterprise Application Integration (EAI) oder B2B-Integration (Srivastava und Koehler, 2003).

2.4.2 In intelligenten Umgebungen

In Chakraborty u. a. (2005) wird eine Architektur für die Dienstekomposition mit mobilen Geräten beschrieben (**SCfME** - Service Composition for Mobile Environments). Das Prozessmodell wird nicht generiert, sondern als Anfrage, in DAML-S formuliert, vorgegeben. Die Generierung der Aktionssequenz erfolgt zur Laufzeit. Chakraborty u. a. (2005) konzentrieren sich bei ihrer Arbeit auf die Definition von Protokollen zur Dienstekomposition, aber nicht auf den eigentlichen Kompositionsprozess selbst.
Im Rahmen des EU-Projektes IST **Amigo** enstand die Arbeit von Vallée u. a. (2005). Darin wird in „Plangeneration“, „Dienstekomposition“ und „Ausführungsmanagement“ unterteilt, dabei entspricht der erste Schritt der Synthese und der zweite Schritt der Instanziierung des Prozessmodells. Bei der Plangeneration wird aus einer Menge bereits bestehender abstrakter Prozessbeschreibungen die passende ausgesucht und dann mit den Diensten, die am besten geeignet sind, instanziiert. Das Kompositionsverfahren erstellt Prozessmodell und Aktionssequenz dynamisch. Dabei wird das Prozessmodell manuell, die Aktionssequenz jedoch automatisch erstellt.
Im Projekt **ICrafter** (Ponnekanti u. a., 2001) wird ein Framework für interaktive Arbeitsplätze (Johanson u. a., 2003), welche als Spezialfall von intelligenten Umgebungen angesehen werden können, beschrieben. Dabei kommt jedoch keine echte Dienstekomposition zum Einsatz, vielmehr wird eine Methode beschrieben, Benutzerschnittstellen (User Interface (UI)) automatisch zu generieren. UIs werden dabei nicht nur für einen bestimmten Dienst erzeugt. Durch Einführung von Interfaces im Sinne der objektorientierten Programmierung wird es möglich, mehrere Dienste in einer kombinierten UI verfügbar zu machen. ICrafter konzentriert sich so ausdrücklich auf die Mensch-Maschine-Interaktion, es werden komponierte Benutzerschnittstellen aber keine komponierten Dienste erzeugt.
In der **Obje**-Infrastruktur (Edwards u. a., 2005), die vormals unter dem Namen Speakeasy bekannt war, werden die Funktionen von Geräten auf vier generelle Punkte reduziert: 1. Geräte können sich mit anderen Geräten verbinden, 2. Geräte halten Metadaten über sich bereit, 3. können kontrolliert werden und 4. haben Referenzen zu anderen Geräten. Die Art und Weise, wie Geräte sich untereinander ansprechen können, wird von den Geräten selbst

in Form von Code-Bausteinen mitgebracht. Die Auswahl der Dienste erfolgt über die Metadaten. Hier erfolgt also keine wirkliche Dienstekomposition, da ausführbarer Code ausgetauscht wird und nicht wie bei Diensten die Implementierung verborgen bleibt. Dieses Verfahren des Austausches von Code ist kritisch zu betrachten, da zu bezweifeln ist, dass Hersteller den Code zur Verwendung ihrer Geräte vollständig veröffentlichen.
Eine weitere Architektur für die Umsetzung intelligenter Umgebungen ist **one.world** (Grimm, 2004; Grimm u. a., 2004). Sie soll Anwendungen ermöglichen, die sich automatisch an hoch dynamische Umgebungen anpassen können. Eine gemeinsame Ausführungsumgebung (in diesem Fall eine Java Virtual Machine) und die Möglichkeit der Verschachtelung von Anwendungen werden als einzige Voraussetzungen für deren Komposition angesehen. Das Komponentenmodell ist eventbasiert, da der Datenaustausch zwischen Anwendungen über Eventhandler realisiert wird. Allerdings reichen diese Handler die Events nur weiter. Wie die gesendeten Nachrichten verarbeitet werden, obliegt der jeweiligen Anwendung, das Problem der Komposition ist somit auf den Anwendungsentwickler verlagert. Eine automatische Komposition ist nicht möglich.
Das von Amigoni u. a. (2005) vorgestellte Verfahren **D-HTN** verwendet verteilte HTN-Planung zur Komposition von Diensten in intelligenten Umgebungen. Dabei sind die gewünschten Ziele der Komposition als Aufgabenbeschreibungen vorgegeben. Für den Planungsprozess wird zusätzliches Kontrollwissen in Form von Dekompositionsmethoden benötigt, die den Zieltask in atomare Aufgaben, repräsentiert durch Dienste, zerlegen können. Das Ergebnis des Planungsprozesses erhält ist eine Sequenz von Operatorbeschreibungen, die anschließend auf die vorhandenen Dienste der Umgebung gemappt werden können. Die Kompositionsmethode ist somit dynamisch in Bezug auf die Generierung des Prozessmodells und der Aktionssequenz. Die Komposition erfolgt weiterhin automatisch. Allerdings ist D-HTN auf erwähntes zusätzliches Kontrollwissen angewiesen.
Anstelle von Diensten verwenden Rouvoy u. a. (2008) Variationspunkte, um die verschiedenen möglichen Funktionalitäten von Geräten und Applikationen zu beschreiben. Den Variationspunkten werden mittels eines „property predictors“ Eigenschaften in Abhängigkeit zum gegenwärtigen Kontext zugeordnet. Diese Eigenschaften werden in angeforderte und angebotene Eigenschaften unterschieden, was einer Zuordnung von Vorbedingungen und Effekten entspricht. Komposition besteht bei Rouvoy u. a. (2008) aus der Kalkulation und dem Vergleich von QoS-basierten Eigenschaften von bereits existenten Plänen. Der Plan mit den besten Eigenschaften wird verwendet. Echte Komposition von Diensten wird nicht beschrieben, vielmehr werden Prozessmodelle anhand von QoS-Merkmalen klassifiziert.
Svensson u. a. (2007) verwenden im Rahmen des Projektes **PalCom** eine ei-

gene Sprache, um Prozessmodelle darzustellen. Diese Sprache wird sowohl in einer XML-basierten als auch in einer konkreteren, für den Menschen besser lesbaren Repräsentation angeboten. Um komplexeres Verhalten darstellen zu können, wird zusätzlich eine einfache Skriptsprache vorgestellt, mit der sich z.B. IF-THEN-ELSE-Anweisungen formulieren lassen. Diese Sprache soll es Nutzern erleichtern, das Zusammenspiel verschiedener Geräte zu steuern. Die vorgestellte Kompositionsmethode ist daher auf eine manuelle Erzeugung des Prozessmodells angewiesen. Im Papier wird eine Kompositionsmethode beschrieben, in der konkrete Dienste vom Nutzer direkt ins Prozessmodell eingebunden werden müssen. Daher ist die Instanziierung statisch und manuell.

2.5 Kompositionsmethoden für intelligente Umgebungen

Im Verlauf dieses Kapitels wurde ein allgemeines Kompositionsmodell entworfen. Zunächst wurden die vier Phasen einer Dienstekomposition beschrieben. Im Anschluss wurden zwei Dimensionen (temporal und modal) zur Klassifikation von Kompositionsmethoden herausgearbeitet, um eine Abgrenzung zu ermöglichen und einzelne Methoden so vergleichbar zu machen. Beide Dimensionen sind unabhängig voneinander. Die temporale Dimension kann in die Dynamik bei der Generierung und die Dynamik bei der Instanziierung des Prozessmodells unterschieden werden. Die modale Dimension lässt sich in manuell und automatisch unterteilen. Sowohl Generierung als auch Instanziierung des Prozessmodells können automatisch oder manuell erfolgen.
Für die Verwendung in intelligenten Umgebungen ist die Möglichkeit, neue Prozessmodelle dynamisch zu generieren, eine wesentliche Anforderung an ein Kompositionsverfahren. Im Sinne der geforderten Unaufdringlichkeit intelligenter Systeme sollte eine Kompositionsmethode zusätzlich ohne Interaktion des Nutzers auskommen. Es kommen demzufolge lediglich Methoden in Frage, deren Tempus und Modus bei der Erstellung und bei der Instanziierung des Prozessmodells dynamisch und automatisch ist. Weiterhin kann eine umfassende Gerätekooperation nur dann gelingen, wenn die Kompositionsmethode aktuelle und etablierte Dienstbeschreibungssprachen verwenden kann.

Projektname	Erstellung des Prozessmodells		Erstellung der Aktionssequenz	
	Tempus	Modus	Tempus	Modus
eFlow (Casati und Shan, 2001)	**dynamisch**	manuell	**dynamisch**	**automatisch**
METEOR-S (Aggarwal u. a., 2004)	statisch	manuell	**dynamisch**	**automatisch**
SWORD (Ponnekanti und Fox, 2002)	**dynamisch**	**dynamisch**	**dynamisch**	**automatisch**
Golog (McIlraith, 2002)	**dynamisch**	**automatisch**	**dynamisch**	**automatisch**
STRIPS (Sheshagiri u. a., 2003)	**dynamisch**	**automatisch**	—	—
CASCOM (Klusch und Zhing, 2008)	**dynamisch**	**automatisch**	**dynamisch**	**automatisch**
Linear Logic (Rao und Su, 2004)	**dynamisch**	**automatisch**	**dynamisch**	**automatisch**
Modelchecking (Traverso und Pistore, 2004)	**dynamisch**	**automatisch**	—	—
SCfME (Chakraborty u. a., 2005)	statisch	manuell	**dynamisch**	**automatisch**
AMIGO (Vallée u. a., 2005)	**dynamisch**	manuell	**dynamisch**	**automatisch**
PalCom (Svensson u. a., 2007)	statisch	manuell	statisch	manuell

Tabelle 2.1: Tempus und Modus von Kompositionsmethoden aus der Literatur

Projektname	Komponentenmodell	Orchestrierungsmodell
eFlow (Casati und Shan, 2001)	Properitäres XML-Format	intern
METEOR-S (Aggarwal u. a., 2004)	Service-Templates	BPEL4WS
SWORD (Ponnekanti und Fox, 2002)	Hornklauseln	intern
Golog (McIlraith, 2002)	Situationskalkül	Golog-Programme
STRIPS (Sheshagiri u. a., 2003)	DAML-S	Pläne
CASCOM (Klusch und Zhing, 2008)	OWL-S	OWL-S
Linear Logic (Rao und Su, 2004)	DAML-S	Pi-Kalkül
Modelchecking (Traverso und Pistore, 2004)	OWL-S	Pläne
SCfME (Chakraborty u. a., 2005)	DAML-S	DAML-S
AMIGO (Vallée u. a., 2005)	Properitäres XML-Format	Pläne
PalCom (Svensson u. a., 2007)	Properitäres XML-Format	Properitäres XML-Format

Tabelle 2.2: Komponentenmodell und Orchestrierungsmodell von Kompositionsmethoden aus der Literatur

In den Tabellen 2.1 und 2.2 werden Tempus und Modus sowie Komponentenmodell und Orchestrierungsmodell von ausgewählten Kompositionsmethoden zusammengefasst dargestellt. Um Verfahren hervorzuheben, die für die Anwendung in intelligenten Umgebungen geeignet sind, werden die Merkmale **dynamisch** und **automatisch** in der Tabelle 2.1 hervorgehoben dargestellt. Aus den Tabellen ist ersichtlich, dass die Methoden „SWORD“ (Ponnekanti und Fox, 2002), „CASCOM“ (Klusch und Zhing, 2008), „Linear Logic“ (Rao und Su, 2004) und „Golog“ (McIlraith, 2002) den Anforderungen an eine Komposition in intelligenten Umgebungen entsprechen. Letzteres ist sehr nahe an der verwendeten Sprache entworfen und erzeugt als Ausgabe Golog-Programme. Dieser monolithische Ansatz stellt eine nicht zu unterschätzende Hürde für die Umsetzung in realen Umgebungen dar, da nicht davon auszugehen ist, dass Golog-Interpreter in Ad-hoc-Umgebungen verfügbar sind. Ähnliches gilt für die Methode von Rao und Su (2004). Sie basiert auf der Erzeugung von Beweisen innerhalb der linearen Logik, was eine bestimmte Logikengine erforderlich macht. Die Methode „SWORD“ kommt aufgrund der bereits beschriebenen Nachteile, die durch die Verwendung von Hornklauseln zur Beschreibung der Dienste entstehen, nicht in Frage. Die Architektur von „CASCOM“ kann als sehr elaboriert angesehen werden, sie unterstützt alle wesentlichen Anforderungen an eine Kompositionsmethode.
Daneben kommen die Methoden „STRIPS“ (Sheshagiri u. a., 2003) und „Modelchecking“ (Traverso und Pistore, 2004) in Frage. In beiden Fällen wurde die Erzeugung von ausführbaren Aktionssequenzen jedoch nicht thematisiert. Als Ergebnis erzeugen beide Methoden einfache Pläne. Aufgrund ihrer überschaubaren Semantik ist eine Konvertierung in beispielsweise BPEL4WS-Prozesse möglich. Traverso und Pistore (2004) nennen diese Möglichkeit auch explizit. Beide Varianten basieren auf KI-Planung, sie unterscheiden sich aber in der verwendeten Planungstechnik. Daneben gibt es eine Reihe weiterer Veröffentlichungen, die die Verwendung von klassischer KI-Planung zur Komposition von Diensten thematisieren (McDermott, 2002; Blythe u. a., 2003).
Die Auswertung der Tabelle lässt den Schluss zu, dass KI-Planung eine geeignete Methode ist, um unaufdringliche Dienstekomposition in intelligenten Umgebungen zu ermöglichen. Die Frage welche Planungstechnik den Bedürfnissen intelligenter Umgebungen am besten entspricht, wird im nächsten Kapitel in Abschnitt 3.1 untersucht.

Kapitel 3

KI-Planung zur Dienstekomposition

3.1 KI-Planung

Im letzten Kapitel wurde der gesamte Kompositionsprozess beschrieben. Dabei wurde zwischen der Generierung und der Instanziierung des Prozessmodells unterschieden. In diesem Abschnitt wird die Verwendung von KI-Planung zur Generierung des Prozessmodells beschrieben.
Planung ist eine Methode der künstlichen Intelligenz, die ausgehend von einem Startzustand versucht, eine Sequenz von Aktionen, einen Plan, zu finden, die zu einem gegebenen Zielzustand führt. Sie kann als spezielles Suchproblem angesehen werden, bei dem versucht wird, den Suchraum intelligent zu beschränken. Anwendungsgebiete von Planern außerhalb der Forschung sind insbesondere die Logistik und die Robotik (Nau, 2007).

3.1.1 Klassische Planung

Klassische Planung basiert auf der Formalisierung der Welt als Zustandsübergangssystem (Ghallab u. a., 2004). Ein Zustandsübergangssystem definiert sich auf einer endlichen Menge an Zustandsvariablen $L = \{p_1, p_2, \ldots, p_n\}$, die Propositionen genannt werden und die Wahrheitswerte **true** oder **false** annehmen können. Ein Zustand ist ein n-Tupel von Wahrheitswerten, welches eine mögliche Belegung aller Propositionen repräsentiert. Der Zustandsraum $S = \{s_0, s_1, s_2, \ldots, s_m\}$ eines solchen Systems ist daher definiert durch $S \subseteq \{0,1\}^n$. Es gibt weiterhin eine Menge von Aktionen $A = \{a_1, a_2, \ldots, a_o\}$. Die Ausführung einer Aktion überführt einen Zustand in genau einen Folgezustand. Diese Überführung ist eine Funktion $\gamma(s, a) = s' \in S$, wenn gilt, dass a im Zustand s anwendbar ist. Die Anwendbarkeit einer Aktion a im Zustand s wird über deren Vorbedingungen $\mathit{precond}(a)$ definiert. Eine Vorbedingung besteht aus einer Konjunktion von Propositionen aus P. Sind alle Propositionen im Zustand s erfüllt, ist a in s anwendbar. Wird die Aktion ausgeführt, erfolgt ein Zustandsübergang von s zu s'. Die Änderungen der Zustandsvariablen sind dabei durch den Effekt von a determiniert. Der Effekt einer Aktion wird in positive ($\mathit{effects}^+(a) \in P$) und negative Effekte ($\mathit{effects}^-(a) \in P$) unterschieden, die wie $\mathit{precond}(a)$ Konjunktionen von Propositionen sind. Alle Propositionen aus $\mathit{effects}^+(a)$ werden in s' wahr, die Propositionen aus $\mathit{effects}^-(a)$ werden unwahr. Alle übrigen Propositionen behalten ihren alten Wahrheitswert.[1] Eine Aktion ist somit ein Tripel $a = (\mathit{precond}(a), \mathit{effects}^+(a), \mathit{effects}^-(a))$. Lässt man negierte Propositionen zu, vereinfacht sich die Beschreibung einer Aktion auf $a = (\mathit{precond}(a), \mathit{effect}(a))$. Damit kann ein Zustandsübergangssystem als ein Tripel $\Sigma = (S, A, \gamma)$ bestehend aus S, A und einer Zustandsübergangsfunktion $\gamma : S \times A \rightarrow S$ definiert werden.

[1] Vgl. Frameproblem (Ghallab u. a., 2004, S.162ff)

Ein Planungsproblem $\mathcal{P} = (\Sigma, s_0, G)$ besteht aus einem Zustandsübergangssystem Σ, einem Startzustand $s_0 \in S$ und einem möglichen Zielzustand $g \in G \subseteq S$. Ein klassischer Planer sucht eine mögliche Lösung des Planungsproplems, d.h. eine Sequenz von Aktionen $(a_1, a_2, \ldots, a_k \in A)$, die s_0 in einen Zielzustand g überführt, d.h. $s_1 = \gamma(s_0, a_1), s_2 = \gamma(s_1, a_2), \ldots, s_k = \gamma(s_{k-1}, a_k)$ und $g = s_k$. Für eine umfassende Formalisierung der klassischen Planung wird auf Ghallab u. a. (2004) und Russell und Norvig (2003) verwiesen.
Die Rückgabe eines Planers ist entweder ein Plan $P = (a_1, a_2, \ldots, a_k \in A)$, der eine Lösung für $\mathcal{P}$ darstellt, oder die Aussage, dass kein Plan gefunden wurde. Für den Fall, dass kein Plan gefunden wurde, geben einige Planer Gründe zurück, anhand derer nachvollziehbar ist, ob es nachweisbar keinen Plan für das vorliegende Problem gibt. Ist die formale Richtigkeit eines Planungsalgorithmus bewiesen, schließen sich zunächst falsch positive Planungsergebnisse (Fehler 1.Art) theoretisch aus. Ein gefundener Plan stellt dann immer eine Lösung für das zugrundeliegende Planungsproblem dar. Dennoch können die Planer Implementierungsfehler enthalten, die trotz korrektem Algorithmus falsche Pläne liefern. Es ist aus diesem Grund ratsam, gefundene Pläne vor ihrer Ausführung zu validieren.
Das Nichtfinden eines Planes kann nicht zwingend auf dessen Nichtexistenz zurückgeführt werden, da der Großteil der Planer den Suchraum nicht vollständig traversiert. Es besteht die Gefahr eines falsch-negativen Ergebnisses (Fehler 2. Art). Es liegt vor, wenn das Planungsproblem eine Lösung enthält, der Planer dieses jedoch aufgrund von Begrenzungen bei der Suche, z.B. in Bezug auf Laufzeit oder Speicherbedarf, nicht findet. Neben diesen algorithmischen Ursachen liegen oft auch formale Probleme bei der Problembeschreibung vor, die Planer nicht oder ungenügend durch Fehlermeldungen zurückgeben (siehe Abschnitt 5.2).
Für die Komposition von Diensten wäre eine genauere Begründung oder Erläuterung, warum ein bestimmtes Problem keine Lösung hat, wünschenswert. Mit einer möglichen Angabe von unerreichten Zielen ließen sich durch den Nutzer unter Umständen zielgerichtet Geräte und Dienste hinzufügen, die bisher fehlende Funktionalität bereitstellen. Diese Fehleranalyse ist jedoch mit klassischen Planungsansätzen nicht zu realisieren.

3.1.2 Einschränkungen klassischer Planung

Die Verwendung klassischer Planung ist an Einschränkungen gebunden. Die folgenden Annahmen zur Nutzung von KI-Planung werden von Ghallab u. a. (2004) sowie von Nau (2007) aufgezählt. Zu jeder Annahme wird erläutert, ob sie eine Verwendung in intelligenten Umgebungen zulässt oder welche Möglichkeiten der Anpassung es gibt.

Annahme 1 *(Endlicher Zustandsraum) Die Menge der Zustände des Planungsproblems (S) ist endlich.*

Systeme der realen Welt enthalten kontinuierliche Eigenschaften. Um sie in einem Planungsproblem formalisieren zu können, müssen sie zu diskreten Eigenschaften abstrahiert werden. Ausgehend von diskreten Zuständen und unter Berücksichtigung der Tatsache, dass der Wertebereich jeder realen Zustandsvariablen begrenzbar ist, folgt eine endliche Menge an Zuständen. Nach der hier gegebenen Definition können die Zustandsvariablen aus P nur die Wahrheitswerte **true** und **false** annehmen. Die Darstellung diskretisierter Werte bedarf üblicherweise der Verwendung von Zahlen samt dazugehöriger Arithmetiken. Sollen Zahlen (numerische Fluents) verwendet werden, müssen diese in einem Vorverarbeitungsschritt auf boolsche Variablen abgebildet werden. Verschiedene Planungserweiterungen (Koehler, 1998; Bedrax-Weiss u. a., 2003) leisten diese Umwandlung.

Annahme 2 *(Vollständig beobachtbar) Das Wissen über den Zustandsraum ist vollständig.*

Planung geht davon aus, dass alle relevanten Informationen zur Lösung des Problems zum Zeitpunkt der Planung verfügbar sind. D.h. allen Propositionen aus S muss genau ein Wahrheitswert zugeordnet sein. In realen Umgebungen kann eine vollständige Beobachtung der Umwelt nicht immer garantiert werden. Eine ständige komplette Erfassung aller Zustandsvariablen ist aufwändig und kann in realistischen Szenarien nicht gewährleistet werden. Aus diesem Grund wird das vorhandene Wissen als vollständig angenommen werden. Zu diesem Zweck gehen Planer von der sogenannten Closed-World-Assumption aus. Die Annahme einer geschlossenen Welt weist den Wahrheitswert jeder nicht explizit gesetzten Zustandsvariable aus L den Wert **false** zu.

Annahme 3 *(Deterministische Zustandsänderungen) Alle Zustandsübergänge des Systems haben nur ein und genau ein Ergebnis.*

Konkret bedeutet das, dass die Ausführung einer Aktion nur in genau einen Folgezustand führen kann. Verzweigungen (z.B. If-Then-Else-Anweisungen) oder Disjunktionen in den Effekten einer Aktion und damit in Plänen sind nicht möglich. Da sie nur auf den Zustandsvariablen (Propositionen) des Planungsproblems basieren, können derartige Verzweigungen über die Vorbedingungen der Aktionen ausgedrückt werden. So kann eine Aktion, deren Effekt eine Oder-Verknüpfung ist, in zwei Aktionen aufgeteilt werden, die jeweils einen Teil des Effektes beinhalten und die Bedingung, nach der das Oder aufgelöst wird, in ihrer Vorbedingung enthalten. Voraussetzung dafür ist das Wissen um diese Bedingung.

Annahme 4 *(Statischer Weltzustand) Die Zustände in S haben keine interne Dynamik.*

Im Sinne der KI-Planung bedeutet diese Einschränkung, dass keine Änderung der Zustandsvariablen erfolgen darf ohne Ausführung einer Aktion. In realen Systemen gibt es eine große Anzahl Zustandsvariablen, die nicht direkt kontrollier- und steuerbar sind (z.B. Temperatur, Helligkeit oder Folgen menschlichen Handelns). Um trotzdem in der Lage zu sein, KI-Planung zur Lösung realer Probleme einzusetzen, sind Methoden erforderlich, die die Änderungen der Umgebung, die nicht durch Dienstausführungen bedingt sind, beobachten und den Planer ggf. informieren. Diese Aufgabe übernimmt gewöhnlich die Komponente, die im Anschluss an die Planung auch für die Planausführung verantworlich ist (Ghallab u. a., 2004). Sie sammelt kontinuierlich die Informationen von verschiedensten Sensoren, die Änderungen der Umgebung automatisiert erfassen. Bei getrennter Planung und Ausführung ist u.U. Neuplanen oder eine Planreperatur erforderlich, um Pläne bei Änderungen von Zustandsvariablen aktuell zu halten.

Annahme 5 *(Eingeschränkte Ziele) Es sind keine zusätzlichen Anforderungen an Ziele, wie z.B. zu vermeidende Zustände oder Randbedingungen bei bestimmten Zustandspfaden erlaubt.*

Um dieser Anforderung zu genügen, müssen alle Bedingungen, die zur Problemlösung von Belang sind, über die Vorbedingungen und Effekte (Precondition and Effects (PEs)) der Aktionen beschrieben sein. Dazu gehören auch QoS-Merkmale oder andere Metriken. Eine mögliche Erweiterung, um mit zusätzlichen Anforderungen an Ziele zu planen, wird in Lago u. a. (2002) beschrieben.

Annahme 6 *(Sequenzielle Pläne) Das Ergebnis eines klassischen Planungsalgorithmus ist eine Sequenz von Aktionen.*

Jedes Planergebnis kann zu einer Sequenz von Aktionen (TOP) relaxiert werden. Es ist allerdings für viele Anwendungen vorteilhaft, Aktionen zu parallelisieren. Da reale Aktionsausführungen Zeit in Anspruch nehmen, kann ein paralleler Plan die Gesamtdauer einer Planausführung erheblich verkürzen. Es gibt Planer, die in der Lage sind, auch partiell geordnete Pläne (POP), mit denen eine parallele Ausführung von Aktionen ausgedrückt werden kann, zu generieren. Der Großteil der aktuellen Planer generiert jedoch lediglich sequentielle Pläne. So gab es bei der letzten International Planning Competition (IPC) (siehe Abschnitt 3.1.4.2) im Bereich der klassischen Planer ausschließlich Tracks für sequentielle Planer. Es existieren allerdings Algorithmen, die

sequentielle Pläne in Linearzeit in partiell geordnete Pläne überführen (Edelkamp, 2003) und so auch eine Verwendung von sequentiellen Planern ermöglichen, wenn parallele Aktionsausführungen erwünscht sind.

Annahme 7 *(Implizite Zeit) Die Ausführung einer Aktion geschieht augenblicklich und hat keine Zeitdauer.*

Es ist offensichtlich, dass diese Abstraktion in der Realität nicht zu erreichen ist. Jede Aktionsausführung ist zumindest mit einer Verzögerung (z.B. Reaktionsszeit, Netzwerklatenz) verbunden. Ist die Dauer einer Aktionsausführung von Belang, und das ist in fast allen Problemen der Fall, und kann sie im Vorfeld bestimmt werden (was beispielsweise für Latenzen nicht möglich ist), können Laufzeiten in die Vorbedingungen und Effekte des Dienstes eingefügt werden (Fox und Long, 2003). Dazu ist meist die Verwendung von kontinuierlichen Variablen nötig (siehe Annahme 1).

Annahme 8 *(Offline Planung) Während der Laufzeit des Planungsalgorithmus werden keine Änderungen der Umgebung beachtet.*

Wird in dynamischen Umgebungen geplant, besteht das Risiko, dass sich, während der Planungsprozess läuft, der Zustand der Welt ändert. Dadurch ist es möglich, dass dem Ergebnis eines Planungslaufes die Grundlage entzogen und es so hinfällig wird. Um die Korrektheit von Plänen zu gewährleisten, muss die Gültigkeit des Plans nach Abschluss der Planung sowie während dessen Ausführung evaluiert werden.

Weitere Einschränkungen Bei Srivastava und Koehler (2003, 2004) finden sich einige spezielle Einschränkungen bei der Anwendung von Planern zur Dienstekomposition. Dazu zählt die ungenügende Ausdrucksmächtigkeit von Vorbedingungen und Effekten zur Beschreibung der Funktionalität eines Dienstes. In der KI-Planung werden typischerweise einfache Datentypen zur Beschreibung der Operatoren verwendet. Dienste interagieren aber mit Austauschformaten, die aus sehr viel komplexeren Datentypen bestehen.

Als weiteren Punkt erläutern die Autoren, dass Aktionen während eines Planungslaufes keine neuen Objekte (z.B. Dokumente) erzeugen können, da sich KI-Planung auf einen abgeschlossenen Zustandsraum beschränkt, in dem alle Objekte zum Zeitpunkt der Initialisierung bekannt sein müssen. Diese Einschränkung wird im Abschnitt 4.2.3 genauer beschrieben.

3.1.3 Planungsalgorithmen

Klassische Planung Das erste System, dass klassische Planung umsetzte, war der Stanford Research Institute Problem Solver (STRIPS) (Fikes und Nilsson, 1971). Klassische Planung wird daher auch STRIPS-Planung genannt. Seit der Veröffentlichung von STRIPS wurden diverse andere Methoden entwickelt, klassische Planungsprobleme zu lösen (Rintanen und Hoffmann, 2001; Ghallab u. a., 2004). Es existieren Methoden, die auf Graphen basieren (Blum und Furst, 1995), die Planungsprobleme in Erfüllbarkeitsprobleme (SAT) umwandeln, um sie zu lösen (Kautz und Selman, 1996, 1998), die Suchalgorithmen mit speziellen Heuristiken verwenden (Bonet und Geffner, 2001) und Methoden, die auf Modelchecking (Giunchiglia und Traverso, 2000; Edelkamp, 2003) basieren. Dabei lösen die Algorithmen verschiedene Probleme unterschiedlich schnell (Howe u. a., 1999; Rintanen und Hoffmann, 2001; Roberts und Howe, 2009).

GRAPHPLAN- und SAT-basierte Planer finden immer kürzeste Pläne, da sie erst alle möglichen Lösungen der Länge n (beginnend mit 1 oder einem vom Benutzer gesetzten Wert) prüfen und zum nächstlängeren Plan $(n + 1)$ voranschreiten, wenn nachweisbar kein Plan der Länge n gefunden werden konnte. Daher skalieren diese Algorithmen für Probleme, die lange Pläne zur Lösung erfordern, schlecht. Werden zusätzliche Metriken verwendet, sind kürzeste Pläne zudem nicht notwendigerweise optimal.

Planer, die auf Heuristiken beruhen, verwenden Vorwärts- oder Rückwärtssuche, um Planungsprobleme zu lösen. Sie sind dabei auf die Güte ihrer Heuristik angewiesen. Dabei haben sich in den letzten Jahren vor allem die vorwärtssuchenden heuristischen Planer bewährt (Gerevini u. a., 2009).

Weiterhin können Modelchecker zur Lösung von Planungsproblemen eingesetzt werden (Giunchiglia und Traverso, 2000). Modelchecking wird für Anwendungsgebiete wie z.B. Verkehrsleitsysteme, Prozessorlayout aber auch in der Verifikation von Prozessmodellen (Foster u. a., 2003) verwendet. Das Verfahren erlaubt es zu überprüfen, ob eine Formel ϕ durch ein gegebenes Modell M verifiziert werden kann. Im Erfolgsfall bestätigt ein Modelchecker die Anwendbarkeit der Formel. Kann ϕ nicht verifiziert werden, gibt der Algorithmus ein Gegenbeispiel zurück. Wird Modelchecking zur Planung eingesetzt, wird das inverse Planungsziel als Formel vorgegeben. Das Gegenbeispiel, das der Modelchecker erzeugt, falls ϕ nicht verifiziert werden kann, entspricht dem gesuchten Plan. Planer basierend auf Modellchecking erlauben das Planen mit nicht deterministischen Operatoren. Auch sie haben in der Vergangenheit gute Leistungen gezeigt (Gerevini u. a., 2009).

HTN Planung Neben der klassischen Planung wird auch Hierarchical Task Network-Planung zur Dienstekomposition eingesetzt (Sirin u. a., 2004). HTN-Planung ist eine Erweiterung der klassischen Planung (Erol u. a., 1994). Zwar beruht auch HTN-Planung auf einem Zustandsübergangssystem Σ, allerdings ist das Ziel eines HTN-Planers nicht, einen bestimmten Ziel*zustand* zu erreichen. Stattdessen wird versucht, eine Ziel*aufgabe* zu lösen. HTN-Planer kennen zwei Arten von Aufgaben, zusammengesetzte und atomare. Atomare Aufgaben können als Aktionen im Sinne der klassischen Planung angesehen werden. Sie sind direkt ausführbar. Zusammengesetzte Aufgaben repräsentieren Aktionen, die noch zu dekomponieren und nicht direkt ausführbar sind. Die Dekomposition der Aufgaben wird in sogenannten Methoden beschrieben. Sie müssen dem Planer zusätzlich zu den Operatorbeschreibungen zu Beginn des Planungslaufes bekannt sein. Dekompositionsmethoden repräsentieren so abstrakte Planfragmente. HTN-Planung basiert auf der hierarchischen Dekomposition von Aufgaben. Um einen Plan zu generieren, beginnt ein HTN-Planer mit der Zielaufgabe (der Wurzel des Dekompositionsbaumes) und sucht rekursiv nach Methoden, die auf zusammengesetzte Aufgaben in den Blättern angewendet werden können, bis alle Blätter des Baumes aus atomaren Aufgaben bestehen (Russell und Norvig, 2003). Ein Beispiel für einen HTN-Planer ist das System SHOP2 (Nau u. a., 2003).
Im Vergleich zur klassischen KI-Planung haben HTN-Planer durch ihr zusätzliches Domänenwissen das Potential, Probleme wesentlich schneller zu lösen (Long und Fox, 2003). Des Weiteren macht die hierarchisch strukturierte Natur der Aufgaben das Planungsproblem für einen Benutzer leichter nachvollziehbar (Sirin u. a., 2004). Allerdings muss dieses Kontrollwissen manuell generiert werden. Der Aufwand dieser Erstellung lässt sich jedoch nicht quantifizieren (Long und Fox, 2003). Zwar gibt es Bestrebungen, HTN-Methoden automatisch zu generieren (Ilghami und Nau, 2002; Hogg u. a., 2008). Diese Verfahren basieren allerdings auf Lernverfahren, d.h. es sind mehrere Läufe im entsprechenden Szenario nötig, um eine automatische Generierung zu gewährleisten. Ein direkter Vergleich der Laufzeit eines HTN-Planers mit einem klassischen Planer ist daher nicht möglich.
Ein Hindernis bei der Verwendung von HTN-Planung ist die Art der Formalisierung der Dekompositionsmethoden. Zur Beschreibung dieser Methoden in PDDL (siehe nächster Abschnitt) hat sich bis heute keine gemeinsame Notation durchsetzen können. Zwar existieren Erweiterungen für PDDL (Srivastava, 2003; Armano u. a., 2003), jedoch ist in den aktuellen PDDL-Spezifikationen keine Unterstützung für HTN-Methoden vorgesehen. Projekte, die PDDL in HTN-Planern verwenden, nutzen Elemente der verschiedenen Sprachen aus dem Bereich der Semantic Web Community (z.B. DAML-S (Wu u. a., 2003) oder OWL-S (Sirin u. a., 2004)), um diese Lücke zu füllen.

3.1.4 PDDL

Nachdem eine Einführung in die KI-Planung gegeben wurde, wird in diesem Abschnitt die Planning Domain Description Language als Beschreibungssprache für Planungsprobleme vorgestellt. Dazu wird zunächst erläutert, wie ein Planungsproblem mit PDDL beschrieben ist. Im Anschluss wird die Verwendung der Sprache an einem Beispiel aus der Literatur erläutert. Abschließend wird kurz auf die IPC als treibende Kraft bei der Weiterentwicklung von PDDL eingegangen.

Im Abschnitt 3.1.1 wurde definiert, dass ein Planungsproblem $\mathcal{P}$ aus einem Zustandsübergangssystem Σ, einer initialen Belegung der Zustandsvariablen $s_0 \in S$ sowie einer Menge von Zielzuständen $G \subset S$, besteht. PDDL verwendet demgegenüber eine leicht abgewandelte Notation. PDDL unterscheidet zwischen einer Domäne und einem Problem und folgt damit dem Konzept der Unterteilung von Beschreibungslogiken in TBox und ABox (Baader u. a., 2003). Im Gegensatz zur obigen Definition werden in PDDL-Domänen Variablen für die Beschreibung von Σ verwendet. Im Gegensatz zu STRIPS, was auf reiner Aussagenlogik beruht, nutzt PDDL Elemente der Prädikatenlogik.

Domänen bestehen in PDDL nicht aus Propositionen $p \in P$ und Aktionen $a \in A$, die variablenfrei sind, sondern aus Prädikaten und Operatoren. Während einer *Proposition* immer ein Wahrheitswert zugeordnet ist, kann ein *Prädikat* Variablen enthalten. Um Variablen zu instanziieren, werden in der PDDL-Problembeschreibung Objekte definiert. Erst nachdem eine Prädikat instanziiert wurde, kann es ausgewertet werden. Während reines STRIPS nur eine Liste von Propopsitionen als Vorbedingung enthält, erlaubt PDDL die Verwendung von funktionsfreien Sätzen der Prädikatenlogik erster Ordnung, die sich aus den Prädikaten der Domäne zusammensetzen. Ein Effekt kann dahingegen lediglich eine Konjunktion von (negierten) Prädikaten sein, ein universell quantifizierter Effekt oder ein bedingter Effekt sein. Nichtdeterministische Disjunktionen sind nicht erlaubt (Blankenburg u. a., 2008).

Probleme bestehen aus den Objekten, dem initialen Weltzustand s_0 und einer Beschreibung von G sowie ein Verweis auf die zugrundelegende PDDL-Domäne.[2] Zusätzlich enthalten PDDL-Domänen und -Probleme einige Metadaten.

[2] Um einen vollständigen Zielzustandustand $g \in S$ zu beschreiben, müssten die Belegungen für alle Propositionen $p \in P$ gegeben sein. Zielbeschreibungen sind in PDDL wie auch die Vorbedingungen von Operatoren funktionsfreie Sätze der Prädikatenlogik erster Ordnung. Üblicherweise werden so nur die Teilmenge der verfügbaren Propositionen beschrieben, die im Zielzustand erfüllt sein muss. Es sind also viele Zustände g möglich, um ein gegebenes Ziel zu erreichen. Damit entspricht die Zielbeschreibung immer einer Menge von akzeptierten Zuständen ($G \subseteq S$).

3.1.4.1 Beispiel

In diesem Abschnitt wird die Verwendung von PDDL zur Beschreibung von Planungsproblemen anhand eines Beispiels aus der Literatur erläutert. Die dabei verwendeten Sprachelemente sind bereits in der Version PDDL1.2 enthalten. Es wird ein Szenario von Heider und Kirste (2005) verwendet. Wenngleich das Verhalten des originalen Szenarios spezialisierter und ausdifferenzierter war und im ursprünglichen Projekt EMBASSI elaboriertere Szenarien entwickelt wurden, wird hier eine vereinfachte Variante benutzt, um die Verwendung von PDDL besser darstellen zu können. Zunächst wird eine natürlichsprachliche Beschreibung des Szenarios gegeben.
Ein namenloser Nutzer schaut fern und möchte die Helligkeit des Fernsehbildes erhöhen. Da das Gerät bereits auf maximale Helligkeit eingestellt ist, müsste die umgebende Lichtstärke reduziert werden, um das Ziel zu erreichen. Das kann durch ein Schließen der Rollos und das Dimmen der Lampen im Raum geschehen, da dadurch die wahrgenommene Helligkeit des Fernsehers steigt.
PDDL basiert auf Lisp. Es beruht daher auf Listen und nutzt die Präfixnotation. Variablen werden mit einem vorangestellten Fragezeichen gekennzeichnet, z.B. ?x.
Am Anfang dieses Abschnitts wurde erläutert, dass PDDL zwischen Planungsdomäne und Planungsproblem unterscheidet. Es wird zunächst der Aufbau einer PDDL-Domäne beschrieben. Sie beginnt mit einem Kopf, der den Namen der Domäne und eine Liste von Anforderungen an den zu verwendenden Planungsalgorithmus enthält. Die Anforderung : strips bedeutet, dass der Planer einfache STRIPS-Planung unterstützen muss und ist daher obligatorisch für die meisten Szenarios.[3] Im Anschluss werden Konstanten definiert. Es sind in diesem Fall die Messwerte MAX, HIGH, and LOW.

```
(define (domain brigthenTV)
  (:requirements :strips)
  (:constants MAX HIGH LOW)
```

Da Planung zustandsbasiert ist, müssen Prädikate, die den Zustand der Welt beschreiben, definiert werden. Diese Prädikate repräsentieren Aspekte der Welt, die durch die Operatoren ausgelesen oder verändert werden können. Eine Alternative, die oben genannten Konstanten auszudrücken, wäre die Einführung von verschiedenen Prädikaten, wie zum Beispiel (maxBrightness ?tv), (lowLuminosity ?lamp) oder (highLuminosity ?lamp). Die folgenden Prädikate werden im Beispiel verwendet:

```
(:predicates
  (brightness ?tv ?value) ;Die Helligkeit des Fernsehers
```

[3]Eine vollständige Auflistung der in PDDL möglichen Anforderungen findet sich in der Erweiterte-Backus-Naur-Form (eBNF) von PDDL (Edelkamp und Hoffmann, 2004).

```
(luminosity ?lamp ?value) ;Die Leuchtkraft einer Lampe
(dimmable ?lamp) ;Ist eine Lampe dimmbar
(isOpen ?shutter) ;Ist ein Rollo gerade offen
```

Nach der Definition des Kopfes, enthält eine PDDL-Domäne die Beschreibung der Operatoren.[4] Jeder Operator besteht aus einem Namen, einer Parameterliste der verwendeten Prädikate, seinen Vorbedingungen und seinen Effekten. Vorbedingungen und Effekte beschreiben die Semantik eines Operators.

```
(:action dim-down ;Ein Operator zum Dimmen einer Lampe
  :parameters (?x)
  :precondition (and (luminosity ?x HIGH)(dimmable ?x))
  :effect (luminosity ?x LOW))
(:action closeShutter ;Dieser Operator schließt ein Rollo
  :parameters (?x)
  :precondition (and (isOpen(?x)))
  :effect (not (isOpen(?x))))
```

Die Definition der Prädikate und der Operatoren sind die Hauptbestandteile einer Domänenbeschreibung. Im Anschluss wird der Aufbau eines PDDL-Problems beschrieben. Im Kopf einer Problemdefinition sind der Name des Problems sowie der Name der zugehörigen Domäne angegeben. Diese Notation lässt erkennen, dass Probleme immer fest an eine Domänenbeschreibung gebunden sind. Domänen dagegen können unabhängig von Problemen definiert werden.

```
(define (problem brightenTV-problem)
  (:domain brightenTV)
```

Zunächst werden in einem Problem Objekte definiert. Sie repräsentieren die Objekte der realen Welt, die durch den Plan gesteuert werden sollen. Die Variablen in den Operatorbeschreibungen der Domäne werden während der Planung mit den Objekten des Problems instanziiert, wodurch ausführbare Aktionen entstehen.

```
(:objects TV SHUTTER LAMP)
```

Nach den Objekten wird der initiale Zustand der Welt s_0 definert. Er besteht aus den Zuweisungen der Prädikate der Domäne. Dabei müssen nicht alle Prädikate der Domäne zugewiesen werden. Aufgrund der "closed-world-assumption" (siehe Annahme 2) werden alle nicht definierten Zustandsvariablen vom Planer automatisch als ungültig angenommen.

```
(:init
  (brightness TV MAX)
  (luminosity LAMP HIGH)
  (dimmable LAMP)
  (isOpen SHUTTER))
```

Da KI-Planung versucht, einen Plan von Aktionen zu erzeugen, der ein bestimmtes Ziel erfüllt, muss schließlich dieses Ziel formuliert werden. Ähnlich

[4] Operatoren werden in PDDL mit :action bezeichnet. Zum Unterschied zwischen Operator und Aktion siehe oben.

dem initialen Weltzustand besteht auch das Ziel aus einer Menge von Zuweisungen der Zustandvariablen. Auch hier gilt, es müssen nicht alle Zustandsvariablen der Domäne erwähnt werden.

```
(:goal (and (not (isOpen SHUTTER))(brightness LAMP LOW)))
```

Mit einer Domäne und einem Problem ist ein Planungsalgorithmus in der Lage, einen Plan zu suchen, der das Problem in der dazugehörigen Domäne löst. Ein möglicher Plan, der das gegebene Szenario löst, ist unten dargestellt. Der verwendete Planer erzeugt POPs, die parallele Aktionen enthalten können. Die Zahl vor den Aktionen zeigt den Zeitpunkt, an dem die Aktion ausgeführt werden soll. Aktionen mit gleichem Index beeinflussen sich gegenseitig nicht und können daher parallel ausgeführt werden.

```
0 (DIM-DOWN LAMP)
1 (CLOSESHUTTER SHUTTER)
```

3.1.4.2 IPC

Die International Planning Competition (IPC) ist ein Wettbewerb für Planer und findet zweijährlich im Rahmen der International Conference on Automated Planning and Scheduling (ICAPS) statt. Seit der ersten IPC im Jahr 1998 (McDermott, 2000) wurde sie bis jetzt sechs mal in den Jahren 2000 (Bacchus, 2001), 2002 (Long und Fox, 2003), 2004 (Hoffmann und Edelkamp, 2005), 2006 (Gerevini u. a., 2009) und 2008 veranstaltet[5]. Am Wettbewerb können sich Wissenschaftler und Programmierer von Planungsalgorithmen beteiligen. Im Laufe der vergangenen IPCs haben sich sowohl die Planer als auch die Planungsprobleme immer weiter diversifiziert. Daher ist der Wettbewerb in verschiedene Kategorien, sogenannte Tracks, unterteilt. Jeder Track beinhaltet verschiedene Arten von Problemen und hat verschiedene Anforderungen an die teilnehmenden Planer. So werden beispielsweise in den „satisficing tracks“ lediglich korrekte Pläne, in den „optimization tracks“ dahingegen optimale Lösungen gesucht. Die Autoren der Planer können entscheiden, in welchem Track ihr Planer starten soll. Jedem teilnehmenden Planer wird die gleiche Menge an vorgegebenen Problemen zur Lösung gegeben.
Bis zur IPC5 wurden die Laufzeit sowie teilweise die Qualität der erreichten Pläne zur Bewertung der teilnehmenden Planer benutzt. Seit der letzten IPC wird sich lediglich auf die Planqualität konzentriert und ein festes Zeitfenster vorgegeben, in dem die Planer terminieren müssen. Die Ergebnisse des Wettbewerbs werden analysiert und veröffentlicht. Die jeweils besten Planer der verschiedenen Kategorien werden gekürt.

[5] http://ipc.informatik.uni-freiburg.de/

Um einen fairen Vergleich der Leistungsfähigkeit von Planern zu ermöglichen, wurde für die erste IPC eine gemeinsame Modellierungssprache für Planungsprobleme, die PDDL, entwickelt. Die Sprache basiert sehr stark auf der Syntax der Problemrepräsentation, die der LISP-basierte Planer UCPOP (Penberthy und Weld, 1992) verwendet.
Die Vereinheitlichung der Domänen- und Problembeschreibung für Planungsprobleme in PDDL war ein wesentlicher Schritt, um eine breite Anwendung von Planern außerhalb der Forschung zu ermöglichen.
Die IPC hat sich als zentraler Austausch- und Evaluierungspunkt für aktuelle Entwicklungen auf dem Gebiet der Planer aber auch der Problemmodellierung etabliert und ist damit die treibende Innovationskraft für die (Weiter)Entwicklung von PDDL. Neue Versionen der PDDL sind regelmäßig Gegenstand von wissenschaftlichen Diskursen (Bacchus, 2003; Thiébaux u. a., 2005). Bis jetzt wurde zu jeder IPC eine neue Erweiterung von PDDL vorgestellt, die aktuelle Version auf der IPC6. Am Ende dieses Abschnitts findet sich eine kurze Versionshistorie von PDDL. Da es jedoch keine Standardisierungsinstanz oder einen formalen Standardisierungsprozess für PDDL gibt, ist die Sprache kontinuierlichen und ungesteuerten Veränderungen unterworfen. Dennoch hat sich PDDL in der Planungscommunity als gemeinsamer Standard zur Beschreibung von Planungsdomänen und -problemen durchgesetzt und wird von den meisten aktuellen Implementierungen unterstützt. PDDL stellt damit einen de-facto Standard (Eisenberg und Melton, 1998) auf diesem Gebiet dar.
Neben den Ergebnissen über die Leistungsfähigkeit und Qualität aktueller Planer sind die Berichte der IPC eine gute, weil zentrale, Quelle für die Fähigkeiten der Planer. In den Veröffentlichungen zu den letzten IPCs waren Tabellen enthalten, die die unterstützten Fähigkeiten für alle teilnehmenden Planer darstellen. Diese Tabellen sind für den praktischen Einsatz der Planer hilfreich, da die Dokumentierung der Planer in Hinblick auf deren Fähigkeiten und konkrete Unterstützung von PDDL sehr dürftig ist. Es ist zu beobachten, dass kaum ein Planer den gesamten Umfang von PDDL vollständig umsetzt. Lediglich die gebräuchlichsten Elemente der Sprache werden vom Gros der Planer unterstützt. Um möglichst viele Planer verwenden zu können, bietet sich daher eine Problemmodellierung an, die sich auf diese verbreiteten Eigenschaften beschränkt.

3.1.4.3 Versionshistorie

Der Abschnitt 3.1.4.1 stellte die wesentlichen Sprachelemente von PDDL1.2 vor. Diese Version wurde bereits im Jahr 1998 vorgestellt (Ghallab u. a., 1998). Im Laufe der darauffolgenden IPCs wurde die Sprache sukzessive erweitert. In

Vorbereitung der dritten IPC wurde PDDL2.1 auf Grundlage einer Teilmenge der originalen Beschreibung von 1998 neu definiert. Die Sprache wurde um drei Punkte ergänzt:

- Umgang mit einer endlichen Menge von numerischen Variablen
- Beschreibung von Zeit und Dauer (bei der Aktionsausführung)
- Einführungen von Planmetriken als Teil der Problembeschreibung

Die Verwendung numerischer Variablen erleichtert die Modellierung von Planungsdomänen (siehe Annahme 1). Ohne eine solche Erweiterung kann Planung nur mit boolschen Variablen arbeiten. Durch eine Einführung von Zeitspannen wird eine sehr viel realitätsnähere Modellierung der Aktionen ermöglicht (siehe Annahme 7). Planmetriken ermöglichen es, Qualität von Plänen über die Länge der Pläne hinaus zu definieren. Damit ist es möglich Qualitätsanforderungen aus den Vorbedingungen zu separieren und gesondert zu definieren (siehe Annahme 5).
Neben den aus PDDL2.1 bekannten Elementen, wurden der PDDL2.2 für die vierte IPC abgeleitete Prädikate, mit denen Domänenaxiome ausgedrückt werden können, und zeitgesteuerte initiale Literale, mit denen sich externe Ereignisse abbilden lassen, hinzugefügt. Domänenaxiome werden im Abschnitt 4.2.6 genauer beschrieben.
Die Erweiterungen von PDDL3 und der zur Zeit aktuellen Version PDDL3.1 sind für diese Arbeit unerheblich und werden nicht genauer betrachtet.[6]

These 3 *PDDL ist gegenwärtig die einzige Beschreibungssprache für Planungsprobleme, die von den meisten aktuellern Planungsalgorithmen akzeptiert wird und damit als intermediäre Austauschsprache unverzichtbar, sobald KI-Planung zur Problemlösung verwendet werden soll.*

[6]Eine vollständige Übersicht über die Entwicklung von PDDL mit den entsprechenden Referenzen findet sich unter http://ipc.informatik.uni-freiburg.de/PddlResources.

3.2 Mapping zwischen Dienstekomposition und KI-Planung

Im Abschnitt 3.1.1 wurde KI-Planung als eine Möglichkeit der Generierung eines Prozessmodells im Rahmen der Dienstekomposition beschrieben. Dabei besteht ein Planungsproblem $\mathcal{P}$ aus einem Zustandsübergangssystem Σ, einem Startzustand s_0 und der Beschreibung der Menge möglicher Zielzustände $G \subseteq S$, wobei der wesentliche Bestandteil von Σ die Operatoren O sind (vgl. Abschnitt 7.2). In diesem Abschnitt wird dargestellt, wie das Problem der Dienstekomposition in einer intelligenten Umgebung auf ein solches Planungsproblem abgebildet werden kann.
Intelligente Umgebungen sind mit einer Vielzahl von Sensoren ausgestattet. Dazu gehören herkömmliche Sensoren zur Messung physikalischer Standardgrößen, wie Temperatur, Helligkeit und Lautstärke aber auch komplexere Sensorsysteme zur Lokalisierung, mit denen Standorte von Personen im Raum ermittelt werdem können (Stelios u. a., 2008). Durch neue leistungsfähige Verfahren in der Bilderkennung ist es weiterhin möglich, Gesichter und sogar die Emotionen der jeweiligen Person in Echtzeit zu erkennen (Bartlett u. a., 2006; Panning u. a., 2008). Sie bilden die Grundlage für Intentionserkennungssysteme. Neben der Integration von Sensoren in die Umgebung verspricht die Vision des pervasiven Computings die ständige Verfügbarkeit von Informationen aller Art. So können Ablaufpläne von Sitzungen als auch Informationen über die teilnehmenden Personen und unter Umständen sogar die relevanten Daten der Teilnehmer verfügbar sein.[7]
Um aus diesen heterogenen Eingangsdaten ein Planungsproblem zu generieren, wird der Zustand der Umgebung auf s_o, die Dienste auf O sowie die Intentionen des Nutzers auf G abgebildet. Die Ergebnisse der Planung, die Pläne, werden schließlich in Aktionssequenzen überführt. Diese können als neue, zusammengesetzte Dienste betrachtet werden (siehe Abbildung 3.1).
Die generelle Herausforderung des Mappings auf ein klassisches KI-Planungsproblem ist die Abstraktion der Realität intelligenter Umgebungen, als auch der Kommunikation der enthaltenen Geräte auf ein logikbasiertes Zustandsübergangssystem. Dabei funktionieren dienstbasierte Systeme auf dem Austausch von Nachrichten, wohingegen Planung die Welt über Objekte und deren Eigenschaften modelliert.
Damit ein Planungsproblem erzeugt werden kann, müssen die Beschreibungen der Situation, der Intentionen und der Dienste die gleiche Syntax und Seman-

[7] Insbesondere beim Zugriff auf persönliche Daten stellen sich dringende Fragen nach Privatsphäre und Vertraulichkeit der Daten in intelligenten Umgebungen. Dieser Punkt kann im Rahmen dieser Arbeit nicht behandelt werden. Für weiterführende Informationen und Ansätze zu diesem Thema wird stellvertretend auf Bünnig und Cap (2007) verwiesen.

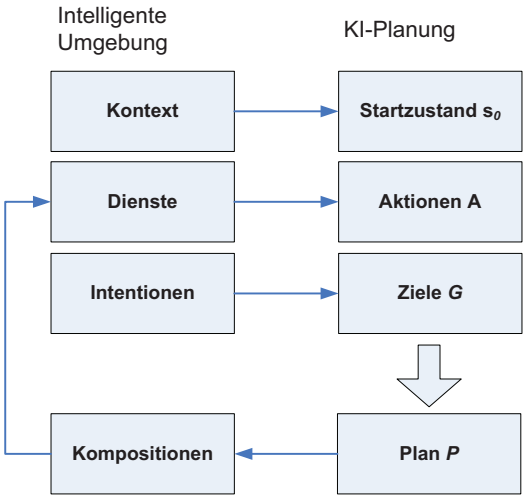

Abbildung 3.1: KI-Planung zur Dienstekomposition in intelligenten Umgebungen

tik, also ein gemeinsames Vokabular, basierend auf einer gemeinsamen Ontologie haben. Da PDDL als Beschreibungssprache verwendet wird, ist die Syntax klar. Allerdings gibt es für PDDL keine Ontologiesprache, die jedoch für eine gemeinsame Semantik unabdingbar wäre.[8] Es wird daher vereinfachend angenommen, dass syntaktisch gleiche Elemente auch die selbe Semantik haben. Elemente gleicher Semantik, die unterschiedlich bezeichnet sind, können durch diese Vereinfachung jedoch nicht als semantisch identisch erkannt werden. In Abschnitt 4.2.7 wird näher auf die Problematik einer gemeinsamen Ontologie eingegangen. Abschnitt 4.2 beschreibt zusätzlich Modellierungsrichtlinen, die für eine reibungslose Interaktion heterogener Geräte unumgänglich sind.

3.2.1 Situation zu initialem Weltzustand

Die gegenwärtige Situation einer intelligenten Umgebung wird im Planungsproblem durch den initialen Weltzustand s_0 repräsentiert. Eine genaue Eingrenzung des Begriffes Situation ist dabei problematisch. Für eine erfolgreiche Dienstekomposition muss die Situation Informationen über alle Prädikate enthalten, die in den Operatoren verwendet werden. Im Rahmen dieser Arbeit sollen unter der gegenwärtigen Situation einer Umgebung daher alle relevanten Informationen verstanden werden, die für eine Kompositionsinstanz potentiell verfügbar sind und keine Dienstbeschreibung oder Nutzerintention darstellen.

[8] Als Beschreibungssprache ist PDDL selbst in gewisser Weise eine Sprache, mit der eine Ontologie formuliert werden kann. Darin würden die Objekte der Welt über ihre Wechselwirkung mit der Umgebung beschrieben werden. Die hier geforderte Semantik meint eher die gemeinsame Verwendung gleicher Terme.

Dazu zählen die beschriebenen Sensorinformationen, Informationen aus verschiedenen Datenbanken und weiteren Datenquellen sowie Informationen über den aktuellen Zustand der Geräte.

3.2.2 Services zu Operatoren

Um das Zusammenwirken von Diensten planen zu können und so eine Komposition der Dienste zu erreichen, ist es notwendig, die Dienstbeschreibungen auf Planungsoperatoren abzubilden. Dabei kann die Beschreibung des Dienstes kein szenariospezifisches Wissen enthalten, da die Umgebung, in der der Dienst verwendet wird, zum Zeitpunkt der Formulierung der Dienstbeschreibung nicht bekannt ist. Trotzdem soll die Beschreibung allgemein genug sein, um eine breite Anwendbarkeit des Dienstes zu ermöglichen.
Es stellt sich die Frage, ob es eine direkte Möglichkeit der Abbildung einer Dienstbeschreibung auf einen Planungsoperator gibt, oder ob es nötig ist, zusätzliche Informationen zu verwenden, um ein Mapping zu ermöglichen.[9]
Eine übliche Dienstschnittstelle verfügt über eine Liste von Methodenbeschreibungen mitsamt einer Beschreibung der Parameter und der Rückgabetypen der einzelnen Methoden. Ein Planungsoperator verfügt über eine Beschreibung der Vorbedingungen und der Effekte seiner Ausführung (vgl. Abschnitt 3.1). Eine Schnittstellenbeschreibung und ein Planungsoperator stellen so unterschiedliche Sichten (nachrichtenbasierte Dienstbeschreibung und zustandsbasierte Operatorbeschreibung) auf dieselbe Entität, den Dienst, dar. Da der Planungsoperator eine Repräsentation des realen Dienstes ist, müssen alle Informationen, die im Operator verwendet werden, auch in der Dienstbeschreibung vorhanden sein. Es stellt sich die Frage, ob die Ein- und Ausgänge des Dienstes (also dessen Parameter und Rückgabewerte) als Vorbedingungen und Effekte (PEs) des Operators aufgefasst werden können.
Blankenburg u. a. (2008) schlagen ein Prädikat agentHasKnowledgeAbout für alle Informationen, die ein Dienst über seine Eingänge benötigt oder seine Ausgänge produziert, vor. Damit schaffen sie eine Abbildung von den Ein- und Ausgängen eines Dienstes auf dessen Vorbedingungen und Effekte. Konzeptionell werden damit jedoch Beschreibungen der zwei erwähnten Sichten auf den Dienst miteinander in Verbindung gebracht, was problematisch ist. Parameter und Rückgaben eines Dienstes spezifizieren lediglich den Zugriff auf einen Dienst, treffen aber keine Aussage über dessen Funktionalität, die

[9] Dabei muss zwischen der einfachen Beschreibung des Dienstes und der zustandsbehafteten Instanz des Dienstes auf einem Gerät unterschieden werden. Für den weiteren Verlauf des Kapitels ist dabei immer die Dienstbeschreibung von Interesse und nicht die Instanz des Dienstes. Wenn von einem Dienst gesprochen wird, ist daher immer die Beschreibung des Dienstes gemeint.

durch die PEs beschrieben sind. Zwar ist es denkbar, über die Parameter und Rückgaben eines Dienstes auf dessen Funktionalität zu schließen. Das Ergebnis eines solchen Schlusses wäre allerdings nicht eindeutig.
Für eine eindeutige semantische Beschreibung eines Dienstes ist es nötig, seine Vorbedingungen und Effekte zusätzlich zu seinen Schnittstellen zu hinterlegen. Diese Vorbedingungen und Effekte können dann zur Erzeugung eines entsprechenden Operators verwendet werden.
Werden die Ein- und Ausgänge bei der Planung nicht berücksichtigt, ist die Erzeugung von Plänen möglich, in denen Dienste interagieren müssen, deren Schnittstellen nicht zueinander passen. Ein Matchmaker muss dann das Anpassen der Nachrichten übernehmen. Im Rahmen der WSMO Initiative (Lausen u. a., 2005) werden solche Komponenten Mediatoren genannt. Falls kein Mediator gefunden werden kann, stellen solche Pläne im schlechtesten Fall keine Lösungen für ein Kompositionsproblem dar.
Während bei Webservices Dienste hauptsächlich durch ihre Ein- und Ausgaben charakterisiert sind, gibt es in intelligenten Umgebungen zudem eine große Anzahl von Diensten, die über PEs verfügen, jedoch weder Ein- noch Ausgaben haben. Beispiele dafür sind Dienste, die ein Gerät ein- oder ausschalten.
Es muss weiterhin verdeutlicht werden, dass in der Literatur bezüglich der Input Output Precondition Effects (IOPEs) verschiedene Arten von Diensten unterschieden werden. Diese Unterteilung stammt aus dem Bereich der Webservices und ist in der Literatur nicht einheitlich. So unterscheiden McIlraith (2002) und Rao und Su (2004) informationsliefernde und umgebungsändernde Dienste. Klein u. a. (2005) unterteilen in Wissensdienste zur Abfrage von Informationen, Informationsdienste, mit denen Informationen in Wissensbasen verändert werden können, umgebungsändernde Dienste, die Einfluss auf den Zustand eines Objektes der realen Welt haben, und Befähigungsdienste, durch die Zugang zu bestimmten Fähigkeiten erlangt wird. Für die Verwendung in intelligenten Umgebungen lassen sich die letzten drei von Klein u. a. (2005) genannten Dienstarten dabei zu umgebungsändernden Diensten (im Sinne von McIlraith (2002) sowie Rao und Su (2004)) zusammenfassen.
Informationsliefernde und umgebungsändernde Dienste unterscheiden sich dadurch, dass informationsliefernde Dienste allein durch ihre Ein- und Ausgaben und möglicherweise zusätzlich durch Vorbedingungen beschrieben sind. Sie haben im Gegensatz zu umgebungsändernden Diensten keine Effekte. Aus Sicht der Planung ist ein Operator ohne Effekt jedoch sinnlos, und eine Unterteilung in informationsliefernde und umgebungsändernde Dienste daher nicht notwendig, da ein solcher Operator nie ein Ziel erreichen könnte und so auch nie Teil eines automatisch generierten Planes sein könnte. So muss in einem Planungsproblem ein Operator, der eine Information liefert, einen Effekt haben,

der im Zustandsübergangssystem hinterlegt, dass die gewünschte Information vorhanden ist. Eine Möglichkeit dafür ist das bereits erwähnte Prädikat agentHasKnowledgeAbout von Blankenburg u. a. (2008), was allerdings die beschriebenen Nachteile hat.
Noch ein weiterer Punkt spricht gegen eine Unterteilung in informationsliefernde und weltverändernde Dienste. Im Abschnitt 1.3 wurde gesagt, dass ein wesentliches Unterscheidungsmerkmal von intelligenten Umgebungen gegenüber Webservice-Umgebungen der zusätzliche räumliche Bezug der Dienste ist. Alle Dienste einer intelligenten Umgebung können einem Gerät zugeordnet werden, verbrauchen bei einer Ausführung dessen Ressourcen und haben so immer Auswirkungen auf die Umgebung. Selbst Dienste, die klar als informationsliefernder Dienst einzuordnen wären, bekommen so in intelligenten Umgebungen Effekte, wie beispielsweise die Verringerung der Batteriekapazität des Gerätes, auf dem sie installiert sind. In Webservice-Umgebungen wird eine Unterscheidung in informationsliefernde und weltverändernde Dienste vor allem getroffen, um erstere als gefahrlos ausführbar zu kennzeichnen. Innerhalb eines Prozesses können diese Dienste beliebig oft ausgeführt werden, um beispielsweise aktuelle Informationen, die für den weiteren Prozessverlauf von Belang sind, abzurufen. Innerhalb einer intelligenten Umgebung kann diese gefahrlose Ausführbarkeit jedoch nicht mehr gewährleistet bleiben, da die Geräte, auf denen die Dienste ausgeführt werden, Teil der Umgebung sind und somit im Zweifelsfall immer einen Effekt auf die Umgebungen haben.
Letztlich sind informationsliefernde Dienste vor allem dann von Interesse, wenn eine Online-Planungsmethode verwendet wird, die während der Planung Informationen aus der Umgebung anfordern kann, um unbekannte Zustände zu determinieren (siehe auch Abschnitt 6.2.2). Da im Rahmen dieser Arbeit keine Online-Planung eingesetzt wird, kann aus diesem und den vorherigen Gründen auf die Unterteilung in informationsliefernde und umgebungsverändernde Dienste verzichtet werden.
Zurückkehrend zur ursprünglichen Frage, ob ein Mapping von Dienstbeschreibung auf Planungsoperator direkt möglich ist, kann festgehalten werden, dass das Vorhandensein von Vorbedingungen und Effekten in der Beschreibung eines Dienstes zwingend notwendig ist, um eine Komposition von Diensten zu ermöglichen. Mit Hinblick auf die Tatsache, dass eine Erfassung von PEs in üblichen Gerätebeschreibungen zum jetzigen Zeitpunkt nicht vorhanden ist, erscheint diese Anforderung herausfordernd. Wie bereits angedeutet, besteht aber unter Umständen die Möglichkeit, aus der aktuellen Konstellation der Umgebung und den Signaturen vorhandener Dienste auf deren PEs zu schließen. Dabei könnte aus gegenwärtiger Situation, aktueller Nutzerintention und den Schnittstellenbeschreibungen der vorhandenen Dienste auf deren Verwendungszweck geschlossen werden. Zugrundeliegend wäre die Annahme, dass ein

Nutzer vor allem sinnvoll zu kombinierende Geräte in einer Umgebung zusammenbringt. Die Untersuchung dieser These muss jedoch zukünftigen Arbeiten vorbehalten bleiben, da es zu deren erschöpfender Überprüfung derzeit noch keine ausreichend umfangreichen Umgebungen gibt.

These 4 *Um eine KI-planungsbasierte Komposition von Diensten zu ermöglichen, ist eine explizite Beschreibung ihrer Vorbedingungen und Effekte zusätzlich zur Beschreibung der Dienstschnittstelle notwendig. Eine Unterscheidung in informationsliefernde und umgebungsändernde Dienste ist nicht nötig. Alle Dienste werden als potentiell umgebungsändernd angenommen.*

3.2.3 Zielsynthese für dynamische Komposition

Durch die Hinzunahme einer Intentionsanalyse wird eine intelligente Umgebung teleologisch, d.h. zweck- bzw. zielorientiert. Es ist also kein System der automatischen (evolutionären) Anpassung an die Umwelt mehr, sondern es agiert zielgerichtet (Zambonelli und Parunak, 2002, S.274). Auch klassische KI-Planung ist zielgerichtet.
Entscheidend für die Modellierung einer intelligenten Umgebung als Planungsproblem ist, welcher Art diese Ziele sind. Dabei können drei verschiedene Zieltypen unterschieden werden, funktionsbasiert, zustandsbasiert und aufgabenbasiert (vgl. Heider und Kirste (2002)). Aufgabe der Strategiesynthese, also der Kompositionsinstanz, ist es, ausgehend vom jeweils formulierten Ziel eine mögliche Aktionssequenz zu finden, die entweder die gewünschte Funktion bereitstellt, den gesuchten Zustand erreicht oder die gestellte Aufgabe erfüllt.

Funktionsbasiert Die funktionsbasierte Zielbeschreibung beruht auf der gegenwärtig üblichen Wahrnehmung von Multi-Media-Geräten. Dabei werden die angebotenen Funktionen der Geräte direkt ausgeführt, z.B. durch Betätigen der Tasten „Play“, „Pause“ oder „Stop“ an einem Wiedergabegerät. Durch das Wissen um die Beschränktheit von Informationsressourcen in der Vergangenheit, war es für einen Nutzer auch sinnvoll in Funktionen zu denken, denn diese Funktionen befanden sich im Realitätshorizont des Nutzers. Mit der zunehmenden Allgegenwärtigkeit von Information erweitert sich dieser Horizont mehr und mehr und ein Wunsch, wie z.B. „Ich möchte jetzt und hier einen bestimmten Film ansehen“ wird realistischer. Die dahinterstehende Annahme ist das Vertrauen darauf, dass in der gegenwärtigen Umgebung irgendein Gerät, was einen Videostream wiedergeben kann, vorhanden ist. Zusätzlich wird

dasselbe oder ein anderes Gerät in der Lage sein, eine Internetverbindung aufzubauen und eine Quelle zu finden, von der sich der Film herunterladen lässt. Es ist nicht mehr nötig, sich Gedanken um die Herkunft der Informationen zu machen. Es reicht eine grobe Ahnung, dass die Informationen und die nötige Funktionalität am aktuellen Ort in der gegenwärtigen Situation verfügbar sind.
Da nach der Vision von intelligenten Umgebungen der Nutzer vom Wissen um die Fähigkeiten einzelner Geräte befreit werden soll und zusätzlich Informationen und Funktionalitäten allgegenwärtig verfügbar sein sollen, wird von einer funktionsbasierten Zielbeschreibung abgesehen. Sie entspricht nicht der angestrebten Vision.

Zustandsbasiert Eine weitere Möglichkeit ist die zustandsbasierte Zielbeschreibung, welche auch von klassischer KI-Planung verwendet wird. Eine zustandsbasierte Zielbeschreibung besteht üblicherweise aus einer Konjunktion von Propositionen, wobei jede Proposition Eigenschaften der Umgebung oder Zustände der in der Umgebung befindlichen Objekte beschreibt.
Für eine Bewertung, ob eine zustandsbasierte Zielbeschreibung realistisch ist, muss sich vergegenwärtigt werden, wie diese Ziele im konkreten Fall aussehen und ob sie durch eine wie auch immer gestaltete Intentionsanalyse inferriert werden können.
Dazu soll auf das Beispiel des SmartLab vorgegriffen werden, welches im Abschnitt 4.1.6 näher erläutert wird. Dort ist in einer Umgebung, die mit Projektoren und Leinwänden ausgestattet ist, das Ziel formuliert, ein Dokument DOC1 auf einer Leinwand LW1 anzuzeigen. Die entsprechende Zielproposition heißt dort (isActive DOC1 LW1). Es sind verschiedene Szenarien denkbar, die zu diesem Ziel führen könnten:

1. *Bob zeigt auf Leinwand LW1 und sagt: „Mein Dokument DOC1 hier anzeigen!“*

2. *In der Agenda des Meetings ist Bob als nächster Redner verzeichnet und das Dokument für den Vortrag ist in der Agenda verlinkt. Eine Lokalisierung registriert, dass sich Bob zur Leinwand LW1 bewegt. Es kann so geschlossen werden, dass er seinen Vortrag dort beginnen möchte und dazu das Dokument angezeigt werden soll.*

3. *In einer Diskussion wird das Dokument DOC1 von mehreren Teilnehmern erwähnt und als aktuell relevant eingestuft. Die Leinwand LW1 befindet sich im Sichtbereich aller anwesenden Personen.*

Im ersten Szenario wird das Ziel verbal aber explizit angegeben. Es ist unzweideutig gefordert, dass DOC1 auf LW1 angezeigt werden soll. Die Intentionserkennung könnte durch eine Spracherkennung geleistet werden. Es muss lediglich sichergestellt sein, dass die Vokabel „anzeigen“ auf das Prädikat isActive referenziert.
Das zweite Szenario verlangt von der Intentionsanalyse die Fähigkeit, Inferenzen zu bilden. Aus den Fakten, dass Bob der nächste Redner ist und DOC1 als Bobs Vortrag in der Agenda hinterlegt ist und der Beobachtung, dass sich Bob auf Leinwand LW1 zu bewegt, muss geschlossen werden, dass DOC1 auf LW1 anzuzeigen ist.
Auch das dritte Szenario verlangt von der Intentionsanalyse die Fähigkeiten der Inferenzbildung, da das Ziel aus der Beobachtung, dass DOC1 häufig erwähnt wird und dem Fakt, dass LW1 im Blickfeld aller Personen ist, geschlossen werden muss.
Für die Verwendung von zustandsbasierten Zielbeschreibungen im Rahmen einer Dienstekomposition sprechen mehrere Punkte. Zunächst ist ein Mapping eines zustandsbasierten Zieles auf ein klassisches KI-Planungsziel trivial, sofern ein gemeinsames Vokabular von Zustandsvariablen verwendet wird. Auch die von Aiello u. a. (2002) vorgeschlagene Anfragesprache Web Service Request Language (WSRL) für komponierte Webservices verwendet zustandsbasierte Zielbeschreibungen. Weiterhin lassen sich realistische Szenarien finden, in denen das intuitive Verhalten des Nutzers eine zustandsbasierte Zielbeschreibung vorgibt oder sie inferiert werden kann. Schließlich abstrahiert sie von den Funktionen der verwendeten Geräte, unterstützt den Nutzer und leistet so einen Beitrag zur Vision der intelligenten Umgebungen.

Aufgabenbasiert Als dritte Möglichkeit können Ziele einer Umgebung aufgabenbasiert formuliert werden. Dabei sind Aufgaben abstrakte Beschreibungen von komplexem Verhalten der Umgebung. Ein Beispiel dafür wäre das Ziel (showMovie ‘‘Avatar’’) oder (givePresentation ‘‘Bob’’).
Auch Workflows können als aufgabenbasiete Zielbeschreibungen angesehen werden. Srivastava und Koehler (2003) unterscheiden in diesem Zusammenhang zwischen impliziten (aufgabenbasierten) und expliziten (zustandsbasierten) Zielen. Sie erläutern, dass im Prozessmanagement von Unternehmen meist keine expliziten Zielbeschreibungen existieren. Hier sind vor allem mit der Zeit gewachsene Beschreibungen von Workflows vorhanden. Die wahren, d.h. zustandsbasierten Ziele sind meist nur noch implizit in den Workflows enthalten. So ist zum Beispiel das implizite Ziel einer Routenplanung (vgl. Abschnitt 4.1.2) die exakte Route zu erhalten. Das explizite zustandsbasierte Ziel ist jedoch das Vorhandensein des Ausdrucks der Karte. Srivastava und Koehler (2003) argumentieren daher, dass es nicht trivial ist, implizite Ziele

explizit zu machen. Im letzten Abschnitt konnte jedoch gezeigt werden, dass in intelligenten Umgebungen Szenarien existieren, in denen explizite Ziele gegeben werden.
Eine Planungstechnik, die es erlaubt aufgabenbasierte Ziele zu planen, ist die HTN-Planung (siehe Abschnitt 3.1.3). Das Übersetzen impliziter zu expliziten Zielen wird dabei durch die HTN-Methoden übernommen. Dazu müssen diese Methoden jedoch modelliert werden, was zusätzlichen Modellierungsaufwand für die intelligente Umgebung bedeutet. Eine Verwendung aufgabenbasierter Ziele setzt daher voraus, dass bei der Strategiesynthese Wissen über die Aufgaben vorhanden ist und dass diese Aufgaben von der Intentionsanalyse identifiziert werden können. Damit verlagert eine aufgabenbasierte Zielbeschreibung einen Teil der Nutzerassistenz auf die Strategiesynthese. Des Weiteren müssen aber auch die Nutzer entsprechend handeln. Greift man das erste Beispiel aus dem obigen Abschnitt über die zustandsbasierte Zielbeschreibung auf, würde der Nutzer Bob nicht mehr artikulieren, dass sein Vortrag jetzt an seinem Standort zu sehen sein soll. Er würde nur noch sagen müssen, dass er jetzt einen Vortrag halten möchte. Das zweite Beispiel würde ein Modell voraussetzen, in dem ausgehend von der aktuellen Position der Menschen im Raum ihre gegenwärtige Absicht hergeleitet werden kann. In Burghardt und Kirste (2007) wird ein solches Modell beschrieben.
Eine aufgabenbasierte Zielbeschreibung hat offensichtlich mehr Freiheitsgrade als eine zustandsbasierte oder gar funktionsbasierte Beschreibung, lässt also noch unspezifischere Nutzerintentionen zu als eine zustandsbasierte Zielbeschreibung. Mit steigender Anzahl der Freiheitsgrade steigt aber auch der algorithmische Aufwand sowie die Menge der zu modellierenden Informationen, um das Problem automatisiert zu lösen. Im weiteren Verlauf der Arbeit wird daher von einer zustandsbasierten Zielbeschreibung ausgegangen.

3.3 Dienstzugriff

Im Abschnitt 3.1 wurde KI-Planung als geeignetes Kompositionsverfahren für intelligente Umgebungen vorgestellt. Aktuelle Planer verwenden dabei PDDL als intermediäres Datenaustauschformat.
Für eine Verwendung in intelligenten Umgebungen stellt sich nun die Frage, welche Sprachen zur Beschreibung von Diensten eingesetzt werden können und inwieweit diese sich auf PDDL abbilden lassen. Im Kapitel 2 wurde erwähnt, dass eine semantische Beschreibung von Diensten grundlegende Voraussetzung für dynamische Dienstekomposition ist. Bei der Verwendung von KI-Planung werden Dienste als Operatoren aufgefasst, deren Semantik durch Vorbedingungen und Effekte (PE) definiert ist. Wird KI-Planung als Kompositionsmethode verwendet, müssen Dienste daher über eine maschinenlesbare Beschreibung ihrer Vorbedingungen und Effekte verfügen (Martin u. a., 2004). In Abschnitt 3.1.4 wurde beschrieben, dass die Vorbedingungen eines PDDL Operators funktionsfreie Sätze der Prädikatenlogik erster Ordnung sind. Dabei hat PDDL annähernd $\mathcal{SOIN}$ Ausdrucksmächtigkeit (Blankenburg u. a., 2008). Daher muss eine Beschreibungssprache über ein logikbasiertes Komponentenmodell der Mächtigkeit $\mathcal{SOIN}$ verfügen oder eine Abbildung darauf zulassen. Des Weiteren muss die Beschreibung des Dienstes zustandsbasiert sein.
Für den Zugriff auf Geräte in einer intelligenten Umgebung sind verschiedene Beschreibungssprachen und Technologien mit jeweils unterschiedlichen Anwendungsgebieten und Zielsetzungen verfügbar. Wesentliche Impulse zur Komposition von Diensten kommen dabei aus dem Gebiet der Webservices. Eine Auswahl aktuell verwendeter und/oder diskutierter Dienstbeschreibungssprachen wird im Folgenden vorgestellt und anhand der eingangs genannten Anforderungen auf ihre Verwendbarkeit in intelligenten Umgebungen überprüft. Für eine umfassendere Erläuterung von Beschreibungssprachen wird auf die Arbeit von Klusch (2008) sowie auf die im Text referenzierten Standardisierungsdokumente verwiesen.
Teile der Evaluierung wurden in Zusammenarbeit mit Christiane Reiße und Christoph Burghardt erstellt und veröffentlicht (Reisse u. a., 2008a,b). Ebenda werden Beispiele für Beschreibungen von Vorbedingungen und Effekten in den Sprachen OWL-S und WSMO gegeben.

3.3.1 WSDL

WSDL (W3C, 2007b) ist eine XML-basierte Schnittstellenbeschreibungssprache für Webservices. Die erste Version der Sprache wurde 2001 bei der World Wide Web Consortium (W3C) eingereicht. Ziel war es, im Internet verfügbare

Informationen, die im Hypertext Markup Language (HTML) Quelltext der Webseiten integriert waren, maschinenlesbar zur Verfügung zu stellen, um so eine Trennung von dynamischem Inhalt und Präsentation der Daten zu erreichen (W3C, 2001). Heute ist der überwiegende Teil der öffentlich verfügbaren Webservices in WSDL beschrieben.
WSDL-Dienste sind durch die folgenden sechs Elemente definiert:

- *types* - definiert mit Hilfe von XML Schema Definition (XSD) die vom Dienst verwendeten Datentypen
- *message* - legt die Struktur der austauschbaren Nachrichten fest
- *portType* - ist eine Menge von abstrakten Operationen, bestehend aus Name, Ein- und Ausgaben und Fehlermeldungen, samt den verwendeten Nachrichtenformaten
- *binding* - beschreibt das für einen portType zu verwendende Protokoll und Datenformat
- *port* - spezifiziert die Adresse, unter der ein binding und damit ein portType zu erreichen ist
- *service* - fasst die verschiedenen ports zu einem Dienst

Unter einem Dienst (*service*) wird bei WSDL eine Menge von Operationen verstanden. WSDL lässt so polymorphe Methoden zu. So kann eine Funktionalität durch mehrere Operationen abgefragt werden. Beispielsweise kann ein Wettervorhersagedienst über eine Ortsangabe oder eine Postleitzahl aufgerufen werden. Andererseits kann die Ausführung mehrerer Methoden nötig sein, um die Funktionalität eines Dienstes bereitzustellen. Für die Komposition ist aber das Zusammenspiel der Aktionen der Geräte von Interesse, die im Fall von WSDL den *ports* des Dienstes entsprechen. Ein WSDL-Dienst beschreibt daher aus Sicht der Komposition in intelligenten Umgebungen eine Sammlung von Einzeldiensten, die gemeinsame Datentypen und Nachrichtenformate verwenden.
Für einen *portType* werden Eingaben, Ausgaben und Fehlerrückgaben definiert. Sie verwenden die vorher eingeführten Nachrichtenformate und Typen. Die Beschreibung ist nachrichtenbasiert und rein syntaktisch. Eine semantische Beschreibung und eine Beschreibung des Zustandes eines Dienstes ist nicht möglich und macht eine direkte Verwendung von WSDL-basierten Dienstbeschreibungen für die KI-planungsbasierte Komposition daher unmöglich.

SAWSDL (Kopecký u. a., 2007) ist eine Erweiterung von WSDL, die semantische Annotationen zulässt. Mit SAWSDL können Datentypen, Nachrichten und Ports semantisch annotiert, d.h. auf eine Ontologie verlinkt werden. Eine vollständige semantische Beschreibung der Funktionalität des Dienstes, wie sie für eine Komposition basierend auf KI-Planung benötigt wird, ist damit jedoch noch nicht geleistet.
Zur Beschreibung REST-basierter Webservices existiert neben WSDL die Sprache Web Application Description Language (WADL) (Hadley, 2009). Jedoch ist auch hier eine Beschreibung der Vorbedingungen und Effekte nicht möglich.

3.3.2 OWL-S

OWL-S(Martin u. a., 2004) wurde auf Basis der Ontologiesprache Web Ontology Language (OWL) entwickelt und stellt eine semantische Beschreibungssprache für Dienste dar. OWL beruht auf Prädikatenlogik erster Stufe mit der Aussdrucksmächtigkeit $\mathcal{SHOIN}$ und ist entscheidbar.[10]
Mit OWL-S lassen sich sowohl einzelne Dienste, als auch Prozessmodelle darstellen. Dabei zielte die OWL-S-Initiative in erster Linie auf die Beschreibung von Webservices ab. Die Dienstekomposition war eine Motivation bei der Entwicklung der Sprache. An der Formulierung von OWL-S waren mehrere Wissenschaftler beteiligt, die im Bereich der KI-Planung forschen.
OWL-S besteht aus drei Teilen, dem *ServiceProfile*, dem *ServiceGrounding* und dem *ServiceModel*.
Das *ServiceProfil* beschreibt, was ein Dienst tut. Es dient dem Auffinden des Dienstes. Im Profil wird ähnlich wie bei WSDL die Schnittstelle des Dienstes beschrieben. Daneben können aber auch funktionale Eigenschaften, wie die Vorbedingungen und Effekte des Dienstes beschrieben werden. Zu diesem Zweck sind die Elemente *hasPrecondition* und *hasResult* vorgesehen.[11] Die Entwickler von OWL-S lassen ausdrücklich offen, wie Vorbedingungen und Effekte innerhalb dieser Elemente zu formulieren sind. Als kleinstmöglicher Kompromiss wurde sich darauf geeinigt, dass sie entweder als XML- oder als String-Literale hinterlegt sind, somit also nicht aus OWL-konformen Elementen bestehen müssen. Es können sowohl Sprachen aus dem Semantic-Web-Umfeld wie Semantic Web Rule Language (SWRL) oder Resource Description Framework (RDF) als auch anderen Sprachen wie Knowledge Interchange

[10] Es ist zu beachten, dass drei verschiedene Varianten von OWL (OWL-Lite, OWL-DL und OWL-Full) über jeweils unterschiedliche Ausdrucksmächtigkeiten verfügen. Im Rahmen dieser Arbeit ist immer OWL-DL gemeint, wenn von OWL gesprochen wird.

[11] Da Dienste sehr oft Effekte haben, die von den Ausgaben des Dienstes abhängen, werden in OWL-S die Effekte nicht direkt angegeben, sondern zusammen mit Bedingungen an die Ausgaben des Dienstes als „Results“ beschrieben. Ein „Result“ ist somit durch eine Menge bedingter Effekte definiert.

Format (KIF) und PDDL zur Beschreibung der Vorbedingungen und Effekte verwendet werden. Soll PDDL als Grundlage zur Komposition verwendet werden, muss ein Mapping von OWL-S auf PDDL (Gerber u. a., 2005; Blankenburg u. a., 2008; Hatzi u. a., 2009) durchgeführt werden. Sind Vorbedingungen und Effekte innerhalb der OWL-S-Beschreibung bereits mit Hilfe von PDDL formuliert, ist dieses Mapping trivial.
Im *ServiceGrounding* wird der Zugriff auf den Dienst beschrieben. Dieser Teil der Dienstbeschreibung wird für die Ausführung des Dienstes benötigt. Im Rahmen einer Komposition wird das Grounding eines Dienstes verwendet, um ein Prozessmodell in eine Aktionssequenz zu überführen.
Der dritte Teil, das *ServiceModel*, ist eine Prozessbeschreibung, in der hinterlegt ist, wie der Dienst arbeitet. Es ermöglicht, einen Dienst nicht nur als atomar ausführbare Funktionalität zu betrachten sondern ihn vielmehr als Prozess, der ggf. aus anderen Diensten besteht zu beschreiben. Dabei werden atomare (atomic) und komponierte (composite) Prozesse unterschieden.[12] Wird ein Dienst als komponierter Prozess beschrieben, erfolgt dies als statische Prozessbeschreibung, die der von Workflowbeschreibungssprachen wie BPEL4WS ähnelt. Allerdings verfügt BPEL4WS über wesentlich mehr Kontrollstrukturen zur Definition von Prozessen. In verschiedenen Veröffentlichungen werden OWL-S und dessen Vorgänger DAML-S zur Beschreibung von Prozessmodellen verwendet. Beispiele dazu wurden im Abschnitt 2.3 gegeben.

3.3.3 WSMO

Eine weitere Sprache zu Beschreibung von Webservices ist WSMO (Lausen u. a., 2005; Roman u. a., 2005). WSMO ist ein europäisches Forschungs- und Entwicklungsprojekt, das sich in drei Arbeitsgruppen (WSMO, Web Service Modelling Language (WSML) und Web Service Execution Environment (WSMX)) unterteilt. WSMO entwickelt ein konzeptionelles Modell für die Beschreibung von semantischen Webservices. WSML formalisiert die konkrete, F-Logik-basierte (Kifer und Lausen, 1989) Sprache zur Beschreibung der Dienste. WSMX befasst sich mit der Umsetzung einer Ausführungsumgebung für WSMO-Dienste.
Die Beschreibung von Diensten wird durch WSMO definiert. Vorbedingungen und Effekte können darin in der funktionalen Beschreibung des Dienstes („capability“) beschrieben werden. Diese Beschreibung erfolgt in F-Logik. Im Unterschied zu klassischen Beschreibungslogiken beinhaltet F-Logik Semantik. So gibt es Elemente der F-Logik, die beispielsweise die Zugehörigkeit eines Objektes zu einer Klasse darstellen, die in einer Beschreibungslogik keine

[12] Daneben können noch abstrakte Prozesse (simple process) spezifiziert werden, der hier jedoch nicht weiter betrachtet werden soll.

Entsprechung haben und sich nur auf ein spezielles Prädikat (in diesem Fall z.B. isA(object, class)) abbilden lassen. Des Weiteren ist F-Logik im Gegensatz zu OWL unentscheidbar. Die Beschreibung einer Abbildung von WSMO-Diensten auf PDDL zur klassischen KI-Planung konnte in der Literatur nicht gefunden werden, wenngleich sie aufgrund der Abbildbarkeit von F-Logik auf generelle Beschreibungslogiken (Bruijn und Heymans, 2008) möglich sein sollte. Tabatabaei u. a. (2008) erläutern lediglich eine Verwendung von WSMO zur HTN-Planung, ohne jedoch auf PDDL einzugehen.
Neben weiteren Unterschieden (Lara u. a., 2004) geht WSMO im Gegensatz zu OWL-S sehr viel stärker auf das semantische Finden der Dienste („discovery“) als zwingende Voraussetzung für erfolgreiche Komposition von Webservices ein. Da WSMO neuer als OWL-S ist, verfügt es über eine noch schlechtere Durchdringung in Forschung und Industrie.

3.3.4 DSD

DSD ist eine im Rahmen des Projektes Dienste in ad-hoc Netzen (DIANE) entwickelte Dienstbeschreibungssprache, die im Gegensatz zu den bisher beschriebenen Sprachen komplett zustandsbasiert ist. DSD basiert auf der Ontologiesprache DIANE Elements (Klein, 2004) und damit weder auf Aussagenlogik oder Prädikatenlogik wie OWL-S, noch auf F-Logik wie WSMO (Klein u. a., 2005). Grundlage der Sprache bildet eine Sammlung von ontologischen Konzepten, mit der Dienste beschrieben werden können. Fokus bei der Entwicklung der Sprache lag auf dem Discovery der Dienste.
Während WSDL und OWL-S für die Beschreibung eines Dienstes (beim publish) und die Anfrage nach einem Dienst (beim find) das gleiche Format verwenden, unterscheidet DSD explizit die Beschreibungen von Dienstangeboten und Dienstanfragen. Ähnlich zu OWL-S besteht ein Dienst aus einem Service Profil und einem Grounding. Dabei wird das funktionale Verhalten eines Dienstes im Profil ausschließlich durch Vorbedingungen und Effekte beschrieben.

3.3.5 UPnP

Universal Plug and Play (UPnP) ist ein standardisiertes (ISO / IEC 29341 (ISO, 2008)) Rahmenwerk für verteilte Systeme, das auf dem Transmission Control Protocol/Internet Protocol (TCP/IP) basiert. Es stammt ursprünglich aus dem Bereich der Hausautomation und wurde in erster Linie für die reibungslose und konfigurationsfreie Installation und automatische Erkennung von Geräten innerhalb eines lokalen Netzwerks konzipiert. Dabei liegt der Fokus vornehmlich auf dem Peer-to-Peer-Zugriff auf Multimediageräte (Kist-

ler u. a., 2008). UPnP spezifiziert die Funktionen verschiedener Gerätetypen, wie zum Beispiel Internet-Gateways, Audio-Video-Geräte, Drucker, Scanner, Klima- und Beleuchtungssteuerung sowie Wireless-LAN-Access-Points und digitale Kameras. Daneben werden auch erweiterte Funktionen wie Sicherheit, Remote-Benutzer-Schnittstelle und die Qualität der Dienstleistung betrachtet. UPnP verwendet wie auch Webservices SOAP-Nachrichten für den Datenaustausch zwischen Diensten. Während jedoch Webservices WSDL-Dokumente zur Beschreibung der Dienstschnittstelle benutzen, greift UPnP hier auf ein eigenes XML-basiertes Dienstbeschreibungsformat zurück, wobei ein Mapping zwischen beiden Formaten möglich ist (Kistler u. a., 2008).
Eine explizite Beschreibung von Vorbedingungen und Effekten einzelner Geräte und deren Funktionen ist in dieser Beschreibung nicht vorhanden. Es wäre möglich für die durch UPnP vorgegebenen Diensttypen feste Vorbedingungen und Effekte zu definieren, so könnten die Funktionen von UPnP-Geräten als Planungsoperatoren dargestellt und auch komponiert werden. Eine solche Generalisierung ist allerdings kritisch zu sehen, da die durch UPnP vorgegebene Klassifizierung zu allgemein ist (siehe Abschnitt 8.3.1).

3.3.6 DPWS

Die Beschreibungssprache Device Profile for Web Services (DPWS) (Bohn u. a., 2006; Driscoll und Mensch, 2009) ist eine Weiterentwicklung von UPnP, die versucht, die Lücke zwischen Webservices und Sprachen für lokale Umgebungen zu schließen und so die Vorteile beider Spezifikationen in intelligenten Umgebungen nutzbar zu machen. Ziel ist es, Webservices auf eingebetteten Geräten mit eingeschränkten Ressourcen zu standardisieren. DPWS unterscheidet zwei Arten von Diensten, *Hosting Services* und *Hosted Services*. Hosting Services repräsentieren ein Gerät. Sie sind innerhalb eines Netzwerkes direkt auffindbar. Die Funktionalitäten der Geräte werden durch Hosted Services angeboten. Dabei besteht eine 1:n Kardinalität zwischen *Hosting Service* und *Hosted Service*, da ein Gerät mehrere Funktionalitäten anbieten kann, eine Funktionalität jedoch immer fest an ein Gerät gebunden ist. Zur Beschreibung eines *Hosted Service* dienen dessen Name, die Schnittstelle und Metadaten. Eine semantische Beschreibung durch Vorbedingungen und Effekte des Dienstes kann zwar im Rahmen der beschreibenden Metadaten erfolgen, ist aber in der Beschreibung von DPWS nicht enthalten.

3.3.7 Eignung aktueller Beschreibungssprachen

Neben den vorgestellten Sprachen existieren weitere, teilweise ältere Sprachen und Methoden zur Beschreibung von und zum Zugriff auf Dienste (z.B.

Open Services Gateway initiative (OSGi), Common Object Request Broker Architecture (CORBA), ebXML, RosettaNet, Distributed Component Object Model (DCOM), Enterprise Java Beans (EJB), Jini, Bluetooth, ZigBee u.a.). Diese Ansätze haben teilweise sehr unterschiedliche Zielstellungen und Anwendungsgebiete; CORBA, ebXML, RosettaNet sind vorrangig Werkzeuge zur Business-to-Business (B2B) Integration; DCOM, EJB und Jini sind komponentenbasierte, programmiersprachen- und/oder plattformabhängige Rahmenwerke zur Realisierung verteilter Anwendungen; Bluetooth und ZigBee definieren auf den unteren Schichten des Protokollstapels Standards für den drahtlosen Zugriff auf (Klein)Geräte. Dennoch befassen sich alle mit jeweils unterschiedlichen Schwerpunkten mit Diensten. Die genannten Ansätze wurden im Rahmen dieser Arbeit nicht weiter betrachtet, da ihre Eignung für die hier behandelte Problemstellungen als ungenügend angesehen wird. Aufgrund der Fülle der Spezifikationen kann diese Einschätzung jedoch nicht abschließend sein.

Die Tabelle 3.1 fasst die Fähigkeiten der untersuchten Sprachen hinsichtlich der KI-planungsbasierten Dienstekomposition zusammen. Die Spalte „Vorbedingungen und Effekte“ zeigt an, ob in der jeweiligen Sprache Vorbedingungen und Effekte eines Dienstes beschrieben werden können. Ist dies der Fall, wird in der dritten Spalte das zugrundeliegende Komponentenmodell der jeweiligen Sprache genannt, andernfalls entfällt dieses Merkmal. Die vierte Spalte listet die Art der Dienstbeschreibungen auf. In der letzten Spalte ist vermerkt, ob in der Literatur ein Mapping der jeweiligen Sprache zu PDDL gefunden werden konnte. Erfüllt eine Sprache eine dieser Kriterien, ist dies durch Fettdruck hervorgehoben.

Aus der Tabelle wird ersichtlich, dass keine der untersuchten Sprachen alle Anforderungen erfüllt. Lediglich drei Sprachen unterstützen die explizite Formulierung von PEs. In den anderen Sprachen sind zwar zusätzliche Annotationen vorstellbar, um PEs zu hinterlegen. Eine einheitliche und interoperable Verwendung dieser semantischen Beschreibung ist so jedoch nicht gewährleistet.

Hinsichtlich des Komponentenmodells unterscheiden sich die drei in Frage kommenden Sprachen OWL-S, WSMO und DSD. OWL-S verwendet eine Beschreibungslogik der Ausdrucksmächtigkeit $\mathcal{SHOIN}$. Abbildungen von OWL-S auf PDDL sind beschrieben (Klusch, 2008), standen im Rahmen dieser Arbeit allerdings nicht zur Verfügung. WSMO benutzt F-Logik, die zwar auf Beschreibungslogiken abbildbar ist (Bruijn und Heymans, 2008), eine direkte Abbildung auf PDDL konnte jedoch in der Literatur nicht gefunden werden. DSD umfasst verschiedene Repräsentationen, eine dieser Repräsentationen erlaubt es DSD Beschreibungen in OWL-S umzuwandeln. Daher sollte auch eine Abbildung auf PDDL möglich sein.

Sprache	PEs	Komponenten-modell	Beschrei-bungsart	PDDL-Mapping
WSDL (W3C, 2007b)	nein	-	nachrichten-basiert	nein
SAWSDL (Kopecký u. a., 2007)	nein	-	nachrichten-basiert	nein
OWL-S (Martin u. a., 2004)	**ja**	**Prädikatenlogik 1ter Stufe**	nachrichten- und zustandsbasiert	**ja**
WSMO (Roman u. a., 2005)	**ja**	F-Logik	nachrichten- und zustandsbasiert	nein
DSD (Klein, 2004)	**ja**	DIANE Elements	**zustandsbasiert**	nein
UPnP (ISO, 2008)	nein	-	nachrichten- und ereignisbasiert	nein
DPWS (Bohn u. a., 2006)	nein	-	nachrichten- und ereignisbasiert	nein

Tabelle 3.1: Vergleich von Dienstbeschreibungssprachen zur KI-planungsbasierten Dienstekomposition

Der überwiegende Teil der Beschreibungssprachen für Dienste verwendet eine nachrichtenbasierte Beschreibung. Für die korrekte und vollständige Komposition ist allerdings eine zustandsbasierte Beschreibung der Dienste nötig. Lediglich DSD erfüllt dieses Kriterium. Die getrennte Betrachtung von Nachrichten und Zustand, wie sie von OWL-S und WSMO verwendet wird, kann problematisch werden, wenn beispielsweise mehrere Nachrichten an einen Dienst nötig sind, um eine Zustansänderung herbeizuführen. Im Rahmen dieser Arbeit wird davon ausgegangen, dass atomare Dienste so feingranular implementiert sind, dass sie jeweils nur eine Nachricht zur Ausführung benötigen.

OWL-S stimmt in drei von vier Punkten mit den Anforderungen überein. Dennoch ist die praktische Anwendbarkeit von OWL-S kritisch zu sehen. Nach wie vor ist das Haupthindernis beim praktischen Einsatz von OWL-S dessen kaum vorhandene Verbreitung innerhalb der Industrie (Klusch und Zhing, 2008). Gründe dafür sind insbesondere die hohe Komplexität der Sprache und die ungenügende Verfügbarkeit von Werkzeugen (Balzer u. a., 2004). Mit einer Abbildung von DSD auf PDDL, wird auch eine Verwendung von DSD zur KI-planungsbasierten Dienstekomposition möglich. Allerdings ist DSD noch in der Entwicklung und wird daher hauptsächlich im wissenschaftlichen Bereich eingesetzt.

Die in der Literatur verfügbaren Anwendungen von OWL-S, WSMO und DSD bleiben größtenteils theoretisch und akademisch. Es bleibt abzuwarten, ob sich eine der Sprachen in der Praxis durchsetzen kann und ob PEs von Diensten darin hinterlegt werden. Für die Verwendung in der Dienstekomposition sind alle drei geeignet, wobei jeweils unterschiedliche Anstrengungen nötig sind,

um eine Übersetzung auf PDDL zu realisieren.

These 5 *Aus Sicht der KI-planungsbasierten Dienstekomposition können typische Dienstbeschreibungssprachen zur Beschreibung von Diensten in intelligenten Umgebungen verwendet werden. Sie müssen zur Nutzung durch die Planer in PDDL umgewandelt werden.*

Kapitel 4

Modellierung von Kompositionsproblemen

4.1 Beispielszenarien

Im Abschnitt 3.1.4.1 wurde PDDL anhand eines einfachen Beispiels als Modellierungsprache vorgestellt. In diesem Kapitel werden drei weitere Szenarien aus der Literatur mit Hilfe von PDDL modelliert. Dabei wird sich wie schon im einführenden Beispiel auf Sprachkonstrukte der PDDL-Version 1.2 beschränkt.
Die im Folgenden beschriebenen Szenarien sind Veröffentlichungen aus dem Bereich der intelligenten Umgebungen entliehen. Zwar beschäftigen sich die meisten der betrachteten Publikationen nicht direkt mit Dienstekomposition oder der Strategiesynthese, sondern mit überschneidenden oder generelleren Themen. Dennoch lassen sich mit den Szenarien wesentliche Probleme veranschaulichen, die auch bei der Dienstekompostion auftreten. Aus diesem Grund werden nur die Aspekte aus den Szenarien verwendet, die im Hinblick auf Dienstekomposition relevant sind. Bei der Modellierung sollen alle Operatoren Diensten einer intelligenten Umgebung entsprechen und als solche theoretisch umsetzbar sein.
Im Unterschied zum einführenden Beispiel werden zur Beschreibung der folgenden Szenarien typisierte Variablen verwendet. Typisierung erleichtert die Lesbarkeit einer Domänenbeschreibung und erlaubt eine wesentlich bessere Leistungsoptimierung während der Planung. Dazu muss :typing in den Bedingungen einer PDDL-Domäne aufgelistet sein. Ein getyptes Prädikat, um beispielsweise anzuzeigen, dass ein Gerät angeschaltet ist, wird in PDDL wie folgt beschrieben (isOn ?d − Device). Name ?d und Typ der Variablen Device sind durch einen Bindestrich getrennt. Die in einer Domäne verwendeten Typen müssen im Kopf der PDDL-Domäne deklariert werden. Eine alternative Art der Modellierung von Objekttypen ist die Definition von Typprädikaten wie z.B. (isPerson ?p) oder (isDevice ?d).
Alle Szenarien wurden vollständig in PDDL ausformuliert und erfolgreich mit dem Planer LPG (Gerevini und Serina, 2002) getestet. Aus Platzgründen sind die Beschreibungen der Operatoren im folgenden Abschnitt gekürzt und entsprechen nicht exakt der Syntax von PDDL. Die Definitionen der Vorbedingungen und Effekte wurden ausgelassen und die Syntax der Parameterdefinition der Operatoren wurde etwas vereinfacht. Eine vollständige und lauffähige PDDL-Beschreibung der Probleme findet sich im Anhang A. Für jedes Szenario wird der resultierende partiell geordnete Plan genannt.

4.1.1 Smart House

In Nishikawa u. a. (2006) wird ein intelligentes Haus beschrieben, in dem Alice und ihre Eltern wohnen. Es gibt mehrere Räume, in denen sich Lampen, ein

Fernseher, ein Videorecorder, eine Stereo- und eine Klimaanlage befinden. Weiterhin sind verschiedene Sensoren im Haus installiert, um Lichtintensität, Temperatur und Luftfeuchtigkeit zu messen. Alle Geräte sind durch ein Netzwerk miteinander verbunden. Es werden folgende Anwendungsszenarien beschrieben:
Sobald Alice ihr Zimmer betritt, wird das Licht eingeschaltet und die Stereoanlage beginnt leise Jazzmusik zu spielen. Die Klimaanlage stellt sich auf 25° C und 50% Luftfeuchtigkeit.
Sobald ihr Vater und ihre Mutter den Raum betreten, wird das Licht heller und die Klimaanlage auf 27° C sowie 60% Luftfeuchtigkeit eingestellt. Die Stereoanlage beginnt klassische Musik zu spielen.
Wenn der Vater im Wohnzimmer ist, wird, sobald eine Baseballübertragung beginnt, automatisch der Fernseher mit dem entsprechenden Sender eingeschaltet. Ist der Vater nicht im Wohnzimmer, nimmt der Videorekorder das Spiel auf.
Es ist ersichtlich, dass in diesen Szenarien keine nennenswerten Aktionsketten erzeugt werden müssen. Die meisten Ziele lassen sich in einem Schritt erreichen. Es steht vielmehr die Echtzeitsteuerung des Systems im Vordergrund. Da Dienstekomposition erst interessant wird, wenn Aktionssequenzen generiert werden, die aus mehreren Schritten bestehen, wird an diesem Punkt klar, dass Dienstekomposition hier (noch) nicht nötig ist.

4.1.2 Toms Routenplanung

In diesem Szenario (Fujii und Suda, 2004) wird deutlich, welche Probleme die Annahme 4 (siehe Abschnitt 3.1), der statische Weltzustand, bei der Modellierung von nachrichtenbasierten Diensten verursacht.
Tom möchte die Route zu einem Restaurant, das er im Internet gefunden hat, automatisch ausdrucken. Er möchte sich dabei nicht mit Details, wie dem Suchen der Adresse des Restaurants oder einem Routenplaner beschäftigen.
Aufgrund der Szenariobeschreibung wird nicht klar, wie Toms Wunsch, die Route zu bekommen, ausgedrückt ist und welche Informationen initial vorhanden sind. Es wird daher angenommen, dass eine Spracherkennung das Ziel aus Toms verbalen Äußerungen ableiten kann. Des Weiteren wird angenommen, dass Toms eigene Adresse für den Planer verfügbar ist.

Operatoren:

```
getAddress(?url - Data) ;Ein Dienst, der Adressen extrahieren kann
getRoute(?a1 ?a2 - Location  ?d - Document) ;Ein Routenplaner
;der ein Bild der angefragten Route zurückgibt
switchOn(?p - Printer) ;Ein Dienst, der Toms Drucker einschaltet
printDocument(?d - Document ?p - Printer) ;Der Druckerdienst
```

Ziel:

```
(and (isRoute DOC TOMSADDR RESTAURANTSADDR)(isPrinted DOC))
```

Initialer Weltzustand:

```
(isEmptyDocument DOC)
(isAddress TOMSADDR)
(isStart TOMSADDR)
(isURL RESTAURANTSADDR)
(isDestination RESTAURANTSADDR)
```

Plan:

```
0 (GETADDRESS RESTAURANTSADDR)
0 (SWITCHON TOMSPRINTER)
1 (GETROUTE TOMSADDR RESTAURANTSADDR DOC)
2 (PRINTDOCUMENT DOC TOMSPRINTER)
```

Bemerkungen: Dieses Szenario verdeutlicht das Problem, dass aufgrund des statischen Zustandsraumes während eines Planungslaufes keine neuen Objekte erzeugt werden können (vgl. Annahme 4). Daher ist es nicht möglich, ein neues Dokument als Effekt des Operators getRoute zu generieren. Statt dessen muss zu Beginn der Planung ein leeres Dokument initialisiert werden, welches durch den Operator getRoute mit dem Ergebnis der Routenplanung gefüllt wird. Dazu muss der Operator jedoch um einen zusätzlichen Parameter für das leere Dokument erweitert werden. Die beschriebene Modellierung verlangt das Vorhandensein von leeren Dokumenten in der Domäne, was allen anderen Operatoren der Domäne, die mit Dokumenten arbeiten, bekannt sein muss. So muss der Operator printDocument über eine Vorbedingung (**not** (isEmpty ?d)) verfügen, die (noch) nicht existente Dokumente ausfiltert, da diese offensichtlich nicht gedruckt werden können.

Eine weitere Herausforderung dieses Szenarios verbirgt sich in der Definition der Zielbeschreibung. Ziel ist es, eine Route zu erhalten. Da Dienste einer intelligenten Umgebung ad-hoc zusammenarbeiten sollen, dürfen sie kein globales Wissen in den Effekten enthalten. Der vorhandene Drucker hat daher lediglich den generellen Effekt (isPrinted ?doc) aber beispielsweise nicht den speziellen Effekt (isPrintedRoute ?from ?to). Daher kann auch die Zielbeschreibung nicht aus einem solchen speziellen Prädikat bestehen, sondern kann sich lediglich aus Prädikaten zusammensetzen, die in den Effekten von Operatoren vorhanden sind.

4.1.3 Janes Email

Dieses Szenario stammt aus dem Projekt Aura (Garlan u. a., 2002). Es veranschaulicht zum einen die Beschränkungen, die eine rein boolsche Beschreibung der Welt, wie sie die klassische KI-Planung verlangt, mit sich bringt (vgl. Abschnitt 3.1 und Annahme 1). Zum anderen wird verdeutlicht, dass automatisierte Dienstekomposition keine menschlichen Aktionen beinhalten darf.
Jane ist auf einem Flughafen und wartet am Gate 12 auf ihren Flug. Sie möchte, während sie wartet, eine sehr große Datei per Email versenden. Das WLAN an Janes Abfluggate ist überlastet und die Übertragung der Dateien würde nicht vor dem Boarding abgeschlossen sein. Am Gate 5 ist der WLAN-Durchsatz besser. Der Weg zu diesem Gate ist nicht zu weit. Jane kann dorthin laufen, ihre Email versenden und dann zurück zum Gate 12 laufen und ist trotzdem pünktlich zum Boarding.

Operatoren:

```
connectWLAN(?l - Laptop ?loc - Location ?t - Thruput) ;Ein Dienst auf einem
;Laptop, der sich mit einem WLAN verbinden und dessen Durchsatz messen kann
sendMail(?e - Email ?l - Laptop ?t - Thruput) ;Ein Emaildienst, der Emails
;verschicken kann und überprüft, ob der Durchsatz für eine Übertragung
;ausreichend ist.
move(?from ?to - Location ?l - Laptop) ;Janes Bewegung
checkIn(?l - Laptop ?loc - Location) ;Dieser Operator lässt Jane einchecken.
```

Initialer Weltzustand:

```
(at GATE12 JANESLAPTOP)
(isMovable JANESLAPTOP)
(requiredThruput JANESEMAIL HIGHT)
(availableThruput GATE12 LOWT)
(availableThruput GATE5 HIGHT)
```

Ziel:

```
(and (mailSent JANESEMAIL)(checkedIn GATE12))
```

Plan:

```
0 (MOVE GATE12 GATE5 JANESLAPTOP)
1 (CONNECTWLAN JANESLAPTOP GATE5 HIGHT)
2 (SENDMAIL JANESEMAIL JANESLAPTOP HIGHT)
2 (MOVE GATE5 GATE12 JANESLAPTOP)
3 (CHECKIN JANESLAPTOP GATE12)
```

Bemerkungen: Aufgrund der Annahme, dass Jane ihren Laptop in einem Flughafen nicht allein stehen lassen wird, wurde in diesem Szenario aus Gründen der Vereinfachung nur ihr Laptop modelliert. Trotzdem können die Operatoren move und checkIn nicht als Dienste implementiert werden, da sie Aktionen einer Person beinhalten. Es könnten Jane jedoch durch entsprechende Dienste Hinweise gegeben werden, bestimmte Aktionen zu tun, in der Hoffnung, dass sie diesen Empfehlungen folgen wird. Generell ist festzustellen,

dass, sobald in einem Operator menschliches Handeln benötigt wird, das Ergebnis eines Kompositionslaufes lediglich als Empfehlung denn als vollständiger Plan, der durch Geräte ausgeführt werden kann, zu verstehen ist (Simpson u. a., 2006). Aus diesem Grund kann dieses Szenario nicht durch Dienstekomposition allein gelöst werden.
Im aktuellen Szenario musste explizit gemacht werden, dass das Boarding des Flugzeugs, die letzte Aktion im Plan sein muss. Dazu wurde das Prädikat (isMovable ?laptop) verwendet, welches anzeigt, ob sich der Laptop (resp. Jane) noch bewegen kann. Es wird von der Aktion checkIn ungültig gemacht. Ohne ein solches Prädikat würde der Planer die Aktion checkIn ausführen, bevor Jane zu Gate 5 und zurück gegangen ist, um die Email abzuschicken.
Noch ein weiterer Punkt wird durch dieses Szenario deutlich. Ohne Erweiterungen ist es nicht möglich, die Dauer von Aktionen zu beschreiben (vgl. Annahme 7). In diesem Szenario wurde daher von sämtlichen Zeitspannen, wie zum Beispiel die Dauer des Sendens der Email oder die Dauer des Fußweges zwischen den Gates, abstrahiert.
Daneben wird durch dieses Szenario deutlich, dass es unter der Annahme, nur lokales Wissen zu verwenden, sehr aufwändig ist, alle möglichen Effekte einer Aktion zu modellieren. So bewirkt beispielsweise das Bewegen des Laptops von einem Gate zum nächsten eine Unterbrechung der WLAN Verbindung. Im sogenannten Briefcase-Problem sind ähnliche Effekte beschrieben. Sie sind nur mit Hilfe bedingter Effekte zu modellieren (Weld, 1994).

4.1.4 Barts Anweisung

Das folgende Szenario wurde im Rahmen des Projektes Daidalos (Yang u. a., 2006) entwickelt. Es verdeutlicht ein weiteres Modellierungsproblem, das durch den endlichen Zustandsraum bedingt ist (siehe Annahme 1). Innerhalb eines Planungslaufes kann der Typ eines Objektes nicht geändert werden, daher müssen Dienste, die Dokumente konvertieren können, gesondert modelliert werden.
Bart ist zu Hause und sieht die Nachrichten im Fernsehen, als sein Vorgesetzter ihn via VoIP anruft. Der Anruf wird auf Barts PDA umgeleitet. Während des Anrufes wird die Nachrichtensendung für Bart pausiert. Der Vorgesetzte beauftragt ihn, einen Kunden vom Flughafen abzuholen. Nachdem der Anruf beendet ist, werden die Nachrichten fortgesetzt. Als Bart in sein Auto steigt, wird die Nachrichtensendung als Audioversion abgespielt. Am Flughafen angekommen, wird auf dem Bildschirm des Navigationssystems ein Fluginformationsservice angezeigt. So kann er verfolgen, wann und wo der Kunde landet.
Das Szenario beschreibt einen iterativen Prozess und sollte in mehrere Abschnitte unterteilt werden. In jedem Abschnitt müssen neue Ziele erfüllt oder

neue äußere Umstände in Betracht gezogen werden. Daher kann jeder Abschnitt als eigenes Kompositionsproblem angesehen werden. Ein erstes Ziel ist es, den Anruf auf Barts PDA annehmen zu können. Das zweite Ziel ist, den Kunden vom Flughafen abzuholen. Als drittes Ziel soll Bart während der gesamten Zeit die Nachrichten verfolgen können.
Eine wesentliche Herausforderung dieses Szenarios ist es, die Ziele und Weltzustände entsprechend der aktuellen Ereignisse des Szenarios anzupassen und aktuell zu halten. So muss zum Beispiel nach dem Anruf (automatisch) inferiert werden, dass sich ein neues Ziel (den Kunden vom Flughafen abzuholen) ergeben hat.
Der aus Sicht der Dienstekomposition interessanteste Aspekt dieses Szenarios ist das Transferieren der Nachrichtensendung auf Barts Autoradio. Die Schwierigkeiten dieses Aspektes sollen daher näher diskutiert werden. Es wird davon ausgegangen, dass die Nachrichtensendung über einen gewöhnlichen analogen TV-Sender empfangen wird. Um diese Sendung pausieren zu können, wird ein Aufnahmegerät benötigt. Ein HD-Rekorder könnte diese Aufgabe übernehmen. Um die Nachrichtensendung während der Fahrt mit dem Autoradio wiedergeben zu können, muss sie entweder beendet sein und die Aufzeichnung könnte vor Fahrtbeginn auf das Autoradio überspielt werden, oder der HD-Rekorder verfügt über eine Streaming-Komponente. Dann müsste das Auto jedoch während der Fahrt unterbrechungsfrei mit dem HD-Rekorder verbunden bleiben. Da in der ursprünglichen Szenariobeschreibung kein Hinweis gefunden werden konnte, welche Variante zu bevorzugen ist, wurde sich für die letztere entschieden. Das Szenario wurde wie folgt adaptiert:
Bart hat gerade den Anruf erhalten und ist in sein Auto eingestiegen, um zum Flughafen zu fahren. Der Fernseher in seinem Wohnzimmer zeigt weiterhin die Nachrichtensendung. Da das Autoradio lediglich Audiodaten abspielen kann, muss das Videosignal des Fernsehers in ein Audiosignal umgewandelt werden. Glücklicherweise ist sein PDA dazu in der Lage.

Operatoren:

```
(startStream ?dev - Device ?s - Stream ?f - Format) ;Einen Stream starten
(stopStream ?dev - Device ?s - Stream ?f - Format) ;Einen Stream stoppen
(accessStream ?src ?dest - Device ?s - Stream ?f - Format) ;Ein Dienst
;der auf einen gerade laufenden Stream auf einem Geräte zugreifen
(connect ?dev1 ?dev2 - Device) ;Dieser Dienst stellt eine Netzwerkverbindung
;zwischen zwei Geräten her.
(convertData (?d - Data ?fFrom ?fTo - Format ?dev - Device) ;Ein Dienst,
;der verschiedene Datenformate ineinander konvertiert.
```

Ziel:

```
(and (isActive CarRadio RecordedNews Audio))
```

Initialer Weltzustand:

```
(supportsFormat Audio PDA)
(supportsFormat Video PDA)
(at Home TV)
(supportsFormat Video TV)
(isAvailable RecordedNews TV)
(hasFormat Video RecordedNews)
(isActive TV RecordedNews Video)
(isLocked TV)
(supportsFormat Audio CarRadio)
```

Plan:

```
0 (CONNECT PDA CARRADIO)
0 (STOPSTREAM TV RECORDEDNEWS VIDEO)
1 (CONNECT TV PDA)
2 (STARTSTREAM TV RECORDEDNEWS VIDEO)
3 (ACCESSSTREAM TV PDA RECORDEDNEWS VIDEO)
4 (CONVERTDATA RECORDEDNEWS VIDEO AUDIO PDA)
5 (ACCESSSTREAM PDA CARRADIO RECORDEDNEWS AUDIO)
```

Bemerkungen: Dieses Szenario verdeutlicht mehrere Modellierungsaspekte für intelligente Umgebungen. Es verwendet Typen und Typenvererbung. Prädikate wie (isLocked ?o) oder (isMovable ?o) sind nur in einer Typenhierarchie sinnvoll, da sie für einen gemeinsamen Obertyp (in diesem Fall Object) definiert sind, in den Operatoren aber mit abgeleiteten Typen (z.B. Device) verwendet werden. Ohne Typenvererbung müssten einzelne Prädikate für jeden Objekttyp (z.B. (isMovablePerson ?p)) als auch die entsprechenden Operatoren einzeln definert sein, was die Domänenbeschreibung unübersichtlich und durch die größere Anzahl an Operatoren langsamer machen würde.

Im beschriebenen Szenario wird ein Dienst beschrieben, der Daten konvertiert (convertData). Da PDDL keine Möglichkeit anbietet den Typ eines Objektes zur Planungszeit zu ändern, können spezielle Typen, wie AudioStream oder VideoStream nicht benutzt werden. Zu konvertierende Daten müssen über Prädikate beschrieben werden. Da theoretisch alle Daten konvertierbar sind, bietet sich für dynamische Ad-hoc-Umgebungen daher ein gemeinsamer Typ „Data“ oder „ConvertableData“ an. Weitere Verfeinerungen müssen dann per Typprädikat (z.B. (isAudioStream ?d)) beschrieben werden, wodurch sich allerdings die oben beschriebenen Nachteile ergeben.

Im Szenario enthält der Operator (startStream) drei Parameter, das Device, den Stream und das Format des Streams. Ein Stream kann nur ein Format haben, daher ist der dritte Parameter redundant. Das Format kann über den Stream inferiert werden. PDDL stellt dazu abgeleitete Prädikate zur Verfügung. Der Wahrheitswert eines abgeleiteten Prädikates wird über Axiome aus bekannten Prädikaten abgeleitet. Eine genauere Betrachtung von abgeleiteten Prädikaten wird in Abschnitt 4.2.6 gegeben.

4.1.5 Weitere Szenarien aus der Literatur

Das eingangs beschriebene Szenario aus dem Projekt UBbiREAL (Nishikawa u. a., 2006) verwendet lediglich Stimulus-response-Aktionen. Ähnliches lässt sich bei Brumitt u. a. (2000) beobachten. Der Fokus solcher Szenarien liegt nicht auf der dynamischen Kooperation von Geräten, sondern vielmehr auf der möglichst nahtlosen Reaktion auf den Nutzer entsprechend festgelegter Regeln mit meist festverdrahteten Geräten. Die dort beschriebenen Dienste beziehen sich nicht aufeinander, alle Geräte können gleichzeitig agieren. Daher sind keine Aktionssequenzen nötig, um den Nutzer zu unterstützen. Eine dynamische und automatische Gerätekooperation wird nicht verlangt.

In den Veröffentlichungen zu intelligenten Umgebungen finden sich noch weitere Szenarien. Es wurde auf die Ausmodellierung dieser Szenarien verzichtet, da sich die beschriebenen Probleme auf die genannten Beispiele abbilden lassen und so keine neuen Modellierungsaspekte zu erwarten sind.

Im Projekt GAIA (Roman u. a., 2003) wird eine „ticker tape“ Anwendung beschrieben, die Informationen auf vorhandenen Displays anzeigen kann. Zusätzlich gibt es eine Ortungsanwendung, die bestimmen kann, wo sich eine Person befindet. *Ein Nutzer „Andrew“ betritt den Raum 2401, daraufhin erscheint auf den vorhandenen Displays eine Laufschrift, auf der zu lesen ist „Andrew has entered room 2401“. Weiterhin ist ein Bild von Andrew auf den Monitoren zu sehen.* Da in der Veröffentlichung die Bezeichnung „active space“ verwendet wird, kann von aktiven Räumen ausgegangen werden, die das Eintreffen von Andrew registrieren und dann ihrerseits die ihnen bekannten (weil festinstallierten) Displays ansteuern. Die gewünschten Funktionalitäten bedürfen keiner Dienstekomposition, da sie direkt umsetzbar sind. Die Problematik dieses Szenarios liegt im reibungslosen Zusammenspiel der Komponenten, dem Datenzugriff und der korrekten Ortung der Person.

Das Portolano Projekt (Esler u. a., 1999) beschreibt ein Szenario, welches heute schon weitestgehend der Realität entspricht, und ohne Dienstekomposition auskommt. *Alice sieht eine Werbung einer interessanten 3D-Kamera in ihrer digitalen Zeitung und leitet sie an ihren Freund Bob weiter. Dem gefällt das Gerät und er lässt durch einen Shopping-Agenten den günstigsten Preis ermitteln, um die Kamera zu kaufen. Mit der neuen Kamera macht er einige Bilder der Antiquitätensamlung seines Nachbarn und lässt die Bilder durch einen automatischen Dienst als Webgalerie veröffentlichen.*

Im Szenario GeoTagger des Projektes PalCom (Svensson u. a., 2007) werden Fotos automatisch mit den aktuellen GPS-Daten des Standorts annotiert und dann auf einem mitgeführten Laptop gespeichert. Der Fokus dieses Projektes liegt auf der Interoperabilität zwischen den einzelnen Geräten. Das Prozessmodell der Dienste, Standortbestimmung, Fotografieren und Annotieren

sowie Speichern auf dem Laptop ist statisch.

4.1.6 Modellierung der SmartLab-Szenarios

Die Beispiele, die in den letzten Abschnitten vorgestellt wurden, sind der Literatur entnommen. Das folgende Szenario wurde basierend auf den Geräten des SmartLab der Universität Rostock erstellt. Im SmartLab befinden sich mehrere Lampen, ausfahrbare Leinwände, ein intelligenter Monitorswitch und mehrere Projektoren. Neben diesen fest installierten Geräten werden im Szenario Laptops verwendet, die Dokumente anzeigen können. Vier der Projektoren sind fest und einer drehbar montiert. Ein fest montierter Projektor zeigt auf genau eine im Raum vorhandene Leinwand. Der drehbare Projektor kann auf verschiedene Leinwände gerichtet werden.
Im Folgenden soll iterativ die Modellierung der Domäne nachvollzogen werden, was es erlaubt die einzelnen Modellierungsentscheidungen, die dabei zu treffen sind, nachzuvollziehen. Im Unterschied zu den obigen Szenarien werden dabei vollständige Operatorbeschreibungen verwendet.
Ziel eines Nutzers im SmartLab könnte es sein, mit Hilfe der beschriebenen Dienste die Lampe LAMP1 auszuschalten und ein Dokument DOC1 auf der Leinwand LW1 anzuzeigen. Dazu ist es für den Nutzer nicht von Interesse, auf welchem Gerät das entsprechende Dokument derzeit verfügbar ist, noch soll der Nutzer damit behelligt werden, wie Projektoren und Leinwände gesteuert werden. Initial ist das Dokument DOC1 auf dem Notebook NB1 verfügbar, die Leinwände LW1 und LW2 sind nicht heruntergefahren und der Projektor NEC zeigt auf die Leinwand LW2. In PDDL ausgedrückt sehen initialer Weltzustand und Ziel folgendermaßen aus:

```
(:init (isLightOn LAMP1)
       (isAvailable DOC1 NB1)
       (isMovable NEC)
       (isPointingTo NEC LW2))
(:goal (and (isActive DOC1 LW1)(not (isLightOn LAMP1))))
```

Die aus Sicht der Modellierung einfachsten Geräte des Szenarios sind Lampen und Leinwände. Jede Lampe kann zwei mögliche Aktionen, das Ein- und das Ausschalten, ausführen. Ob die entsprechende Lampe eingeschaltet ist beschreibt das einzige dafür benötigte Prädikat (isLightOn ?l − Lamp). Die Vorbedingung zum Einschalten einer Lampe ist, dass die Lampe ausgeschaltet ist. Und die entgegengesetzte Vorbedingung zum Ausschalten der Lampe ist, dass sie an ist. Die entsprechenden PDDL-Operatoren sehen folgendermaßen aus (Die Operatoren zum Herauf- und Herunterfahren der Leinwände sind äquivalent):

```
(:action LightOn
  :parameters (?l - lamp)
```

```
    :precondition (not (isLightOn ?l))
    :effect (isLightOn ?l))

  (:action LightOff
    :parameters (?l - lamp)
    :precondition (isLightOn ?l)
    :effect (not (isLightOn ?l)))
```

Diese Operatoren haben zunächst keine weiteren Vorbedingungen, die für das aktuelle Szenario von Belang sind. Es ist jedoch denkbar zu fordern, dass das Gerät eingeschaltet oder bereit ist, um einen Dienst dieses Gerätes zu verwenden, was zum Beispiel durch ein entsprechendes Prädikat (isOn ?d − device) ausgedrückt werden könnte. Jedoch unterscheidet diese Forderung eine Lampe nicht von anderen einschaltbaren Geräten und es sollten in einem Operator nur für den repräsentierten Dienst wesentliche Vorbedingungen und Effekte beschrieben werden. Eine Möglichkeit vererbbare PEs zu formulieren stellen abstrakte Operatoren dar, worauf im Abschnitt 8.3.1 näher eingegangen wird.
Im Rahmen dieses Szenarios werden Dienste als gerätefreie PDDL-Operatoren modelliert. Es wird zunächst erläutert, was unter gerätefreier Modellierung verstanden wird. Jeder atomare Dienst einer intelligenten Umgebung wird von einem bestimmten Gerät angeboten. Dabei kann ein Gerät mehrere Dienste anbieten, ein Dienst ist aber genau einem und nur einem Gerät zugeordnet. Der Rückschluss auf welchem Gerät ein Dienst ausgeführt wird, ist daher immer implizit möglich. In Dienstbeschreibungssprachen wird diese Zuordnung durch das sogenannte Grounding der Beschreibung erreicht (Kopecký u. a., 2006), wobei einem Dienst üblicherweise eine URI zugeordnet wird.
In Planungsdomänen werden Operatoren demgegenüber objektfrei modelliert. Wird ein Dienst als Planungsoperator modelliert, erhält er einen zusätzlichen Parameter für das Gerät, auf dem der Dienst läuft. Ein Vorteil von gerätefreier Modellierung ist die Möglichkeit, gleiche Pläne für unterschiedliche Umgebungen wiederzuverwenden. Dafür ist vor der Ausführung eines Planes ein zusätzlicher Groundingschritt notwendig. Allerdings ist dieser für die Generierung der Aktionssequenz aus einem Prozessmodell ohnehin vorgesehen.
Um das Ziel (isActive DOC1 LW1) zu erreichen, ist eine Kette von mindestens drei Aktionen nötig. Das Dokument muss zunächst auf einem Laptop sichtbar sein bzw. auf dessen Videoausgang gelegt worden sein. Hierfür ist die Aktion ShowDocument verantwortlich. Dieser Laptop muss dann mit Hilfe der Aktion SwitchNotebookOnProjector mit einem Projektor verbunden werden, der auf die Leinwand LW1 zeigt. Schließlich muss der Projektor das Signal auf der Leinwand darstellen, wofür der Dienst ProjectToSurface verantwortlich ist.
Der Dienst ShowDocument zum Anzeigen eines Dokumentes auf einem Notebook wurde wie folgt modelliert.

```
  (:action ShowDocument
    :parameters (?d - document ?n - notebook)
```

```
:precondition (and (isAvailable ?d ?n))
:effect (and (isActive ?d ?n)))
```

In der Vorbedingung von showDocument wird gefordert, dass das entsprechende Dokument auf dem Notebook vorhanden ist. Als Effekt ist das Dokument auf dem Gerät „aktiv“. Das Prädikat isActive kann dabei je nach Gerät, auf dem es verwendet wird, eine unterschiedliche Semantik haben. Auf einem MP3-Player würde es beispielsweise anzeigen, dass die entsprechende Datei gerade abgespielt wird.
Bei der hier verwendeten Modellierung von ShowDocument wird jedoch nur der Videoausgang des Notebooks beschrieben. Dessen Monitor ist nicht modelliert, da er für das aktuelle Problem nicht relevant ist. Allgemein können mit Hilfe des Prädikates isActive ?d ?n lediglich Geräte mit einem Ausgang beschrieben werden. Sollen mehrere Ausgänge beschrieben werden, müssten die Ausgänge als Objekte vorhanden sein und isActive anstelle des ersten Arguments vom Typ device beispielweise ein Objekt vom Typ output beinhalten.
Der Operator SwitchNotebookOnProjector repräsentiert den Dienst des Monitorswitches, der einen bestimmten Eingang, an dem ein Notebook angeschlossen ist, mit einem Ausgang, an dem ein Projektor angeschlossen ist, verbindet.

```
(:action SwitchNotebookOnProjector
  :parameters (?e - extron ?n - notebook ?p - projector)
  :precondition ((isActive ?d ?n))
  :effect (and (isConnected ?n ?p)(isActive ?d ?p)))
```

Hierbei wird ein weiterer wesentlicher Aspekt der verwendeten Modellierung sichtbar. Eine Forderung an die Repräsentation von Diensten als Operatoren ist die Beschränkung auf Wissen, welches im direkten Zusammenhang mit dem Gerät bzw. Dienst steht (vgl. Abschnitt 3.2). Im obigen Operator wird in der Vorbedingung verlangt, dass ein bestimmtes Dokument auf einem Notebook aktiv ist (isActive ?d ?n), um dieses Dokument im Effekt auf einem Projektor als aktiv zu kennzeichnen (isActive ?d ?p). Die Funktion eines Monitorswitches beschränkt sich lediglich auf die Verbindung zweier Geräte, sodass allein der Effekt isConnected ?n ?p für die Beschreibung des Operators als ausreichend erscheint. Daher ist es zunächst nicht ersichtlich, warum es für den Dienst des Monitorswitches von Bedeutung ist, welches Dokument oder verallgemeinert welche Daten auf den jeweiligen Ein- und Ausgängen aktiv sind. Da Planung immer nur auf dem aktuellen Zustand von Σ beruht und nicht auf Wissen der Vergangenheit zugreifen kann, müssen alle wichtigen Informationen von Zustand zu Zustand weitergegeben werden. Zu diesen wesentlichen Informationen gehört auch das Wissen um das aktuell angezeigte Dokument. Denn die Präsentation der Daten auf einem Display oder deren Bereitstellung an einem Ausgang ist ein wesentliches zu modellierendes Merkmal des entsprechenden Dienstes.

Der Operator ProjectToSurface veranlasst einen Projektor dazu, das am Eingang anliegende Signal auf die Leinwand zu projizieren, auf die er aktuell gerichtet ist.

```
(:action ProjectToSurface
  :parameters (?p - projector ?s - surface ?d - data)
  :precondition (and (isDown ?s)(isPointingTo ?s ?p)(isActive ?d ?p))
  :effect (and (isActive ?d ?s)))
```

Dazu muss die entsprechende Leinwand heruntergelassen sein, der Projektor muss auf die entsprechende Leinwand zeigen und das Dokument, was auf der Leinwand angezeigt werden soll, muss auf dem Projektor aktiv sein. Mit dem Operator MoveProjector kann ein drehbar montierter Projektor auf verschiedene Leinwände gerichtet werden.

```
(:action MoveProjectorToSurface
  :parameters (?p - projector ?sfrom ?sto - surface)
  :precondition (and (isMovable ?p)(isPointingTo ?sfrom ?p)
                                                (isAdjustedTo ?sto ?p))
  :effect (and (isPointingTo ?sto ?p)(not (isPointingTo ?sfrom ?p))))
```

Dazu muss der Projektor schwenkbar (isMovable ?p) und auf die aktuelle Leinwand eingestellt sein (isAdjustedTo ?sto ?p).[1] Das Einstellen des Projektors übernimmt der Operator AdjustProjectorToSurface.

```
(:action AdjustProjectorToSurface
  :parameters (?p - projector ?sfrom ?sto - surface)
  :precondition (and (isAdjustedTo ?sfrom ?p))
  :effect (and (isAdjustedTo ?sto ?p)(not (isAdjustedTo ?sfrom ?p))))
```

Werden die beschriebenen Operatoren zu einer Domäne zusammengesetzt, lässt sich das eingangs beschriebene Planungsproblem lösen. Im Anhang A.4 findet sich die gesamte Domänen- und Problembeschreibung.

4.1.7 Zusammenfassung

Anhand der Modellierung der einzelnen in diesem Kapitel vorgestellten Szenarien, wird ersichtlich, dass die unterschiedlichen Sichtweisen auf die Dienste bzw. Operatoren seitens der Dienstebeschreibungen und der klassischen KI-Planung wesentliche Anforderungen an die Modellierung stellen.
Es ist zu erkennen, dass allein durch eine gemeinsame Syntax und Semantik, eine verteilte Modellierung von Diensten als Planungsoperatoren nicht zu gewährleisten ist. Zusätzlich sind Richtlinien erforderlich, an denen sich bei der Modellierung orientiert werden kann. Der nächste Abschnitt 4.2 widmet sich diesem Thema.

[1] Diese Einstellungen betreffen insbesondere die Trapezeinstellungen des Projektors, die sich durch die unterschiedlichen Winkel, mit denen der Projektor auf die verschiedenen Leinwände projiziert, unterscheiden.

Abschließend soll erwähnt werden, dass alle betrachteten Beispielszenarien mit sehr einfachen Mitteln modelliert werden konnten. In den PEs der Operatoren wurden lediglich (**and**) und (**not**) verwendet, auf quantifizierende Elemente wie (forall) und (exist) konnte verzichtet werden. Dies ist als Vorteil zu betrachten, da Versuche mit verschiedenen Planern drastisch erhöhte Laufzeiten bei der Verwendung quantifizierender Elemente zeigten.

4.2 Modellierungsrichtlinie

Um eine planungsbasierte Komposition zu erlauben, müssen die Dienste einer intelligenten Umgebung durch ihre Vorbedingungen und Effekte beschrieben sein. Im letzten Abschnitt wurde deutlich, dass gleiche Syntax und Semantik nicht ausreichend sind, um reibungslose Interoperabilität bezüglich der Komposition zu erreichen. In diesem Abschnitt werden Konventionen in Form einer Modellierungsrichtlinie beschrieben, die helfen, eine verteilte Modellierung von Dienstebeschreibungen zu ermöglichen.
In der Literatur findet sich aus Sicht der Dienstekomposition keine Beschreibung von Modellierungsanforderungen, auch eine Sammlung von Referenzfällen existiert nicht. Es finden sich allerdings Veröffentlichungen zur Modellierung mit Beschreibungslogiken (Borgida und Brachman, 2003), worin allgemeine Richtlinien zur Formulierung von Prädikaten beschrieben sind, die auch bei der Erstellung der Szenarien in dieser Arbeit berücksichtigt wurden. Allerdings sind diese Modellierungsrichtlinien nicht spezifisch genug, um den Besonderheiten der Dienstekomposition basierend auf KI-Planung gerecht zu werden.
Auch innerhalb der Planungscommunity ist die einheitliche Modellierung von Domänen ein offenes Problem. Die Benchmarkprobleme der IPC sind speziell für den jeweiligen Anwendungsfall erstellt. Allgemeine Modellierungsrichtlinen konnten nicht gefunden werden. Im Rahmen der ICAPS gibt es seit 2005 die International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS), die wie auch die IPC zweijährlich veranstaltet wird. Ziel der ICKEPS ist es, Methoden der Domänenmodellierung für Planungsprobleme miteinander zu vergleichen und zur Entwicklung von nutzerfreundlichen Modellierungssystemen für Planungsdomänen und -probleme zu animieren. Die Forschung ist hier jedoch noch in einem frühen Stadium. Ergebnisse, die im Rahmen dieser Arbeit zu verwenden wären, konnten nicht gefunden werden.

4.2.1 Nur Dienste als Operatoren

Als Ergebnis des Planungsprozesses wird eine Sequenz von Aktionsausführungen erwartet. Daher dürfen nur Operatoren verwendet werden, die einen real vorhandenen Dienst repräsentieren.
Stellt man sich die Entstehung einer Operatorbeschreibung vor, erscheint diese Anforderung zunächst überflüssig, da es implizit ist, dass ein Diensteanbieter lediglich die Funktionalität seines Dienstes beschreibt. Allerdings ist es denkbar, dass Modellierer eine erweiterte Vorstellung des Begriffes Operator entwickeln. So wird im Beispiel 4.1.3 menschliches Handeln im Operator move

modelliert. Werden solche Operatoren verwendet, kann eine Planung anstelle einer automatisch ausführbaren Sequenz von Aktionen nur noch Vorschläge zum Handeln liefern. Es kann dann nicht mehr von einer reinen Dienstekomposition gesprochen werden. Um im Problemfeld der automatischen Dienstekomposition zu bleiben, muss daher sichergestellt sein, dass ausschließlich direkt ausführbare Dienste der Geräte als Operatoren formuliert werden und keine menschlichen Aktionen für die Ausführung von Aktionen nötig sind.

4.2.2 Kein globales Wissen in Effekten

Im Beispiel 4.1.1 ist eine Aktion erforderlich, um die Temperatur in Alice' Zimmer anzuheben. Vorstellbar zur Beschreibung der Temperatur, ist sowohl ein Prädikat (temperatureAt ?r − Room ?t − Temperature) als auch ein Prädikat (temperatureSetting ?ac − Aircondition ?t − Temperature). Während die Semantik des ersten Prädikates die aktuelle, reale Temperatur im Raum bezeichnet, wird im zweiten Fall lediglich die Einstellung der Klimaanlage beschrieben. Es stellt sich die Frage, welche Modellierung besser geeignet ist. Aufgrund der Ad-hoc-Natur intelligenter Umgebungen kann die Modellierung der Operatoren dabei nur aus lokaler Sicht erfolgen, d.h. zum Zeitpunkt der Modellierung ist kein Wissen über den späteren Einsatzort des Gerätes vorhanden.
Aufgrund ihrer zustandsbeschreibenden Natur, sollten Prädikate immer so formuliert werden, dass sie (zumindest theoretisch) von Sensoren gemessen werden könnten. Im Beispiel scheint also das erste Prädikat angebracht, denn Temperatur ist eindeutig eine durch Sensoren messbare Größe.
Eine sich daraus ergebende Operatorbeschreibung, um die Klimaanlage auf eine bestimmte Gradzahl einzustellen, könnte wie folgt aussehen:[2]

```
(:action setTemperature
  :parameters (?r - Room ?t - Temperature)
  :precondition ()
  :effect (temperatureAt ?r ?t))
```

Dieser Operator würde bei Ausführung die Klimaanlage einstellen und im Weltzustand des Planers die Raumtemperatur sofort auf die eingestellte Gradzahl ändern. Allerdings braucht es Zeit, bis der Raum diese Temperatur tatsächlich erreicht. Diese Dauer ist von verschiedenen Faktoren, wie beispielsweise der aktuellen Temperatur im Raum, der Leistungsfähigkeit der Klimaanlage und der Außentemperatur abhängig. Diese Faktoren haben wiederum andere Abhängigkeiten. Das Problem verschärft sich noch, wenn neben stationären Geräten wie einer Heizung bzw. Klimaanlage auch mobile Geräte einbezogen werden, die zusätzlich die Raumtemperatur beeinflussen. Wie schon eingangs

[2]Es wird angenommen, dass diese Klimaanlage sowohl kühlen als auch heizen kann.

erwähnt, ist das Wissen um die verschiedenen Einflüsse zur Modellierungszeit des Operators aufgrund der Ad-hoc-Natur der Umgebung nicht vorhanden. Unter diesen Umständen kann das Prädikat (temperatureAt ?r ?t) nicht im Effekt des Operators verwendet werden. Dennoch ist es nachvollziehbar, dass die direkte Angabe globaler Effekte, also beispielsweise einer Temperatur, als Ziel einer Komposition wünschenswert ist.
Eine Möglichkeit, globale Effekte einer Aktion berücksichtigen zu können, ist die Verwendung von konditionaler Planung (Peot und Smith, 1992; Kuter u. a., 2004). Dabei enthalten die Pläne nicht nur auszuführende Aktionen, sondern auch Entscheidungspunkte, an denen während der Ausführung des Plans, ausgehend von einer Überprüfung der realen Umgebung, zwischen mehreren Teilplänen ausgewählt werden kann. Nur so kann der reale Einfluss eines Operators in einer Ad-hoc-Umgebung bestimmt werden. Die Operatoren müssen jedoch entsprechend modelliert sein, und die Möglichkeiten der Ausführung von konditionalen Plänen müssen gegeben sein.
Ist die Verwendung von konditionaler Planung nicht möglich, ist es daher erforderlich die Effekte eines Operators auf lokale Eigenschaften zu beschränken. Globale Zustände wie zum Beispiel die Raumtemperatur können dann nicht in den Effekten eines Operators enthalten sein. Die Fallstudien im Abschnitt 4.1 belegen, dass auch ohne die Verwendung globaler Effekte interessante Szenarien umgesetzt werden können.
Im Rückblick auf die Frage, welches der beiden Prädikate (temperatureAt ?r ?t) oder (temperatureSetting ?ac ?t) besser geeignet ist, kann festgehalten werden, dass beide ihre Berechtigung zur Modellierung einer intelligenten Umgebung haben. Das erste Prädikat beschreibt globales Wissen und kann bei der Verwendung klassischer KI-Planung nur in den Vorbedingungen von Operatoren erscheinen. Das zweite Prädikat beschreibt dagegen den Zustand eines Gerätes und damit lokales Wissen und kann daher sowohl in der Vorbedingung als auch im Effekt eines Operators verwendet werden.

4.2.3 Keine Erzeugung von Objekten zur Laufzeit

Im Abschnitt 3.2 wurde erläutert, dass Dienste und KI-Planung verschiedene Sichtweisen auf die Welt haben. Dienste agieren über den Austausch von Nachrichten, Planung modelliert die Welt als Zustandsübergangssystem, in dem die Zustände über die Eigenschaften von Prädikaten definiert werden. Aufgrund der in Abschnitt 3.1 beschriebenen Annahme 1 müssen in einem Planungsproblem alle Prädikate von Beginn an bekannt sein. PDDL erlaubt es darüberhinaus in einem Planungsproblem Objekte zu definieren, denen Prädikate zugeordnet werden können. Dabei gilt wie auch schon für Prädikate, dass alle Objekte von Beginn an bekannt sein müssen. Es ist nicht möglich,

während der Planungszeit neue Objekte zu erzeugen. Planung geht davon aus, dass die Operatoren den Zustand der Welt nur im Rahmen der bestehenden Prädikate verändern. Operatoren, die Objekte nicht nur verändern, sondern in Form von Rückgaben erzeugen, sind ausgeschlossen.
Werden nun die Nachrichten, die zwischen Diensten ausgetauscht werden, als Objekte betrachtet, entsteht ein generelles Problem (Srivastava und Koehler, 2004), da jede Antwort eines Dienstes einer neuen Nachricht und so einem neu erzeugten Objekt entspricht.
Eine Möglichkeit der Modellierung ist die Verwendung von Hilfsobjekten (dummys). Der den Dienst repräsentierende Planungsoperator erhält einen zusätzlichen Parameter für das Rückgabeobjekt (die Antwortnachricht). Dieses wird leer übergeben, und bei Ausführung der Aktion auf den entsprechenden Rückgabewert gesetzt. Das Dummyobjekt muss allerdings vor Ausführung der Planung in der Problembeschreibung vorhanden sein. Dabei ist a priori nicht bekannt, wie viele Dummies innerhalb eines Planes benötigt werden. Es sind Szenarien denkbar, in denen sehr viele Dummies erforderlich sind, die in der Problembeschreibung initialisiert worden sind, was zum einen die Größe der Problembeschreibung unter Umständen stark ansteigen lässt. Zum anderen ist eine Abschätzung der benötigten Anzahl der Dummies nicht trivial, insbesondere wenn zugelassen wird, dass ein Dienst mehrere Rückgaben hat und so mehrere Dokumente erzeugen kann. Die Komplexität dieses Problems ist der des eigentlichen Planungsproblems mindestens gleichwertig, da um die Anzahl der nötigen leeren Dokumente zu bestimmen, alle Planschritte, also der gesamte Plan, bekannt sein muss, was aber genau dem eigentlichen Planungsproblem entspricht.
Neben der Erzeugung ist auch das Löschen von Objekten zur Planungszeit nicht möglich. Um Objekte erstellbar und löschbar zu machen, kann ein entsprechendes Existenz-Prädikat eingeführt werden. Im Beispiel 4.1.2 wurde das Routen-Dokument dafür mit dem Prädikat isEmpty markiert.
Um eine einheitliche Modellierung zu gewährleisten, muss ein solches Prädikat allen Diensten in einer intelligenten Umgebung bekannt sein, die mit Objekten arbeiten, die zur Planungszeit erzeugt werden können. Da diese Menge von Objekten neben den zwischen Diensten ausgetauschten Nachrichten auch auf jegliche Art von digitalen Daten zutrifft, wäre ein allgemeiner Typ digitalData denkbar. Werden Objekte vom Typ digitalData verwendet, muss jeder Operator, der auf ein entsprechendes Objekt zugreift, dessen Existenz in der Vorbedingung mit (**not** (isEmpty ?d)) prüfen. Durch die Verwendung eines Existenz-Prädikates wäre auch ein Löschen möglich, indem das Prädikat isEmpty eines Objektes wieder auf **true** gesetzt würde.
Das Erzeugen neuer Objekte wird auch in der SETTLER-Domäne der IPC3 (Long und Fox, 2003) mit Hilfe von leeren Objekten gelöst. Hier wird eine

feste Anzahl von unbestimmten Fahrzeugen (vehicles) kreiert, die dann zur Planungszeit verschiedenen Fahrzeugtypen wie (is−car ?v) oder (is−train ?v) zugewiesen werden.
McDermott (2002) schlägt zur Lösung des beschriebenen Problems die Einführung eines :value-Feldes für die Beschreibung von PDDL-Operatoren vor. In diesem Feld könnten Rückgabewerte von Diensten beschrieben werden. Allerdings konnte sich dieser Vorschlag nicht durchsetzen. In keiner Version von PDDL sind solche Felder erwähnt.
Das von Blankenburg u. a. (2008) vorgeschlagene agentHasKnowledgeAbout-Prädikat für alle Informationen, die ein Dienst über seine Eingänge benötigt oder über seine Ausgänge produziert, ist eine Anlehnung an die Speech Act Theorie (Austin, 1975), die auch der Knowledge Query and Manipulation Language (KQML) zu Grunde liegt. Die Verwendung eines solchen Prädikates erlaubt es, alle Ausgänge eines Dienstes als Effekte aufzufassen, was zu einer Vermischung der verschiedenen Sichten auf den Dienst führt, wie sie in Abschnitt 3.2.2 beschrieben wurde, und ist daher kritisch zu sehen.
Eine weitere Möglichkeit, die Erzeugung neuer Objekte zu modellieren, ist der komplette Verzicht auf die syntaktischen Hilfen durch Objekte und Typisierung in PDDL. In diesem Fall würden alle Objekte über eigene Prädikate modelliert werden. Dadurch verlöre die Modellierung jedoch an Übersichtlichkeit und den Planern würden Optimierungsmöglichkeiten genommen, die sich aufgrund der Verwendung von (getypten) Objekten eröffnen, was wiederum zu längeren Planungszeiten führen kann.

4.2.4 Andauernde Aktionen

Zunächst soll der Ausdruck „andauernde Aktion“ vom Begriff der Aktionsdauer unterschieden werden. Die Dauer einer Aktion (duration) beschreibt die Zeit, die zwischen dem Einleiten der Aktion und dem tatsächlichen Gültigwerden der Effekte der Aktion vergeht. Ein Beispiel für eine Aktion mit verzögertem Eintreten der Effekte ist beispielsweise ein drehbarer Projektor. Dieser benötigt eine gewisse Zeit, um sich auf eine neue Position zu drehen. Erst nach dieser Drehung ist der Effekt der Aktion moveProjector wahr. PDDL unterstützt seit der Version 2.1 Formulierungen, um die Dauer von Aktionsausführungen zu beschreiben (Fox und Long, 2003).
Dahingegen beschreiben andauernde Aktionen Dienste, die zur Erreichung eines Zieles dauerhaft aktiv sein müssen und nicht unterbrochen werden dürfen. Ihnen ist bei der Modellierung besondere Beachtung zu schenken. Planungsalgorithmen sind zwar in der Lage, Abhängigkeiten zwischen zwei Aktionen zu erkennen (Weld (1994) verwendet dafür „Causal-Links“ und „Threats“ in dem von ihm beschriebenen Partial-Order-Planning Algorithmus). Dieser Mecha-

nismus versagt jedoch bei andauernden Aktionen mit verketteten Abhängigkeiten (Plociennik u. a., 2009). Verkettete Abhängigkeiten treten insbesondere dann auf, wenn Daten über mehrere Stationen geleitet werden müssen. Im Beispiel des SmartLab muss das auf der Leinwand anzuzeigende Dokument ausgehend vom Laptop durch den Monitorswitch und den Projektor geleitet werden, um schließlich auf der Leinwand zu erscheinen. Dabei darf keines der vier involvierten Geräte den Datenfluss unterbrechen, da das Dokument sonst nicht mehr angezeigt werden würde.
Anhand der im Anhang A.4 enthaltenen Domäne des SmartLab soll die Problematik der andauernden Aktionen kurz erläutert werden. Ziel des Szenarios ist es, ein Dokument DOC1 auf der Leinwand LW1 anzuzeigen. Dafür muss die Proposition isActive DOC1 LW1 gelten.[3] Das Szenario lässt sich durch einen Planer ohne Weiteres lösen. Ein resultierender Plan sähe wie folgt aus:

```
0:   (CANVASDOWN LW1)
0:   (ADJUSTPROJECTORTOSURFACE NEC LW2 LW1)
0:   (SHOWDOCUMENT DOC1 NB1)
0:   (LIGHTON LAMP1)
1:   (MOVEPROJECTORTOSURFACE NEC LW2 LW1)
1:   (SWITCHNOTEBOOKONPROJECTOR EXTRON NB1 NEC DOC1)
2:   (PROJECTTOSURFACE NEC LW1 DOC1)
```

Nun soll ein zusätzliches Ziel hinzugefügt werden. Ein zweites Dokument DOC2 soll auf der zweiten Leinwand LW2 angezeigt werden. Dazu muss die Proposition isActive DOC2 LW2 dem Zielzustand hinzugefügt werden. Bei Betrachtung des Szenarios fällt auf, dass ein solches Ziel mit dem gegebenen Geräteensemble nicht zu erreichen ist, da lediglich ein Projektor NEC vorhanden ist. Ohne weitere Anpassung der Domäne wird ein Planer jedoch eine Lösung wie die folgende finden:

```
0:   (CANVASDOWN LW1)
0:   (ADJUSTPROJECTORTOSURFACE NEC LW2 LW1)
0:   (CANVASDOWN LW2)
0:   (SHOWDOCUMENT DOC2 NB1)
0:   (SHOWDOCUMENT DOC1 NB1)
0:   (LIGHTON LAMP1)
1:   (SWITCHNOTEBOOKONPROJECTOR EXTRON NB1 NEC DOC2)
1:   (SWITCHNOTEBOOKONPROJECTOR EXTRON NB1 NEC DOC1)
2:   (PROJECTTOSURFACE NEC LW2 DOC2)
3:   (MOVEPROJECTORTOSURFACE NEC LW2 LW1)
4:   (PROJECTTOSURFACE NEC LW1 DOC1)
```

Es ist zu sehen, dass der Projektor im Planschritt 2 das Dokument DOC2 auf Leinwand LW2 anzeigt. Anschließend dreht er sich in Richtung LW1 und zeigt schließlich DOC1 auf LW1 an. Dadurch ist jedoch die Darstellung von DOC2 auf LW2 nicht mehr gegeben, was der Planer aber nicht beachtet. Um ein realistisches Verhalten zu erreichen, muss der Planungsdomäne das Wissen

[3] Im Szenario ist weiterhin noch gefordert, das die Lampe LAMP1 eingeschaltet sein soll, diese Anforderung soll hier keine Betrachtung finden.

hinzugefügt werden, dass ein Projektor zur gleichen Zeit nur auf eine Leinwand projizieren kann. Eine Möglichkeit dies zu erreichen, ist die Verwendung von blockenden Prädikaten (locks), z.B. (isLocked ?d − device). Die Semantik des Prädikates ist denkbar einfach. Ist ein Gerät während des Planungsprozesses als geblockt gekennzeichnet, darf es im weiteren Verlauf der Planung von keinem weiteren Operator mehr verwendet werden. Um ein geblocktes Gerät zu entsperren muss erst ein entsprechender Dienst ausgeführt werden. Dies werden in den meisten Fällen Dienste sein, die eine Aktion umkehren (siehe Rollback-Dienste).
In Plociennik u. a. (2009) wird das Thema der andauernden Aktionen weiter untersucht. Darin werden drei Prädikate zur Modellierung andauernder Aktionen vorgestellt:

- (isLocked ?d − device) - zeigt an, dass ein Gerät gegenwärtig verwendet wird und durch den Planer nicht anderweitig genutzt werden darf.

- (isActive ?data − data ?d − device) - indiziert den Endpunkt einer verketteten Abhängigkeit. Nur an diesem Punkt können die Daten weiterverarbeitet werden.

- (isConnected ?d1 ?d2 − device) - beschreibt, dass beide Geräte in einer Kette miteinander verbunden sind. Dieses Prädikat wird benötigt, wenn es Aktionen gibt, die eine Kette wieder abbauen können.

4.2.5 Konvertierung von Datentypen

Im Beispiel 4.1.4 wird die Konvertierung von Datenströmen beschrieben. Wie im Abschnitt 3.1.4 erläutert, unterstützt PDDL die Verwendung von Objekttypen. Allerdings ist aufgrund der statischen Zustandsmenge ein echtes Typcasting wie in höheren Programmiersprachen in PDDL nicht möglich. Daher müssen konvertierbare Typen als Prädikate eines Objektes beschrieben werden. Nur so ist eine Konvertierung von Daten als Teil eines Planes möglich.
Es ist dabei wichtig, welche Abstraktionsgrenze bei dieser Konvertierung gezogen wird. Dazu ist es ratsam einen absoluten Obertyp zu definieren, Verfeinerungen dieses Typs sind dann nur noch über Prädikate realisierbar. Im Abschnitt 4.2.3 wurde der Typ digitalData vorgeschlagen, um Daten, die während eines Planungsprozesses erzeugt werden können, zu indizieren. Es ist denkbar, diesen Typ auch als Obertyp hinsichtlich der Konvertierung von Datentypen zu nutzen. Vorteilhaft wäre die dadurch verstärkte klare Trennung von realen und virtuellen Objekten. Möglicherweise ist die „Auflösung“ dieses Ansatzes zu grob, da sich ggf. weitere Unterkategorien von digitalData finden lassen,

die hinsichtlich ihrer Konvertierungsmöglichkeiten disjunkt sind. Bis auf Weiteres wird dennoch digitalData als gemeinsamer Obertyp für konvertierbare Daten verwendet.

4.2.6 Abgeleitete Prädikate

Abgeleitete Prädikate, auch *Axiome* genannt, sind eine durch PDDL angebotene Möglichkeit innerhalb einer Domainenbeschreibung einfache Inferenzen zuzulassen. Ohne abgeleitete Prädikate ist PDDL eine reine Auszeichnungssprache, die zwar Lisp-Syntax verwendet, jedoch keinen Interpreter, der die Ausdrücke in den Vorbedingungen und Effekte auswerten könnte.
Die Verwendung abgeleiteter Prädikate kann vor allem die Beschreibung der Operatoren übersichtlicher gestalten und redundantes Wissen kann vermieden werden. Ohne sie müssen alle Prädikate, die bei der Beschreibung der Vorbedingungen und Effekte eines Operators verwendet werden, in der Parameterliste vorhanden sein. So wird im Abschnitt 4.1.4 beschrieben, wie der Datentyp eines Streams über ein abgeleitetes Prädikat inferiert werden kann. Die Definition eines entsprechenden abgeleiteten Prädikates in PDDL sowie eines Operators, der dieses abgeleitete Prädikat verwendet, sind im Anschluss dargestellt. Die Definition eines abgeleiteten Prädikates wird durch das Element (:derived) gekennzeichnet.

```
(:derived (supportsStream ?s - stream ?dev - device)
  (exists (?f - format)
      (and (hasFormat ?f ?s)(supportsFormat ?f ?dev))))

(:action accessStream
  :parameters (?scrDev ?destDev - Device ?s - Stream)
  :precondition (and (isActive ?scrDev ?s ?f)
                     (isConnected ?scrDev ?destDev)
                     (not (isLocked ?destDev))
                     (hasFormat ?f ?s)
                     (supportsStream ?s ?destDev))
  :effect (and (isLocked ?destDev)
               (isActive ?destDev ?s ?f)))
```

Obwohl sie in der Definition von PDDL 2.1 zwischenzeitlich nicht enthalten waren (Fox und Long, 2003), sind abgeleitete Prädikate ein mächtiges und sinnvolles Mittel, den Umfang einer Domäne sowie den Planungsaufwand zu reduzieren (Thiébaux u. a., 2005). In der Definition von PDDL 2.2 wurden sie daher wieder eingeführt (Edelkamp und Hoffmann, 2004). Dennoch werden sie von vielen Planern nicht unterstützt. Zwar ist es möglich, Domänen mit abgeleiteten Prädikaten in Domänen, die nur noch gewöhnliche Prädikate enthalten, umzuformen. Allerdings resultiert diese Umwandlung in einem super-polynomiellen Wachstum der Domäne (Thiébaux u. a., 2005).
Abgeleitete Prädikate können als Teil der Dienstebeschreibung in die Umgebung gelangen und müssen dann mit den Geräten und deren Diensten bereit-

gestellt werden. Wie oben beschrieben, können Axiome als einfache Inferenzen betrachtet werden. Beschreibungssprachen wie OWL-S unterstützen die ontologische Beschreibung von solchen Inferenzen. Es bleibt zu prüfen, inwieweit diese zusätzliche Anforderung an die Dienstebeschreibung von aktuellen Beschreibungssprachen unterstützt wird und wie eine konkrete Abbildung erfolgen könnte.

4.2.7 Ontologien

Wesentliche Leistung der Semantic-Web-Community ist die Bereitstellung von umfassenden Techniken zur Erstellung und Verwendung von Ontologien, um eine interoperable Verwendung von Diensten im Internet zu gewährleisten. Auch in lokalen Ad-hoc-Umgebungen ist die Verwendung von Ontologien unverzichtbar, sobald Gerätebeschreibungen aus unterschiedlichen Quellen verwendet werden.

Im hier betrachteten Fall der Dienstekomposition in intelligenten Umgebungen liegt die Semantik der Dienste in den Operatorbeschreibungen, konkret in den verwendeten Prädikaten und Variablentypen. Die Verwendung von PDDL als Beschreibungssprache schränkt die möglichen nutzbaren Ontologiekonzepte dabei auf eine Typentaxonomie und einfache Inferenzen zwischen Prädikaten ein. Die Typentaxonomie wird in der Domänenbeschreibung hinterlegt. Einfache Inferenzen zwischen Prädikaten können durch die von PDDL unterstützten abgeleiteten Prädikate ausgedrückt werden (vgl. Abschnitt 4.2.6).

Eine mögliche Ontologie für pervasive Umgebungungen wird in Chen u. a. (2004) unter dem Namen Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) vorgestellt. Sie basiert auf OWL und ist modular aufgebaut. Eine direkte Anwendung von SOUPA ist jedoch nicht möglich, da in ihr lediglich sehr abstrakte Konzepte wie zum Beispiel Person, Agent, Ereignis oder Zeit definiert sind. Spezifizierungen sind für den jeweiligen Anwendungsfall nötig.

Durch SOUPA werden zunächst nur Objekttypen beschreibbar gemacht. Die Beschreibung von bestimmten Eigenschaften also einer Definition der Prädikate ist nicht enthalten. Diese Definitionen werden teilweise von DSD geleistet (siehe Abschnitt 3.3.4). Darin werden verschiedene Ontologien zur Beschreibung sogenannter Zustandsklassen für Dienste eingeführt (Klein u. a., 2005). Dazu zählt beispielsweise die Ontologie category . possesssion , in der Eigenschaften wie Owned, Lent oder Rented definiert werden. Diese Eigenschaften können direkt auf die Prädikate einer Planungsdomäne angewendet werden. Eine konkrete Formulierung einer Ontologie für intelligente Umgebungen findet sich weiterhin bei Heider (2009). Darin sind jedoch lediglich problemspezi-

fische Konzepte einer Umgebung als DAML+OIL[4] Beschreibungen hinterlegt. Es werden globale Eigenschaften einer intelligenten Umgebung in Konzepten wie ambientNoise oder ambientTemperature formuliert. Jedoch lassen sich auch hier wie schon bei den Ontologien von DSD nicht alle der in Abschnitt 4.2 verwendeten Prädikate finden.
Wenngleich Erweiterungen und Anpassungen nötig sind, erscheint dennoch eine Verwendung sowohl von SOUPA als auch der Ontologien von DSD sowie der Ontologie von Heider (2009) für die KI-planungsbasierte Dienstekomposition in intelligenten Umgebungen als möglich.

4.2.8 Rollback-Dienste

Aufgrund der dynamischen Natur intelligenter Umgebungen kann es vorkommen, dass Ausführungen einer Komposition abgebrochen und neu geplant werden müssen. In einem solchen Fall können bestimmte Dienste schon ausgeführt worden sein (zum Beispiel wurden schon Blenden heruntergelassen, um einen Raum zu verdunkeln). Um eine Neuplanung zu ermöglichen, muss die Ausführung der Dienste soweit wie möglich rückgängig gemacht werden. Dazu ist die Umsetzung von Rollback-Diensten erforderlich (um im Beispiel die Blenden wieder heraufzufahren). Ein Rollbak-Dienst wäre auch dafür verantwortlich blockende Prädikate wieder frei zu geben (vgl. Abschnitt 4.2.4).
Es ist offensichtlich, dass nicht jede Dienstausführung rückgängig gemacht werden kann. Falls es jedoch möglich ist, sollte ein Rollback-Dienst vorhanden sein.

4.2.9 Modellierungsrichtlinie

Sollen Operatoren zur Dienstekomposition verwendet werden, die von verschiedenen Entwicklern unabhängig voneinander erstellt wurden, helfen die folgenden Richtlinien, eine interoperable Modellierung zu gewährleisten.

1. Ein Operator darf lediglich die Funktionalität genau eines Dienstes beschreiben (siehe Abschnitt 4.2.1).

2. Die Effekte von Operatoren dürfen kein globales Wissen beeinflussen (siehe Abschnitt 4.2.2).

3. Die direkte Erzeugung von Objekten zur Planungszeit ist nicht möglich, kann aber mit Einschränkungen durch Zuhilfenahme von Dummyobjekten modelliert werden (siehe Abschnitt 4.2.3).

[4]DAML+OIL ist direkter Vorgänger von OWL und wird heute zugunsten von OWL nicht mehr weiterentwickelt.

4. Die Verwendung von andauernden Aktionen mit verketteten Abhängigkeiten erfordert blockende Prädikate in den Operatorbeschreibungen (siehe Abschnitt 4.2.4).

5. Konvertierbare Typen müssen als Prädikate eines Objektes beschrieben werden (siehe Abschnitt 4.2.5).

6. Die Verwendung von abgeleiteten Prädikaten ist hilfreich, um Operatorbeschreibungen zu vereinfachen (siehe Abschnitt 4.2.6).

7. Die verwendeten Konzepte (Objekte und Prädikate) müssen in einer gemeinsamen Ontologie beschrieben sein (siehe Abschnitt 4.2.7).

8. Für alle reversiblen Dienste muss ein inverser Dienst (Rollback-Dienst) zur Verfügung stehen, mit dem sich ein Vorgängerzustand wieder herstellen lässt (siehe Abschnitt 4.2.8).

These 6 *Anforderungen klassischer Planer und der Realität in intelligenten Umgebungen sind nicht deckungsgleich und erfordern entsprechende Anpassungen und Richtlinien für die verteilte Modellierung.*

Kapitel 5

Evaluierung von Planern

5.1 Laufzeitverhalten

In diesem Kapitel wird das Laufzeitverhalten von Planern als wesentliches Qualitätsmerkmal der Dienstekomposition untersucht. Im Abschnitt 2.3.1 wurde erläutert, dass der Kompositionsprozess aus vier Phasen besteht, wobei im Rahmen dieser Arbeit insbesondere die Generierung des Prozessmodells betrachtet wird. Dabei teilt sich die Erstellung des Prozessmodells wiederum in verschiedene Phasen auf (siehe Abschnitt 6.3), unter denen die eigentliche Planung die höchste Komplexität und somit potentiell den größten Einfluss auf die Gesamtlaufzeit hat. Da klassische KI-Planung als Kompositionsmethode eingesetzt wird, ist es sinnvoll, das Laufzeitverhalten von klassischen Planern zu untersuchen. Kurze Planungszeiten sind aus den folgenden zwei Gründen anzustreben:

Unaufdringlichkeit Die Reaktion auf den Nutzer soll schnellstmöglich, bestenfalls vom Nutzer unbemerkt, erfolgen. Dazu ist es nötig, die Wartezeit eines Nutzers unterhalb einer Schwelle zu halten, ab der ihm bewusst wird, dass er wartet. Nach Nielsen (1994) können drei Schwellen (0,1 s; 1 s; 10 s) bei der subjektiven Bewertung von Antwortzeiten unterschieden werden. Alle Reaktionszeiten unter 0,1 Sekunden werden als augenblicklich wahrgenommen. Dauert die Reaktion einer Applikation länger, wird es vom Nutzer als Wartezeit realisiert. Dauert etwas länger als 1 Sekunde, wird der Nutzer in seinem Gedankenfluss unterbrochen. Nach einer Dauer von mehr als 10 Sekunden verliert der Nutzer die Aufmerksamkeit. Sollte die Reaktion einer Applikation noch länger dauern, sind Rückmeldungen an den Nutzer erforderlich, da ansonsten die Gefahr besteht, dass von einer Fehlfunktion ausgegangen wird. Um das Ziel der Unaufdringlichkeit zu erreichen, sollte die Reaktionszeit einer Kompositionskomponente unterhalb einer Sekunde bleiben. Hierbei wird lediglich die Erstellung des Prozessmodells betrachtet. Die Planausführung kann im Anschluss sehr viel mehr Zeit in Anspruch nehmen. Wichtig ist, dass der Nutzer bemerkt, das die Umgebung möglichst unverzüglich reagiert und möglichste minimale Wartezeiten auftreten.

These 7 *Reaktionsschnelligkeit ist primäres Qualitätskriterium intelligenter Umgebungen. Dabei ist der Verlust einer Lösung weniger kritisch als das Überschreiten einer Zeitgrenze.*

Offline Planung Die Einschränkung 8 (vgl. Abschnitt 3.1.2) besagt, dass während eines Planungslaufes Änderungen des Planungsproblemes nicht be-

rücksichtigt werden können. Änderungen des Planungsproblems ergeben sich bei einer Änderung des Weltzustandes, einer Änderung des Ziels oder einer Änderung der verfügbaren Dienste[1]. Unter der Annahme, dass diese Ereignisse zufällig auftreten, folgt, dass bei steigender Planungsdauer, die Wahrscheinlichkeit einer Zustandsänderung in der Umgebung größer wird. Eine solche Zustandsänderung während der Planung kann dazu führen, dass der resultierende Plan nicht mehr anwendbar ist und damit eine Neuplanung nötig wird. Es ist daher vorteilhaft, die Planungszeit so gering wie möglich zu halten.

Es sind keine statistischen Werte verfügbar, die eine Orientierung geben, mit wie vielen Umgebungsänderungen in einer intelligenten Umgebung pro Zeiteinheit zu rechnen ist. Wird dieses Wissen in Zukunft verfügbar, kann mit einer Aussage über eine maximale Laufzeit einer Komposition deren Anwendbarkeit in einer entsprechenden Umgebung bewertet werden.
Ein guter Ansatzpunkt, um einen Überblick über die Leistungsfähigkeit verschiedener Algorithmen innerhalb eines Anwendungsgebietes zu bekommen, sind sogenannte Challenges. Auf dem Gebiet der Webservices gibt es die *WS-Challenge*[2], die sich auch mit der Komposition von Webservices beschäftigt hat. Für den 2007er Wettkampf wurde ein Kompositionsproblem gestellt, bei dem ein Anwender auf einen Dienst zugreifen möchte, der, gegeben einen Namen, eine Adresse zurückgibt. Dieser Dienst existiert allerdings nicht atomar. Es gibt aber einen Dienst, der zu einem Namen eine ID zurückgibt und einen weiteren Dienst, der mit einer ID als Eingabe Adressen suchen und zurückgeben kann. Die Herausforderung für eine Kompositionsmethode ist es also lediglich, zwei vorhandene Dienste miteinander zu verknüpfen. Die Challenge zielt mehr auf das korrekte und effiziente Matchmaking von Webservices als auf die Qualität und Leistungsfähigkeit der Kompositionsmethode ab. Die Resultate der WS-Challenge sind daher für die Bewertung der Schnelligkeit von Kompositionsmethoden ungeeignet.
Zur Bewertung der Effektivität von Planern geben die Resultate der IPC (siehe Abschnitt 3.1.4.2) eine erste Orientierung. Hier haben sich in den letzten Jahren die vorwärts suchenden heuristischen Planer als besonders erfolgreich erwiesen (Gerevini u. a., 2009). Allerdings liegt den Ergebnissen der IPC eine Problemsammlung zugrunde, die keinen direkten Bezug zum Problem der Dienstekomposition hat. Auch über die IPC hinaus existiert bis jetzt kein Benchmarkproblem für die Dienstekomposition (in intelligenten Umgebungen). Könnten innerhalb der Menge der IPC-Probleme solche gefunden werden, deren Struktur auf das Problem der Dienstekomposition übertragen wer-

[1] Damit in Zusammenhang steht die Entscheidung, wann ein Kompositionslauf zu starten ist. Eine genauere Betrachtung dessen erfolgt im Abschnitt 6.4.
[2] http://ws-challenge.georgetown.edu/wsc07/

den kann, wäre es anhand der IPC-Ergebnisse möglich, Aussagen über die Eignung von Planern zur Dienstekomposition zu tätigen. Allerdings ist die Suche nach vergleichbaren Domänen nicht trivial. Es konnten in den Problemen aus Abschnitt 4.2 keine Strukturen identifiziert werden, die sich zweifelsfrei auf Probleme der letzten IPCs abbilden ließen.
Eine weitere Evaluierung der Effizienz von Planern wurde von Howe und Dahlman (2002) durchgeführt. Dabei wurden verschiedene Planer anhand von 263 verschieden Problemen inklusive der Probleme der ersten IPC (AIPS'98) evaluiert. Ein Ziel der Arbeit war, aufgrund empirischer Daten zu untersuchen, ob bestimmte Probleme von einzelnen Planern unterschiedlich effizient gelöst werden. Die Studie bestätigte, dass es keinen herausstechenden Planer gab. Verschiedene Planer lieferten für verschiedene Probleme beste Ergebnisse.
Howe und Dahlman (2002) untersuchten im Rahmen ihrer Studie weitere Thesen bezüglich der Effizienz von Planern und der verschiedenen Einflussfaktoren auf Planungslaufzeiten. So sind Planer anfällig für syntaktische Änderungen von Domänendefinitionen. In ihren Untersuchungen konnten Howe und Dahlman (2002) nachweisen, dass zufälligen Permutationen der Operatoren und der Prädikate in den Vorbedingungen der Operatoren zu signifikant unterschiedlichen Laufzeiten und Ergebnissen der Planung führten. Zwar schwankte diese Volatilität je Planer zwischen 40% und 4%, dennoch konnte diese Beobachtung für alle untersuchten Planer gemacht werden.[3]
Rintanen (2004) führte eine weitere experimentelle Evaluierung von Planern basierend auf generierten Problemen durch. Dabei betrachtete er explizit das Verhalten der Planer in Phasenübergangsregionen (siehe Abschnitt 5.1.2), was von Howe und Dahlman (2002) nicht berücksichtigt wurde. Mit den Experimenten konnte gezeigt werden, dass für randomisierte Probleme SAT-Solver den heuristischen Planern überlegen waren. Dieses Ergebnis steht im Widerspruch zu den Erfahrungen aus der IPC, in der die heuristischen Planer seit mehreren Jahren dominieren (Hoffmann und Edelkamp, 2005; Gerevini u. a., 2009). Eine mögliche Erklärung dafür ist, dass die von Rintanen generierten Probleme (bis jetzt) keine Entsprechung in der Realität haben und reale Probleme durch entsprechende Heuristiken sehr gut eingegrenzt werden können.
Wenngleich die genannten Evaluierungen erste Abschätzungen der Leistungsfähigkeit der Planer zulassen und darüber hinaus wertvolle Hinweise für den praktischen Einsatz von Planern geben, ist eine direkte Anwendbarkeit der jeweiligen Ergebnisse auf das Problem der Dienstekomposition in intelligenten Umgebungen nicht gegeben. Aus diesem Grund wurden eigene Experimente durchgeführt, um die Effizienz aktueller Planer bei der Verwendung zur

[3] Bei den eigenen Laufzeitexperimenten konnten ähnliche Schwankungen auch für exakt gleiche Probleme beobachtet werden. Dieses Verhalten kann durch eine nicht-deterministische Implementierung der Planungsalgorithmen erklärt werden.

Dienstekomposition in intelligenten Umgebungen zu untersuchen.
Erste Experimente mit Planungsalgorithmen (Marquardt und Uhrmacher, 2008) zeigten, dass sie zwar überwiegend schnell waren, die Laufzeit eines Planungslaufes in wenigen Fällen jedoch unverhältnismäßig länger war, als die vergleichbarer Läufe. Diese Laufzeitspitzen (Peaks) traten insbesondere bei schwierigen Problemen auf. Im Rahmen der Dienstekomposition in intelligenten Umgebungen können solche Peaks die Unaufdringlichkeit der Nutzerassistenz stören, da im Falle eines Peaks ungewöhnlich lange keine Reaktion auf den Nutzer erfolgt. Eine detailliertere Untersuchung der Leistungsfähigkeit der Planer und der Peaks wird in den folgenden Abschnitten beschrieben.

5.1.1 Umfrage

Es ist offensichtlich, dass die Anzahl der Dienste und damit die Anzahl der Operatoren einen wesentlichen Einfluss auf das Laufzeitverhalten eines Planers und somit auch auf das Laufzeitverhalten einer Komposition hat. Um einen groben Überblick über die Anzahl der Dienste in aktuellen intelligenten Umgebungen zu bekommen, wurde eine Umfrage unter Wissenschaftlern, deren Forschung im Bereich der Dienstekomposition in intelligenten Umgebungen liegt (Marquardt u. a., 2008), durchgeführt.
Die Umfrage wurde per Email versandt.[4] Von den gut 20 angeschriebenen Projekten antworteten fünf, was einer Rücklaufquote von knapp 25% entspricht. Für die Umfrage wurden die folgenden drei Metriken identifiziert, da sie einen maßgeblichen Einfluss auf das Laufzeitverhalten einer Kompositionsmethode haben: *Anzahl der atomaren Dienste*, *Anzahl der komponierten Dienste* und *Anzahl der Dienste pro Gerät*. Des Weiteren wurde gefragt, ob eine Dienstekomposition in den jeweiligen Umgebungen verwendet wird und ob diese automatisch abläuft.
Die Umfrage bestand im Kern aus den folgenden vier Fragen:

1. Wie viele Dienste sind in Ihrer intelligenten Umgebung vorhanden?

2. Wieviele atomare und wieviele komponierte Dienste werden in Ihrer intelligenten Umgebung angeboten?

3. Was ist bezüglich der Anzahl der Dienste pro Gerät der üblichere Fall, ein 1:1 (ein Dienst pro Gerät) oder eine 1:n Verhältnis (mehrere Dienste pro Gerät)?

4. Verwenden Sie eine automatische oder semi-automatische Komposition?

[4] Im Anhang C ist der gesamte Text der versendeten Email enthalten.

Von folgenden fünf Projekten wurde auf die Umfrage geantwortet: D-HTN Amigoni u. a. (2005), FOKUS Lee u. a. (2007), Amigo Dupuis u. a. (2007), MagicLamp[5] und MuSAMA Reisse u. a. (2008b) Die Ergebnisse der Umfrage sind in Tabelle 5.1 zusammengefasst.

Projektname	Atomare Dienste	Komponierte Dienste	Gerät / Dienst	Art der Komposition
D-HTN	15	10	1:n	automatisch
FOKUS	30	0	1:1, 1:n	keine
Amigo	15	0	1:1, 1:n	automatisch
MagicLamp	1	6	1:n	keine
MuSAMA	43	2	∼1:2	automatisch

Tabelle 5.1: Ergebnisse der Umfrage aus Marquardt u. a. (2008) (angepasste Version)

Es überwiegen Umgebungen mit wenigen Geräten und Diensten, dabei ist jedoch zu beachten, dass es sich bei allen um prototypische Umsetzungen im Rahmen von Forschungsarbeiten handelt. Realistische Szenarien werden wesentlich mehr Dienste beinhalten, deren Anzahl in Zukunft sicherlich noch weiter ansteigen wird. Drei Projekte verwenden zusammengesetzte bzw. komponierte Dienste, die übrigen basieren lediglich auf atomaren Diensten. Trotzdem lässt sich in neueren Veröffentlichungen eine Tendenz zu komponierten Diensten feststellen (Papazoglou u. a., 2006). Bei den Angaben über das vorwiegende Verhältnis zwischen Gerät und darauf gehosteten Diensten überwiegt das Verhältnis 1:n. Die Projekte Amigo und FOKUS gaben auch ein 1:1 Verhältnis an, was vermuten lässt, dass hier auch Kleingeräte wie Sensoren verwendet werden. Durch die Angaben über die Anzahl der verwendeten Dienste, die kumuliert zwischen 7 und 45 liegt, kann die Anzahl der in einer aktuellen intelligenten Umgebung vorhandenen Geräte auf einen zweistelligen Bereich eingeschränkt werden.
Beim Versuch, die Gesamtmenge der zu betrachtenden Dienste für zukünftige Szenarien abzuschätzen, stellt sich dennoch die Frage, wie groß intelligente Umgebungen werden können und wie feingranular Dienste beschrieben werden. Es ist konzeptionell nicht immer klar, wo eine Umgebung endet, und wann ein Dienst nicht mehr Teil der Umgebung ist. Ist räumliche Nähe oder Nachbarschaft das entscheidende Kriterium für die Zugehörigkeit zu einer Umgebung? Was ist mit im Internet verfügbaren Diensten? Sind diese auch immer Teil der Umgebung und müssen berücksichtigt werden? Im Beispiel 4.1.4 werden Dienste verwendet, die sich nicht in direkter, örtlicher Nähe zum Benutzer

[5] http://www.magiclamp.org

befinden, sondern eher durch einen persönlichen und funktionellen Bezug notwendige Teile der Umgebung sind. Wie, wo und wann wird festgelegt, welche Dienste Teil eines Umgebung sind und welche nicht? Diese Fragen können durch die vorliegende Arbeit nicht beantwortet werden, allerdings geben sie einen Hinweis darauf, dass intelligente Umgebungen bereits heute mit vielen Diensten umgehen können müssen und mittel- bis langfristig zu erwarten ist, dass sich diese Anzahl noch signifikant erhöhen wird.
Im Rahmen dieser Arbeit wird davon ausgegangen, dass an einem Kompositionslauf alle verfügbaren Dienste einer Umgebung teilnehmen. Andere Projekte (Blankenburg u. a., 2008) verwenden einen vorgeschalteten Filter, der die Gesamtzahl der Dienste für einen Kompositionslauf zum Zwecke der Planungszeitoptimierung reduzieren soll. Dabei werden Dienste, die voraussichtlich nicht zur Lösung beitragen, herausgefiltert. Allerdings verwenden Blankenburg u. a. (2008) in diesem Zusammenhang lernende Verfahren, um die Eignung eines Dienstes für ein Problem abzuschätzen. Dadurch ist eine Verwendung des Ansatzes in Ad-hoc-Umgebungen ungeeignet. Zudem zeigen die Erfahrungen aus den Laufzeitexperimenten mit den Planern, dass Planer sehr gut mit unnötigen Operatoren umgehen können und diese die Laufzeit nur unwesentlich beeinflussen, sodass eine vorherige Filterung der Dienste einer intelligenten Umgebung nicht zwingend notwendig erscheint.

5.1.2 Phasentransitionen

Für das weitere Verständnis der Arbeit soll kurz das bereits erwähnte Phänomen der Phasentransitionen erläutert werden. Es betrifft alle Probleme der Komplexität NP-hart (Cheeseman u. a., 1991). Da klassische Planung mit mindestens einer Vorbedingung und einem Effekt pro Operator NP-hart ist (Bylander, 1992), existieren Phasentransitionen auch für klassische Planung (Bylander, 1996). Jedem Problem in NP kann ein Ordnungsparameter zugeordnet werden. So sind beispielsweise beim Travelling-Salesman-Problem die Anzahl der Knoten und Kanten Ordnungsparameter des Problems. Ordnungsparameter der klassische KI-Planung sind beispielsweise die Anzahl der Operatoren oder die Anzahl der Zustandsvariablen. Die Theorie der Phasentransition sagt voraus, dass die Lösung von einzelnen Problemen in einer bestimmten kritischen Region dieses Ordnungsparameters für alle Algorithmen extrem schwer wird. Dadurch benötigen Algorithmen vergleichbar viel Zeit zur Lösung dieser Probleme. Bylander (1996) definiert für die KI-Planung untere und obere Grenzen, die es theoretisch ermöglichen, die Region des Phasenübergangs zu identifizieren. Allerdings unterscheiden sich diese Grenzen um einen Faktor, der exponentiell zur Anzahl der PEs steigt und sind so für eine praktische Anwendung ungeeignet. Bylander (1996) zeigte weiterhin,

dass außerhalb der Phasentransition einfache Algorithmen schnelle Lösungen hervorbringen. Wenn also Probleme der Dienstekomposition in intelligenten Umgebungen nicht im Bereich einer Phasentransition liegen, genügen einfache Planungsalgorithmen, um die Probleme schnell zu lösen.

5.1.3 Eigene Laufzeitexperimente

Um die Leistungsfähigkeit von Planungsalgorithmen in intelligenten Umgebungen experimentell evaluieren zu können, wird zunächst ein Modell einer solchen Umgebung benötigt. Dabei stellt sich die Frage, wie intelligente Umgebungen charakterisiert werden können. Können Metriken bzw. Attribute gefunden werden, die helfen, intelligente Umgebungen hinsichtlich der Dienstekomposition zu unterscheiden? Diese Metriken sollten leicht messbar sein und einen Effekt auf die Effizienz der Komposition haben.
Ein Planungsproblem besitzt eine Vielzahl von Parametern (siehe Abschnitt 3.1). Aus dieser Menge wurden fünf Parameter ausgewählt, da sie einerseits einen Einfluss auf die Effizient der Planung haben und des Weiteren in einer intelligenten Umgebung leicht gemessen werden können. Bei der Benennung der Parameter wird die Notation von Bylander (1996) verwendet:

- o - Anzahl der Operatoren
- n - Anzahl der Propositionen
- i - Anzahl der wahren initialen Weltzustände
- g - Anzahl der Zielzustände
- r und s - Anzahl der Vorbedingungen und Effekte pro Operator

Da Dienstekomposition in intelligenten Umgebungen ein junges und spezialisiertes Forschungsfeld ist und es daher an ausdifferenzierten Anwendungen mangelt, war es nicht möglich, charakteristische Strukturen zu identifizieren, anhand derer spezielle Problemklassen abgeleitet werden konnten und das Problem der Dienstekomposition so gegen andere Planungsprobleme abgegrenzt werden konnte. Daher wird auf künstlich generierte Probleme innerhalb von Parameterintervallen, wie sie auch in intelligenten Umgebungen auftreten können, zurückgegriffen, wodurch allgemeine Problemklassen definiert werden.
Der wesentliche Nachteil bei der automatischen Generierung von Probleminstanzen ist deren fragwürdige Relevanz (Weihe, 2001). Daher muss ein sehr großer Problemraum untersucht werden, um die Wahrscheinlichkeit zu erhöhen, auch relevante Probleme abzudecken.

Durch die Verwendung von randomisierten Problemem überlappen sich die im Folgenden beschriebenen Experimente teilweise mit den Arbeiten von Bylander (1996) und Rintanen (2004).
Die Ergebnisse der in Abschnitt 5.1.1 beschriebenen Umfrage sowie die Erfahrungen aus den ersten Experimenten (Marquardt und Uhrmacher, 2008) dienen dabei als Ausgangspunkt für die folgenden Experimente, deren vorläufige Ergebnisse in Marquardt und Uhrmacher (2009c) veröffentlicht wurden.

5.1.3.1 Experimentaufbau

Mit Hilfe der Experimente soll basierend auf den Parametern o, n, i, g und r eine möglichste breite aber realistische Menge an Planungsproblemen erstellt und untersucht werden. Dafür wird in bestimmten Intervallen über alle fünf Variablen iteriert, sodass ein 5-dimensionaler Problemraum entsteht. Um die Durchfürhbarkeit der Experimente zu gewährleisten, wird sich dabei auf eine Auflösung von 10 Schritten pro Parameter beschränkt, was zu einer maximalen Problemmenge von $10^5 = 100.000$ Problemen führt.
In den Experimenten ist die Anzahl der Vorbedingungen und Effekte eines Operators immer gleich (daher gilt $s = r$) und gerade. PEs beginnen mit 2 und werden mit einer Schrittweite von 2 auf 20 erhöht. Genauso wird i von 2 auf 20 erhöht. Aus den ersten zwei Dimensionen r und i resultieren 55 sinnvolle Kombinationen.[6] Für jede dieser 55 Kombinationen werden auch die Werte von o (10 bis 100, Schrittweite = 10), n (10 bis 100, Schrittweite = 10) und g (1 bis 10, Schrittweite = 1) iteriert. Was zu 1000 weiteren Kombinationen für jede der 55 Kombinationen aus r und i führt.[7] Insgesamt wurden so über 50.000 Problemkonfigurationen identifiziert. Für jede Konfiguration wurden 100 Replikationen mit unterschiedlichen Startwerten (seeds) des Zufallsgenerators durchgeführt, was in einer Gesamtmenge von über 5.000.000 Probleminstanzen resultiert.
Um die absolute Laufzeit der Experimente auf einen realistischen und vorhersagbaren Wert zu begrenzen, wurde die maximale Laufzeit eines Planungslaufes auf 2000 ms beschränkt. Einige Planer erlauben die Angabe einer maximalen Laufzeit als Parameter. Leider konnten diese Parameter in den Experimen-

[6] Nicht alle Kombinationen von r und i in der beschriebenen Spanne sind sinnvoll. Zum Beispiel können Operatoren mit 12 Vorbedingungen in einem Problem, in dem lediglich 10 Propositionen im initialem Weltzustand wahr sind, nie angewendet werden, was die Lösung eines solchen Problems unmöglich macht. Im Anhang D sind die sich daraus ergebenden Dreiecksmatrizen in den entsprechenden Scattersplots zwischen „initWS“ und „PEs“ zu erkennen.

[7] Vergleichbar zu r und i sind auch hier nicht alle Kombinationen sinnvoll, so ist ein Problem mit 10 Zielzuständen aber insgesamt nur 10 möglichen Propositionen unlösbar, da in diesem Fall keine Propositionen zur Beschreibung des initialen Weltzustandes übrigbliebe.

ten nicht verwendet werden, da zunächst nicht alle Planer diese Parametrisierung unterstützten und beispielsweise Vor- und Nachverarbeitungsschritte der Planer teilweise nicht berücksichtigt wurden. Darüber hinaus überschritten in den Experimenten einige Laufzeiten, die über einen Parameter begrenzt waren, diese Grenze um ein Vielfaches. Daher wurden alle Planerprozesse durch einen externen Timer kontrolliert und beendet, sobald die Laufzeit 2000 ms überschritten hatte.
In den Experimenten wurden drei Planer in ihren jeweils aktuellsten Versionen (Stand Ende 2008) untersucht: SGP (Weld u. a., 1998), blackbox (Kautz und Selman, 1998), und LPG (Gerevini und Serina, 2002). Alle Planer unterstützen PDDL und stellen aktuelle Vertreter verschiedener Planungsalgorithmen dar. SGP ist ein Graphplaner, blackbox verwendet einen SAT-Solver und LPG benutzt heuristische Planung. Es wurde sich auf drei Planer beschränkt, da die Dauer der Experimente pro Planer bei sieben 24-Stunden-Tagen lag. Obwohl auch UCPOP (Penberthy und Weld, 1992) ein Kandidat für die Experimente war, wurde dieser Planer nicht verwendet, da er kein PDDL unterstützt.
Da drei Planer untersucht wurden, führten die mehr als 5.000.000 Probleminstanzen zu über 15.000.000 Experimentläufen. Alle Läufe wurden auf drei Maschinen ausgeführt, die über eine komplett gleiche Konfiguration verfügten (Intel Core2Duo E8200 @ 2.66GHz mit 2GB RAM unter WindowsXP ServicePack 3). Die reine Rechenzeit der Experimente betrug ungefähr drei Wochen. Alle Experimente wurden mitsamt ihrer Parametrisierungen in einer Experimentdatenbank gespeichert, was es erlaubt die Läufe im Nachhinein zu reproduzieren. Durch die Versuche wurden über 2GB Daten in der Datenbank gespeichert.

Problemgenerator Rintanen (2004) stellt drei verschiedene Modelle zur Generierung von Planungsproblemen vor. Für die hier beschriebenen Experimente wird das Modell genutzt, welches auch von Bylander (1996) verwendet wurde und kurz erläutert werden soll.
Zunächst wurde entsprechend der Anzahl der PEs pro Operator eine Menge von Propositionen ausgewählt. Die erste Hälfte davon bildet die Vorbedingung des Operators die zweite Hälfte bildet die Effekte. Dadurch können keine Operatoren entstehen, die gleiche Propositionen in Vorbedingung und Effekt gleichzeitig haben. Um den Effekt eines Operators zufällig zu generieren, wurde die Menge der Propositionen wieder geteilt. Aus einer Hälfte dieser Propositionen ergeben sich die positiven, aus der anderen Hälfte die negativen Effekte des Operators. Auf diese Art und Weise werden viele Instanzen generiert, die sehr leicht als unlösbar zu klassifizieren sind. Diese Kritik wurde von Rintanen (2004) aufgenommen und veranlasste ihn, ein alternatives Modell zu entwerfen, in dem viele dieser unlösbaren Probleme ausgeschlossen wer-

den. Da in intelligenten Umgebungen nicht davon ausgegangen werden kann, dass eine Lösung für ein Kompositionsproblem gefunden wird, ist diese Einschränkung bei der Generierung der Probleme für intelligente Umgebungen nicht zulässig. Dennoch ist es möglich zu argumentieren, dass durch die Verwendung dieses Modells zu viele triviale Probleme erzeugt werden, die keine Herausforderung für die Planer darstellen und es in intelligenten Umgebungen weiterhin wahrscheinlich ist, dass eine Lösung existiert. Da diese Wahrscheinlichkeit allerdings nicht zu quantifizieren ist und die Planer für alle möglichen Probleme evaluiert werden sollten, wurde dennoch das oben beschriebene Modell verwendet.

Für die Domänenbeschreibungen wurden parameterfreie Operatoren verwendet. Im Gegensatz zu Bylander und Rintanen wurde auch der Einfluss der wahren Propositionen im initialen Weltzustand auf die Planung explizit untersucht. Der initiale Weltzustand wird dabei vom Problemgenerator immer so generiert, dass er disjunkt zum Zielzustand ist.

Alle Probleminstanzen wurden in PDDL erzeugt und in diesem Format an die Planer übergeben.

5.1.3.2 Hypothesen

Mit den Laufzeitexperimenten sollte untersucht werden, ob aktuelle Planer Probleme der Dienstekomposition in intelligenten Umgebungen, wie einleitend gefordert, im Sekundenbereich lösen können, ob ihre Reaktionszeit robust und nicht zu volatil ist und ob die Ergebnisse verlässlich sind. In ersten Experimenten zeigte sich bereits, dass Planer in Regionen mit schweren Problemen dazu tendieren in Laufzeitspitzen zu geraten. Aus diesen Erfahrungen und den zu Beginn des Abschnitts beschriebenen Beobachtungen von Howe und Dahlman (2002) (siehe Abschnitt 5.1) ergeben sich die folgenden Hypothesen, die mit Hilfe der Experimente zu überprüfen sind:

1. Wenn ein Planungslauf in eine Laufzeitspitze gerät, ist es sehr wahrscheinlich, dass kein Plan gefunden wird und der Lauf daher vorzeitig abgebrochen werden kann.

2. Verschiedene Planer haben verschiedene Laufzeitspitzen.

3. Es kann eine signifikante Problemregion identifiziert werden, die stellvertretend für eine größere Region steht, um Vorhersagen über die Effizienz von Planern darin zu treffen.

5.1.3.3 Ergebnisse

Für jeden Lauf wurden verschiedene Charakteristiken aufgezeichnet. Zunächst wurde die Laufzeit gespeichert.[8] Zusätzlich wurden die Rückgaben der Planer gespeichert. Die Rückgabe ist entweder ein Plan oder eine Aussage, dass kein Plan gefunden werden kann. Einige Planer geben Gründe für eine erfolglose Planung aus, wobei LPG die detailliertesten Beschreibungen zurückgab. Des Weiteren wurde gespeichert, ob ein Plan gefunden wurde und ob ein Planungslauf nach Überschreitung des Zeitlimits abgebrochen wurde.
Durch die Aggregation aller 100 Replikationen einer Probleminstanz können vier verschiedene Messwerte pro Instanz ermittelt werden: Durchschnittliche Laufzeit μ (avgTime), Standardabweichung σ der Laufzeiten (stdTime), Quote der gefundenen Lösungen (avgSolutions) und Quote der abgebrochenen Läufe (avgAborted). Abbildung 5.1 zeigt die Ergebnisse der Messwerte, in Abhängigkeit zu den Experimentvariablen (o, n, i, g, und r).
Die erste Zeile der Abbildung 5.1 zeigt die durchschnittliche Laufzeit der Planer. Hier ist blackbox immer am schnellsten, gefolgt von LPG und SGP. Bis auf das letzte Diagramm, in dem der Einfluss der PEs auf die Laufzeit dargestellt wird, ist auch die Form der Kurven der einzelnen Planer ähnlich.
Die Standardabweichung σ ist ein guter Indikator für die Komplexität einer Problemregion. Wenn sich die Laufzeiten von verschiedenen Läufen einer Probleminstanz sehr stark unterscheiden, kann die Problemmenge als schwierig erachtet werden. Aus diesem Grund ist die zweite Zeile von besonderem Interesse. Wenngleich blackbox auch hier die besten Ergebnisse zeigt, schrumpft der Unterschied zwischen den Planern im Gegensatz zur durchschnittlichen Laufzeit deutlich und die zu beobachtenden Kurven zeigen wesentlich mehr Unregelmäßigkeiten. Bezüglich der Anzahl der Zustandsvariablen im zweiten Diagramm zeigen alle Planer ein Maximum. LPG und SGP errreichen ihr Maximum bei 20 Zustandsvariablen, wohingegen blackbox' Standardabweichung bis 30 Zustandsvariablen ansteigt. Das vierte Diagramm zeigt, dass LPG mit steigender Anzahl von Zielzuständen robuster wird, während die Instabilität von blackbox im Rahmen der betrachteten Probleme stetig wächst. SGP stagniert auf hohem Level.
Die Quote der gefundenen Lösungen in der dritten Zeile beschreibt den Anteil der gefundenen Lösungen pro Probleminstanz. Die Diagramme zeigen, dass alle Planer nah beieinander liegen, was bedeutet, dass sie ähnlich viele Lösungen finden. SGPs Ergebnisse sind immer etwas schlechter als die von blackbox und LPG. Der Grund hierfür ist die im Schnitt längere Laufzeit von SGP und der Tatsache, dass alle Läufe nach 2000 ms abgebrochen wurden, sodass einige

[8] Zeitangaben werden im Zusammenhang mit den Experimenten immer in Millisekunden (ms) angegeben.

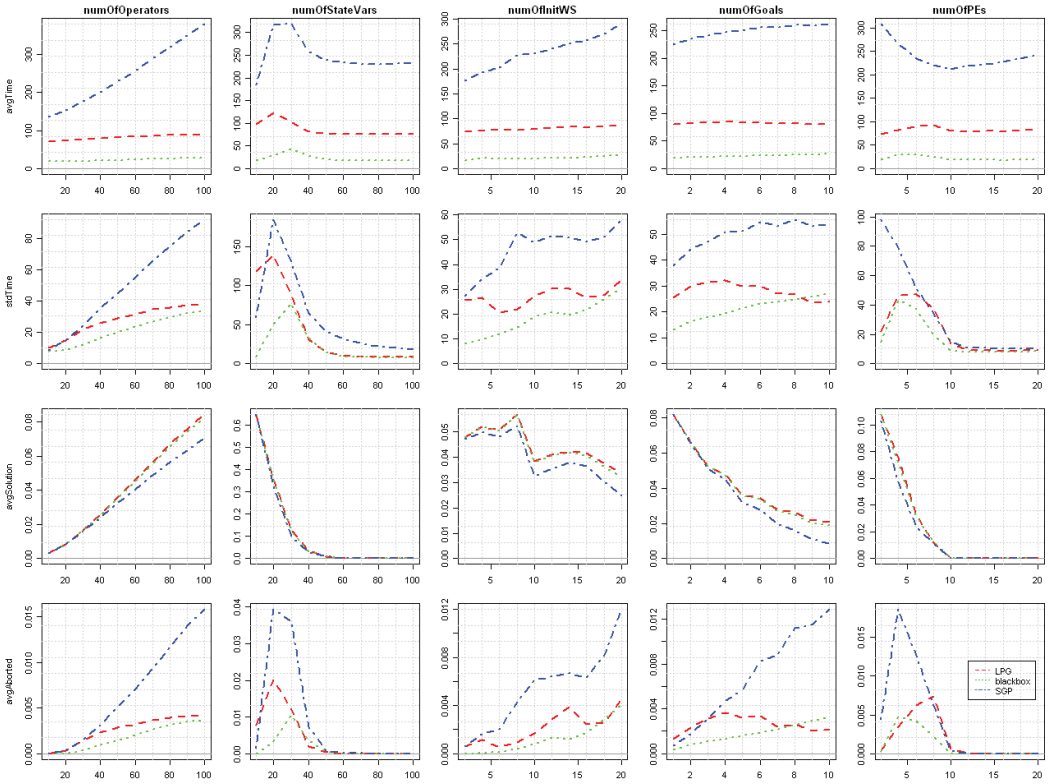

Abbildung 5.1: Kennlinien für die durchschnittliche Laufzeit (avgTime), Standardabweichung (stdTime), Quote der gefundenen Lösungen (avgSolution) und Quote der abgebrochenen Läufe (avgAborted) für alle fünf Experimentvariablen o (numOfOperators), n (numOfStateVars), i (numOfInitWS), g (numOfGoals) und r (numOfPEs). Jede Kennlinie stellt das Verhalten eines der drei Planer LPG (gestrichelt, rot), blackbox (gepunktet, grün) und SGP (Strich-Punkt, blau) dar.

Lösungen von SGP nicht mehr gefunden werden konnten, während LPG und blackbox schon eine Lösung gefunden hatten. Die Ergebnisse der vierten Zeile (Quote der abgebrochenen Läufe pro Instanz) sind zu großen Teilen vergleichbar mit denen der zweiten Zeile. Beide sind Indikatoren für Regionen schwerer Probleme.

Neben den Kennlinien lassen sich auch Scatterplots erstellen, die eine genauere Darstellung der interessanten Problemregionen zulassen.[9] Eine Sammlung aller Scatterplots findet sich aufgrund ihrer Größe im Anhang D.

[9] Es ist zu beachten, dass die Skalen aller Scatterplots in dieser Arbeit adaptiv sind und sich zwischen den einzelnen Diagrammen unterscheiden können. Der Grund dafür ist, dass für die hier präsentierten Fragestellungen lediglich die Tendenzen und nicht die absoluten Werte von Interesse sind. Daher wurde für die Matrizendiagramme auch auf Legenden verzichtet. Rote Farbe steht in allen Diagrammen für hohe Werte, grün stets für niedrige Werte.

Identifizierung von Phasentransitionen Bei einem Vergleich der Diagramme, die σ (*avgSolution*) und μ (*stdTime*) in Abhängigkeit von der Anzahl der Zustandsvariablen (n) sowie der Anzahl der Ziele (g) darstellen, werden die Charakteristiken einer Phasenübergangsregion deutlich (siehe Abbildungen 5.2 und 5.3). Bei der Betrachtung von zwei Variablen werden gleichzeitig Phasentransitionen zwei verschiedener Richtungen deutlich, sodass in dieser Darstellung auch der unterschiedliche Einfluss der jeweiligen Variablen zu erkennen ist.

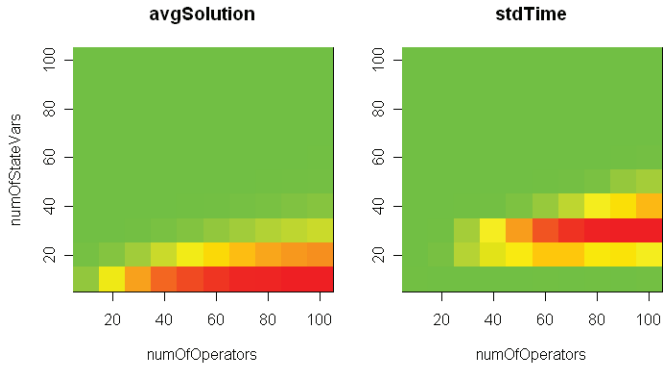

Abbildung 5.2: Visualisierung der Phasentransitionsregion bezüglich der Variablen n und o anhand der durchschnittlich gefundenen Lösungen (avgSolution) und Darstellung der Standardabweichung (stdTime) im selben Bereich. (Planer: blackbox)

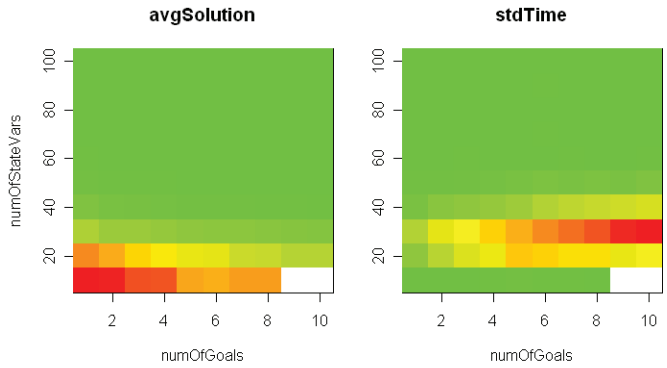

Abbildung 5.3: Visualisierung der Phasentransitionsregion bezüglich der Variablen n und g anhand der durchschnittlich gefundenen Lösungen (avgSolution) und Darstellung der Standardabweichung (stdTime) im selben Bereich. (Planer: blackbox)

Die Wahrscheinlichkeit auf eine Lösung fällt in beiden Diagrammen propor-

	Anzahl	Quote	avgTime	stdTime
LPG	8,724	4.5%	125.37	153.20
blackbox	185,106	95.5%	21.73	29.84
SGP	-	0.0%	0	0

Tabelle 5.2: Läufe, die zu einer Lösung führten und vom entsprechenden Planer am schnellsten gelöst wurden

tional zur Anzahl der Zustandsvariablen n von einem hohen Wert (rot) auf nahezu Null (grün). Sie ist ebenfalls proportional zur Anzahl der Operatoren o und invers proportional zur Anzahl der Ziele g.
Beginnend mit einem geringen Wert bei $n = 10$ wächst μ mit steigenden n bis das Maximum bei $n = 30$ erreicht wird. Im Anschluss fällt sie wieder stetig auf ein konstant niedriges Niveau bei $n = 50$. Interessanterweise treten die höchsten Werte der Standardabweichung in Regionen auf ($n = 30$, $g > 6$ bzw. $o > 40$), in denen die Wahrscheinlichkeit, eine Lösung zu finden, fast an ihrem Minimum ist.
Basierend auf den letzten beiden Diagrammen und der Abbildung 5.1 konnte experimentell nachgewiesen werden, dass im Vergleich zu g und o die Anzahl der Zustandsvariablen n den größten Einfluss auf die Laufzeit eines Planungsproblems hat.
Ziel einer Evaluierung ist es, den besten Kandidaten herauszufinden. Für den gesamten Datensatz lässt sich feststellen, dass blackbox in 99,6% der Läufe am schnellsten ein Ergebnis zurückgab, gefolgt von LPG (0.0038%) und SGP (0.00002%). Werden lediglich Probleme betrachtet, die zu einer Lösung führten (vgl. Tabelle 5.2), verschiebt sich das Ergebnis zugunsten von LPG (4.5%), dennoch dominiert blackbox (95.5 %). Ein Blick auf die durchschnittliche Laufzeit verrät, dass blackbox heraussticht, wenn die Laufzeiten klein sind, LPG wird mit wachsenden Laufzeiten besser.

Spezifischer Offset bei der Ausführung der Planer Alle Implementierungen der Planer ziehen einen gewissen Offset nach sich, der bei jeder Ausführung der Planer seinen Tribut fordert und unabhängig von der Komplexität des gegenwärtigen Problems ist. Um eine grobe Abschätzung der Offsets der Planer zu erhalten, wurden zwei Problemregionen genauer untersucht. In der ersten Region ist die Wahrscheinlichkeit für eine Lösung beinahe Eins, hier existieren viele mögliche Lösungen und die Planer sind in der Lage, eine dieser Lösungen schnell zu finden. Diese Region kann mit folgenden Werten der Experimentvariablen beschrieben werden ($o = 100, n = 10, g = 2, r = 2$). Die zweite Region liegt hinter der Phasentransition, wo es sehr unwahrscheinlich ist, dass ein Problem eine Lösung hat, und Planer in der Lage sind, auch das

	blackbox	LPG
$n = 20$	1.568 ‰	4.118 ‰
$n = 30$	3.943 ‰	2.431 ‰
$n = 40$	0.734 ‰	0.894 ‰
$n = 50$	0.121 ‰	0.358 ‰

Tabelle 5.3: Wahrscheinlichkeit einer Laufzeitspitze für Probleme mit $o > 60$

schnell zu realisieren. Eine solche Region liegt bei $o = 10$, $n = 30$ und $r = 14$. Damit ist in beiden Regionen der Anteil der echten Planung an der Gesamtlaufzeit des Planers minimal, sodass hier eine untere Schranke für die Offsets der Planer identifiziert werden kann. Die minimalen Laufzeiten in diesen Regionen sind 15 ms für blackbox, 62 ms für LPG and 140 ms für SGP. Die durchschnittlichen Laufzeiten sind entsprechend 18 ms, 75 ms und 150 ms. Beide Werte liegen sehr nah beieinander, sodass davon ausgegangen werden kann, dass es sich hierbei um eine untere Schranke für die Offsets der Planer handelt. Werden diese Offsets bei den Ergebnissen (z.B. in den Diagrammen von Abbildung 5.1 oder in Tabelle 5.2) berücksichtigt, relativiert sich die Dominanz von blackbox.

5.1.3.4 Überprüfung der Hypothesen

1. Laufzeitspitzen führen wahrscheinlich zu keiner Lösung. Wenn ein Planungslauf in eine Laufzeitspitze gerät, ist es sehr wahrscheinlich, dass kein Plan gefunden wird und der Lauf daher vorzeitig abgebrochen werden kann.
Zunächst ist es interessant, wie oft Laufzeitspitzen auftreten. In Tabelle 5.3 wird das Verhältnis zwischen Peaks und der Gesamtzahl der Läufe dargestellt. Alle Läufe mit einer Dauer von mehr als 500 ms wurden dabei als Peaks angesehen. Weiterhin wurden nur Läufe betrachtet, die nicht abgebrochen wurden. Aufgrund der im Schnitt sehr hohen Laufzeiten wurden die Ergebnisse von SGP hier nicht betrachtet. Da Peaks vor allem in Regionen eines Phasenübergangs zu erwarten sind, wurden die entsprechend interessanten Probleme betrachtet. Dazu zählen alle Probleme mit mehr als 60 Operatoren im Bereich von 20 bis 50 Zustandsvariablen. In den Scatterplots im Anhang D lässt sich für alle Planer erkennen, dass innerhalb dieses Bereiches die Standardabweichung am höchsten ist und es sich daher um eine Region mit schweren Problemen handelt.
Das Auftreten von Laufzeitspitzen ist für beide Planer sehr niedrig. Während es seinen Maximalwert für LPG bei $n = 20$ hat, wird dieser bei blackbox erst bei $n = 30$ erreicht und fällt danach bei beiden Planern deutlich. Diese Werte

	blackbox Gesamt	Peaks	LPG Gesamt	Peaks
$n = 20$	54.20 %	54.02 %	55.40 %	67.83 %
$n = 30$	23.08 %	95.53 %	23.74 %	66.13 %
$n = 40$	6.84 %	100.00 %	7.44 %	29.56 %
$n = 50$	1.74 %	92.86 %	1.84 %	10.84 %

Tabelle 5.4: Wahscheinlichkeiten einer Lösung für Probleme mit $o > 60$

decken sich mit der Beobachtung, die im Diagramm, was die Standardabweichung bezüglich der Anzahl der Zustandsvariablen in Abbildung 5.1 darstellt, zu erkennen ist (siehe oben).
In Tabelle 5.4 sind die Wahrscheinlichkeiten, eine Lösung zu finden, dargestellt. Die erste Spalte pro Planer („Gesamt“) zeigt als Orientierung und Vergleichswert die Wahrscheinlichkeit für alle Probleme an, die zweite Zeile („Peaks“) die Wahscheinlichkeit für Peaks.
Für $n = 20$ also vor bzw. zu Beginn der Phasentransition ist die Wahrscheinlichkeit, unter den Peaks eine Lösung zu finden, gleich der der restlichen Probleme. Im Anschluss fallen die Wahrscheinlichkeiten bei den normalen Problemen (in den Spalten „Gesamt“) für blackbox und LPG fast identisch auf knapp 2% bei $n = 50$. Bei der Betrachtung der Peaks zeigt sich jedoch ein Unterschied zwischen den beiden Planern. Beide starten mit vergleichbaren Werten für $n = 20$. Während die Wahrscheinlichkeiten, dass ein Peak zu einer Lösung des Planungsproblems führt, bei blackbox in der Folge stark ansteigen, fallen sie bei LPG. Allerdings ist das Gefälle bei LPG weniger stark als bei den normalen Problemen, sodass geschlussfolgert werden kann, dass für beide Planer die Wahrscheinlichkeit, bei einer Laufzeitspitze eine Lösung zu erwarten, höher ist, als die durchschnittliche Wahrscheinlichkeit auf eine Lösung in dieser Region ähnlicher Probleme.
Eine erste Vermutung, dass das Ergebnis einer Laufzeitspitze auf die relative Lage des Problems zur Phasentransition zurückzuführen ist und damit zur generellen Wahrscheinlichkeit einer Lösung in dieser Problemregion korreliert, konnte nicht bestätigt werden. Auf beiden Seiten der Phasentransition tendieren Peaks dazu, einen Plan zu beinhalten. Aufgrund dieser experimentellen Ergebnisse muss die erste Hypothese verworfen werden.

These 8 *Planungsläufe einzelner Probleme können zu Laufzeitspitzen führen. Grund dafür ist das Phänomen der Phasentransition. Auftretende Peaks sind daher unvermeidbar. Laufzeitspitzen (Peaks) von Planungsläufen gefährden die Unaufdringlichkeit KI-planungsbasierter Komposition. Die Wahrscheinlichkeit, dass ein in einem Peak resultierender Lauf zu einer Lösung führt,*

ist höher, als die Wahrscheinlichkeit ähnlicher Probleme zu einer Lösung zu führen.

2. Verschiedene Planer haben verschiedene Peaks. In den Experimenten wurden für gleiche Kombinationen der Experimentvariablen jeweils 100 Läufe durchgeführt. Die Abbildungen 5.4 und 5.5 zeigen die Ergebnisse dieser 100 Läufe der Planer LPG und blackbox für eine bestimmte Kombination der Experimentvariablen o, n, i, g und g.

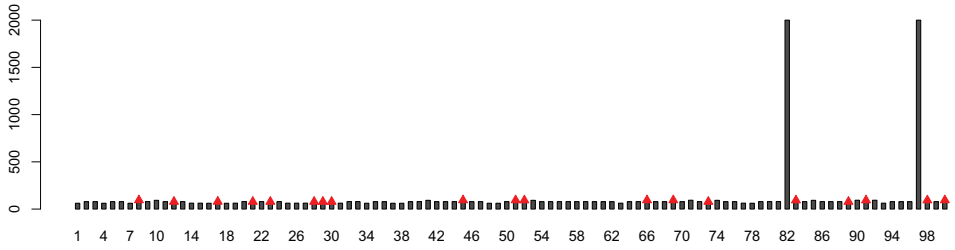

Abbildung 5.4: Laufzeiten und Lösungen von LPG für die Problemkombination $o = 100$, $n = 30$, $i = 10$, $g = 10$ und $r = 4$.

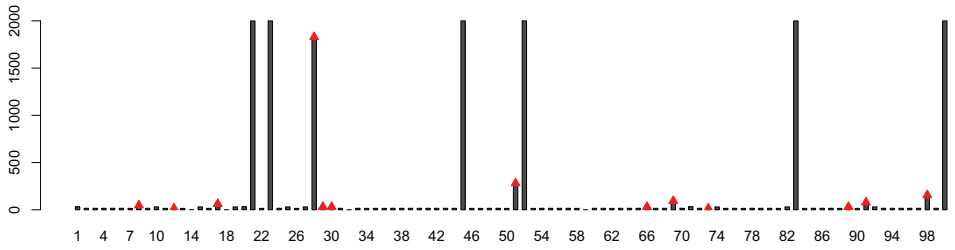

Abbildung 5.5: Laufzeiten und Lösungen von blackbox für die Problemkombination $o = 100$, $n = 30$, $i = 10$, $g = 10$ und $r = 4$.

Läufe, die zu einer Lösung führten, sind in den Abbildungen durch ein rotes Dreieck am entsprechenden Balken gekennzeichnet. Da die X-Achsen beider Diagramme gleiche Probleminstanzen an gleicher Stelle abbilden, lässt sich vergleichen, wie die beiden Planer mit exakt gleichen Problemen umgehen. Beim Vergleich beider Abbildungen ist deutlich zu erkennen, dass die Planer

bei der Mehrzahl der Läufe schnell terminieren und im Wesentlichen bei den selben Problemen eine Lösung zurückgeben. Bei einzelnen Läufen lassen sich die Peaks erkennen, die im Rahmen der Experimente nach 2000 ms abgebrochen wurden. Es ist weiterhin zu sehen, dass für LPG deutlich weniger Peaks auftreten bei Problemen, die blackbox in einen Peak laufen lassen. Allerdings ist LPG nicht immer besser als blackbox. Beide Peaks von LPG (Lauf 82 und Lauf 97) werden von blackbox schnell gelöst.
Werden pro Planer die Läufe aufaddiert, bei denen nur der andere Planer in einen Peak läuft, ergibt sich ein Maß, welches, auf den gesamten Datenbestand angewandt, einen Hinweis darauf gibt, welcher Planer für welche Problemregion besser geeignet ist und so eine Aussage zulässt, wie unterschiedlich die Planer gleiche Probleme lösen. Dieses Maß soll *Dominanz bezüglich der Peaks* (kurz: Dominanz) genannt werden.
Anhand der beiden Abbildungen 5.4 und 5.5 soll die einfache Errechnung der Dominanz veranschaulicht werden. Von den 100 Läufen von blackbox werden 6 Läufe nach 2000 ms abgebrochen, und damit als Peaks gewertet. Da beide Diagramme auf der X-Achse die gleichen 100 Probleme darstellen, ist zu erkennen, dass alle sechs Läufe die bei blackbox zu Peaks führen, durch LPG gefunden werden. Dagegen werden die zwei Läufe die bei LPG Peaks verursachen, schnell durch blackbox gelöst. Für die dargestellten 100 Läufe der Problemkombination ergibt sich somit der Wert 6 für LPG und der Wert 2 für blackbox. Der kumulierte Wert, der aussagt, wie unterschiedlich die Planer bei der Lösung dieser Problemkombination sind, ist demzufolge 8.[10]
Wird die Gesamtheit der Daten betrachtet, lassen sich die Werte aggregieren und es ergibt sich eine Kennline (siehe Abbildung 5.6). Absolut kann LPG in 4071 Fällen ein Ergebnis zurückgeben, während blackbox mehr als 2000 ms benötigt. Blackbox schafft das in 7972 Fällen.

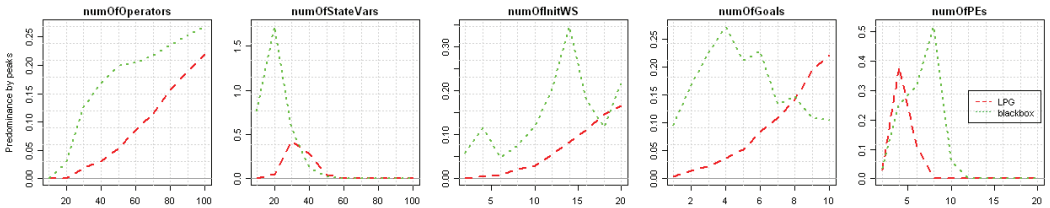

Abbildung 5.6: Kennlinien der Dominanz der Planer LPG (gestrichelt) und blackbox (gepunktet)

In Abbildung 5.6 lässt sich gut erkennen, dass die Dominanz von blackbox

[10] Da sich in den Experimenten immer auf eine Grundgesamtheit von je 100 Läufen pro Problemkonfiguration bezogen wird, ist eine Normierung des Dominanzwertes nicht nötig, um verschiedene Problemkonfigurationen zu vergleichen.

vor allem bei steigender Anzahl der Ziele („numOfGoals“) abnimmt und LPG stärker wird.
Für die Dominanz lassen sich wie für die statistischen Daten im letzten Abschnitt neben den Kennlinien auch Scatterplots (siehe Anhang D) erstellen. Da blackbox für die durchgeführten Experimente als schnellster Planer identifiziert wurde, ist vor allem interessant, wann LPG besser als blackbox war (siehe Abbildung D.13). In den Abbildungen 5.7 ist zu erkennen, dass es eindeutig Regionen gibt, in denen LPG blackbox überlegen ist und diese Dominanz nicht zufällig auftritt. Damit kann die zweite Hypothese bestätigt werden.

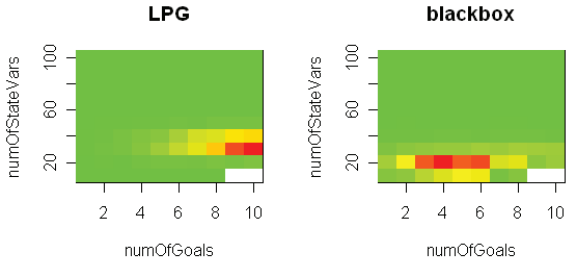

Abbildung 5.7: Dominanz der Planer LPG und blackbox zueinander

Bei Betrachtung der kummulierten Dominanz in er im Anhang beigefügten Abbildung D.15, ist zu erkennen, dass es für große Teile des Problemraumes Unterschiede bei den Planern gibt.

3. Signifikante Problemregion. Kann eine signifikante Problemregion identifiziert werden, die stellvertretend für eine größere Region steht, um Vorhersagen über die Effizenz von Planern darin zu treffen?

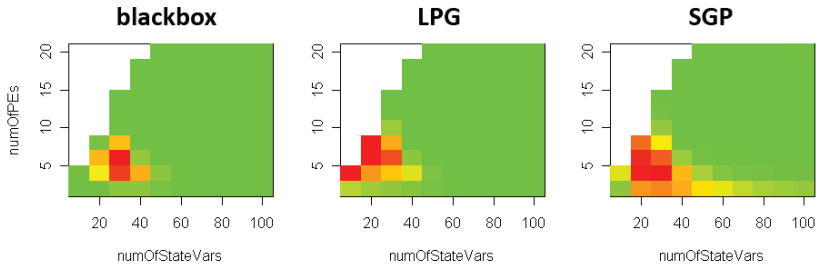

Abbildung 5.8: Standardabweichung der Planungslaufzeit zur Identifizierung von schweren Problemen

Die interessanteste Region ist sicherlich die der Phasentransition, in der die Wahrscheinlichkeit, eine Lösung zu finden, rapide von fast Eins auf fast Null

fällt. Diese Region ist durch eine hohe Standardabweichung der Laufzeiten gekennzeichnet. Anhand der Experimentdaten lässt sich in Abbildung 5.8 eine solche kleinere Region, in der die Standardabweichung der Läufe sehr hoch ist, isolieren. Unter Berücksichtigung aller Planer befindet sich diese interessante Teilmenge im Bereich von $r < 10$ und $n < 50$. Alle übrigen Probleme sind durch eine konstant niedrige Standardabweichung gekennzeichnet. Dennoch ist der interessante Problemraum viel zu groß, um ihn beispielsweise zur Laufzeit einer intelligenten Umgebung im Hintergrund auszuwerten. Die Identifizierung einer signifikanten Problemregion kann induktiv auf Grundlage von umfangreichen Experimentaldaten bestimmt werden. Eine deduktive Bestimmung dieser Region aufgrund von bestimmten Parametern ist zu komplex.

5.1.4 Experimente mit den Beispielszenarien

Eine weitere Möglichkeit, die Laufzeit der Planer zu evaluieren, ist die Verwendung der in Abschnitt 4.1 vorgestellten Szenarien. Für diese Evaluierung wurden neben den beschrieben Szenarien (vgl. Anhang A) auch vier Varianten einer erweiterten Version des einfachen SmartLab Szenarios verwendet. Im einfachen SmartLab-Szenario, das im Abschnitt 4.1 vorgestellt wurde, sollte ein Dokument auf einer bestimmten Leinwand angezeigt werden. In der erweiterten Variante kommen einerseits neue Operatoren hinzu, die es ermöglichen Dokumente zu konvertieren. Weiterhin ist die Domäne so modelliert, dass es möglich ist, mehrere Dokumente auf jeweils verschiedenen Leinwänden anzeigen zu lassen.
Neben den im letzten Abschnitt verwendeten Planern kam hierbei auch MIPS-XXL (Edelkamp u. a., 2006) zum Einsatz. MIPS-XXL zeigte bei den letzten IPCs sehr gute Ergebnisse, allerdings ist MIPS-XXL ein sequentieller Planer und erzeugt so lediglich TOPs. Des Weiteren wurde ein zweiter im LPG enthaltener Algorithmus separat betrachtet. In der Standardkonfiguration nutzt LPG zunächst einen Graphplan-basierten Algorithmus. Findet dieser nach einer bestimmten Anzahl von Iterationen keine Lösung, wechselt LPG intern auf einen heuristischen Suchalgorithmus, der dem Planer FF (Hoffmann und Nebel, 2001) entliehen ist. Durch Angabe eines Parameters kann in der verwendeten Implementierung von LPG die alleinige Verwendung eines der beiden Algorithmen bestimmt werden. Der zweite Algorithmus wird in der Tabelle *FF* genannt. Für diese Experimente kamen daher 5 Planer zum Einsatz (LPG, FF, blackbox, SGP und MIPS-XXL)
Es wurden pro Planer und Problem 100 Replikationen durchgeführt, was bei acht Problemen zu 4000 Läufen führte. Diese liefen auf einem Intel Core2Duo T5500 @ 1.66GHz mit 2GB RAM unter Windows Vista SP2. Die maximale Laufzeit wurde auf 20.000 ms pro Lauf festgesetzt. Die Ergebnisse sind in

Problem / Planer	LPG		FF		SGP		MIPS-XXL	
	avg	std	avg	std	avg	std	avg	std
1 Toms Routenplanung	253,4	51,1	337,5	61,4	591,2	132,1	969,7	98,8
2 Janes Email	259,6	27,1	317,7	48,0	670,2	64,1	1113,7	93,5
3 Barts Anweisung	253,7	39,2	318,4	65,6	3043,5	233,7	1754,0	191,4
4 SmartLab Simple	280,3	53,5	NA	NA	758,8	745,3	1400,7	146,1
5 SmartLabX 1 Doc	333,7	54,1	380,4	60,1	C	C	C	C
6 SmartLabX 2 Docs	803,1	498,1	622,7	81,7	C	C	C	C
7 SmartLabX 3 Docs	2667,3	2163,1	758,3	101,2	C	C	C	C
8 SmartLabX 4 Docs	4510,0	3966,9	629,3	91,6	C	C	C	C

Tabelle 5.5: Durchschnittliche Laufzeit (avg) und Standardabweichung (std) der Planer bei der Lösung der Beispielszenarien (alle Angaben in Millisekunden, NA = keine Lösung, C = nach 20.000 ms abgebrochen)

Tabelle 5.5 dargestellt. Sie erlauben einen Vergleich der Planer anhand der verschiedenen Szenarien.

Die Auswertung der Experimente ergibt, dass sich die Ergebnisse zwischen den Planern deutlich unterscheiden. Im Unterschied zu den Experimenten im letzten Abschnitt zeigt LPG deutlich bessere Leistungen. Blackbox war nicht in der Lage Lösungen für die Probleme 2-8 zu finden und wurde daher nicht in die Tabelle aufgenommen.[11] SGP blieb mäßig erfolgreich und war nicht in der Lage, einen Lauf der Probleme 5-8 innerhalb der gegebenen 20.000 ms zu lösen. Auch MIPS-XXL konnte aufgrund der im Vergleich sehr langen Laufzeiten nicht überzeugen. Zwar gab MIPS-XXL, wie auch SGP, für jeden Lauf der ersten vier Probleme eine korrekte Lösung zurück, die Laufzeiten waren jedoch zu langsam.

Allein die vom LPG angebotenen Algorithmen konnten bei diesen Experimenten überzeugen. Wobei LPG alle und FF sieben von acht Problemen lösen konnte. Im Anschluss wird daher das Verhalten dieser beiden Planer weiter analysiert.

Es kann zusammengefasst werden, dass die ersten vier Szenarien hinsichtlich der Laufzeit keine Herausforderung für LPG oder FF darstellten. Grund dafür ist die geringe Komplexität der Probleme. Die beteiligten Dienste werden meist nur einmalig verwendet, wenngleich das wiederholte Verwenden eines Operators für einen Planungsalgorithmus herausfordernder ist. Dennoch war FF nicht in der Lage das einfache SmartLab-Problem zu lösen. Der Planer liefert als Rückgabe sogar: best **first** space empty! problem proven unsolvable . Diese Rückgabe ist nachweislich falsch, denn das Problem konnte von den anderen Planern richtig gelöst werden. Die Ursache für diese Falschmeldung kann nicht weiter eingegrenzt werden, scheint aber nicht durch die Komplexität des Problems beeinflusst zu sein.

[11] Die Ursache dafür ist, dass blackbox nicht in der Lage ist, vererbte Typen zu verarbeiten (siehe Abschnitt 5.2.2).

Bei den SmartLabX-Problemen erhöht sich die Komplexität. Hier wird das wiederholte Ausführen eines Operators benötigt, um zu einer Lösung zu gelangen. Dabei zeigt sich, dass die Probleme mit steigender Anzahl darzustellender Dokumente schwerer werden. Die Standardabweichung der Laufzeiten von LPG nimmt bei den Problemen 5, 6, 7 und 8 stark zu. Während die Laufzeiten bei Problem 5 (268 ms Minimum und 504 ms Maximum) noch relativ nah beieinander sind, reichen sie bei Problem 6 schon von 277 ms bis 2630 ms, bei Problem 7 gehen sie von 625 ms bis 12342 ms und von 620 ms bis 18302 ms bei Problem 8. Entsprechend hoch wird auch die Standardabweichung (siehe Tabelle 5.5).
Zwar steigen auch für FF die maximalen Laufzeiten der Probleme 5, 6, 7 und 8 mit 639, 858, 1051 und 993 ms im Gegensatz zu den ersten vier Problemen an. Allerdings längst nicht so stark wie bei LPG. Auch die Standardabweichung von FF bleibt vergleichsweise niedrig.
Die Ursache für die unterschiedlichen Laufzeiten bei der Lösung ein und desselben Problems sind nicht-deterministische Komponenten in Implementierungen der Planer. Um genauere Aussagen über das Verhalten der Planer zu treffen, kann eine Analyse der Verteilung der Planungszeiten hilfreich sein. Dazu wurden approximierte Dichtefunktionen der Laufzeiten (vgl. Abbildung 5.9) erstellt. Die Dichtefunktionen aller Planer für alle gelösten Probleme befinden sich im Anhang E.
Eine erste Vermutung, dass die Laufzeiten normalverteilt vorliegen, kann durch die Dichtefunktionen nicht vollständig bestätigt werden. Zwar weisen alle Diagramme die für eine Normalverteilung typische Glockenform auf, jedoch besitzen sie in fast allen Fällen ein schiefes Moment und sind daher unsymmetrisch. Die Verteilungen sind fast durchgehend linkssteil, d.h. der Median der Durchführungen ist kleiner als der Mittelwert. Dennoch ähneln sich die Dichtefunktionen prinzipiell. Eine genauere Untersuchung, welche Verteilung vorliegt, konnte nicht mehr geleistet werden. Das Wissen um die Verteilung erlaubt eine wesentlich trennschärfere Identifizierung von Ausreißern, was u.U. zu besseren Ergebnissen der im nächsten Abschnitt beschriebenen Neustartstrategie führen könnte.
Es wird deutlich, dass sich die Erkenntnisse aus dem letzten Abschnitt bei der Überprüfung der Beispielszenarien in diesem Abschnitt wiederfinden. Jedoch wären die im letzten Abschnitt formulierten Hypothesen allein aufgrund der Ergebnisse der Beispielszenarien nicht eindeutig zu überprüfen gewesen, sodass festgehalten werden kann, dass eine Durchführung der umfangreichen Experimente mit generierten Problemen hilfreich war.
Daneben ist aufgrund der Experimente zu sehen, dass nicht nur die Laufzeit der Planer ein wichtiges Kriterium für den Einsatz als Kompositionsmethode ist. Planer differieren zum Teil erheblich in ihrer Funktionalität und Umset-

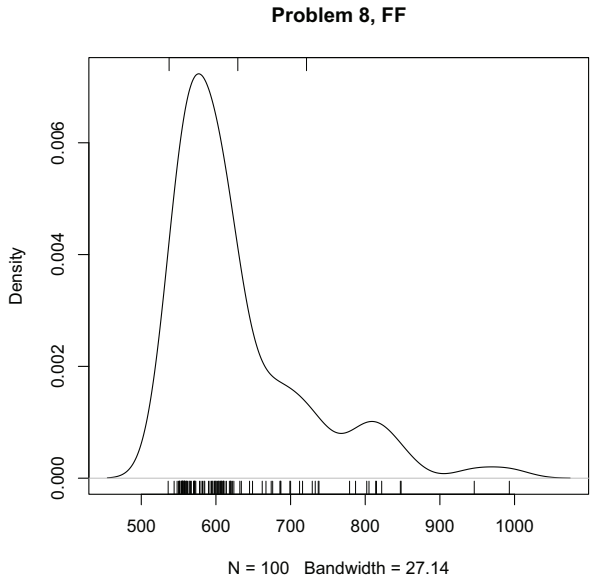

Abbildung 5.9: Verteilung der Laufzeiten von FF für das Problem 8. Die Bandbreite (Bandwidth) zur Glättung der Kennlinie ist automatisch generiert. Am unteren Rand des Diagramms sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std abgetragen. Alle 100 Läufe pro Problem konnten dabei von FF gelöst werden und ein Plan wurde erzeugt.

zung. Dieser Problematik wird sich im Abschnitt 5.2 gewidmet.

These 9 *Die Eignung von Planer gleiche Probleme zu lösen ist unterschiedlich. Daher ist die Verwendung mehrerer Planer sinnvoll.*

5.1.5 Strategien zur Verringerung der Laufzeit

Motivation für die Laufzeitevaluierung der verschiedenen Planer war es zu überprüfen, ob aktuelle Planer für den Einsatz in intelligenten Umgebungen geeignet sind. Eine spezielle Anforderung intelligenter Umgebungen ist die geforderte Unaufdringlichkeit. Die Experimente zeigten, dass teilweise sehr starke Peaks auftreten, deren Laufzeit um ein Vielfaches höher war als die vergleichbarer Probleme. Um die Unaufdringlichkeit der intelligenten Umgebung nicht zu gefährden, sollen im Anschluss mehrere Strategien überprüft werden, die es ermöglichen die Laufzeit von Planern positiv zu beeinflussen.
Um die Strategien zu evaluieren, wurden die Laufzeitexperimente aus Abschnitt 5.1.3 als Grundlage genommen.

5.1.5.1 Schwellwertmethode

Eine direkte Möglichkeit die Unaufdringlichkeit zu gewährleisten, ist eine Begrenzung der Laufzeit der Planer, wie sie zum Beispiel auch von Chan u. a. (2007) und Howe und Dahlman (2002) gefordert wird.
Wird ein Planungslauf abgebrochen, wird kein Plan erzeugt. Dabei besteht die Gefahr, eine positive Lösung (einen Plan), die der Planer möglicherweise kurz nach dem Abbruch gefunden hätte, zu verwerfen. In den Experimenten konnte gezeigt werden, dass Planungsläufe mit vergleichsweise hohen Laufzeiten dazu tendieren, einen Plan zu beinhalten, wodurch der Abbruch eines Peaks ein erhöhtes Risiko auf den Verlust einer Lösung darstellt. Dennoch ist die Wahrscheinlichkeit von Peaks insgesamt so gering (vgl. Tabelle 5.3), dass ein Abbruch von Planungsläufen trotz allem eine gangbare Möglichkeit zur Beschränkung der Laufzeit des Planungsprozesses sein könnte, ohne zu große Verluste bei den Lösungen zu verursachen. Neben der durch einen Schwellwert gewonnenen Beschränkung der Laufzeit, was für die Unaufdringlichkeit der Umgebung bedeutsam ist, stellen die durch die verminderte Laufzeit gesparten Ressourcen der Geräte, einen weiteren positiven Effekt dar, der in intelligenten Umgebungen mit vielen mobilen, batterieabhängigen Geräten essentiell und für stationäre Geräte aufgrund der Stromersparnis anzustreben ist.

In ihrer Studie untersuchten Howe und Dahlman (2002) die Frage, ob Planungsabbrüche aus dem oben beschrieben Grund der verpassten Lösungen unfair sind. Dabei kamen sie zu dem Ergebnis, dass ein vorzeitiger Abbruch der Planungsläufe bei 15 Minuten anstatt 30 Minuten keine signifikanten Änderungen der Ergebnisse nach sich zieht. Allerdings ist der gesetzte Schwellwert von 15 bzw. 30 Minuten sehr hoch. Im Gegensatz dazu liegt die im Rahmen dieser Arbeit erforderliche maximale Laufzeit bei nur einer Sekunde.
Im Folgenden soll die Anwendung verschiedener Schwellwerte zum Abbruch von Planungsläufen auf Grundlage der Daten aus den Laufzeitexperimenten überprüft werden. Es werden lediglich die Probleme mit hoher Lösungswahrscheinlichkeit betrachtet (mit n <50 und r <10), da die Problemregionen ohne Lösung auch so gut wie keine Peaks aufwiesen. Probleme, die aufgrund des Experimentaufbaus nach 2000 ms automatisch abgebrochen wurden, sind nicht in die Auswertung eingeflossen.[12]
Die Diagramme in Abbildung 5.10 zeigen die Veränderungen bezüglich der eingesparten Laufzeit sowie bezüglich der gefundenen Lösungen für jeden der drei untersuchten Planer. Da SGP in den Experimenten der langsamste Planer war, verliert er bei niedrigen Schwellwerten die meisten Lösungen. Dagegen beeinflusst selbst ein Schwellwert von 100 ms die Anzahl der gefundenen Lösungen bei blackbox und LPG kaum.

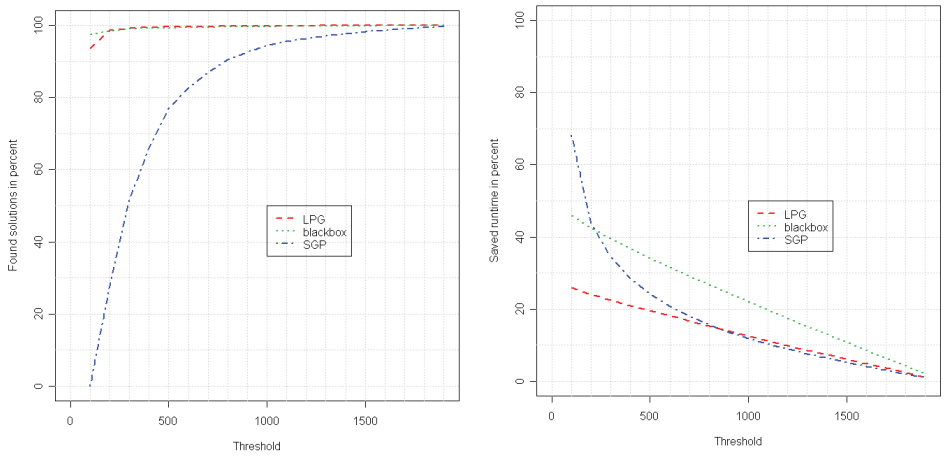

Abbildung 5.10: Kennlinien zur Darstellung des Verhältnisses der gefundenen Lösungen (found solutions) bzw. der eingesparten Laufzeit (saved runtime) in Abhängigkeit von verschiedenen Schwellwerten (thresholds)

Es kann offensichtlich ein Schwellwert gefunden werden, bei dem der Abbruch

[12] In diesem Fall stellen die 2000 ms bereits einen Schwellwert dar und es kann keine Aussage darüber getroffen werden, ob die Planer nach diesen 2000 ms noch eine Lösung gefunden hätten.

Problem / Planer	LPG	FF	SGP	MIPS-XXL
1 Toms Routenplanung	100	100	99	70
2 Janes Email	100	100	100	1
3 Barts Anweisung	100	100	0	0
4 SmartLab Simple	100	0	95	0
5 SmartLabX 1 Doc	100	100	0	0
6 SmartLabX 2 Docs	71	100	0	0
7 SmartLabX 3 Docs	24	97	0	0
8 SmartLabX 4 Docs	16	100	0	0

Tabelle 5.6: Anzahl der erfolgreichen Planungsläufe (von 100 Läufen pro Problem) für die Planer LPG, FF, SGP und MIPS-XXL bei Verwendung eines Schwellwertes von 1000 ms basierend auf den Experimenten mit den Beispielszenarien

von Planungsläufen insgesamt betrachtet eine signifikante Laufzeitersparnis verspricht ohne zu viele Lösungen zu verlieren. So bringt die Verwendung eines Schwellwertes von 300 ms ca. 23% Laufzeitersparnis für LPG und ca. 40% Ersparnis für blackbox, während bei beiden nur etwa 1% der Lösungen verloren gehen. Für SGP übersteigt die Anzahl der noch gefundenen Lösungen erst bei ca. 800 ms die Markte von 90%, was noch einer Laufzeitersparnis von ca. 15% entspricht.

Ein Schwellwert kann sowohl statisch als auch adaptiv festgelegt werden. Kann eine Heuristik für einen passenden Schwellwert gefunden werden, ist es möglich, signifikant Laufzeit zu sparen. Allerdings ist diese Heuristik vom Planer, vom Problem und von der Leistungsfähigkeit des Gerätes, auf dem der Planer läuft, abhängig und kann so nicht a priori erstellt werden.

Bei Betrachtung der Experimente mit den Beispielszenarien, zeichnet sich bei Anwendung der Schwellwertemthode ein differenziertes Bild (siehe Tabelle 5.6). Während FF mit Ausnahme der falsch negativen Lösungen zu Problem 4 in nahezu allen Läufen die Probleme ausreichend schnell richtig löst, würde eine angewandte Schwellwertmethode bei der Verwendung von SGP und MIPS-XXL sehr oft zu Abbrüchen führen. LPG kann alle Läufe mit den ersten 5 Problemen innerhalb von 1000 ms richtig lösen, anschließend häufen sich jedoch die Abbrüche bis bei Problem 8 nur noch 16 Läufe unterhalb einer Sekunde liegen.

Die Schwellwertstrategie beschränkt die Laufzeit der Planer lediglich. Sie ist daher nicht geeignet, die Planung schneller oder erfolgreicher zu machen. Wesentlicher Vorteil der Schwellwertmethode ist die Gewährleistung einer maximalen Laufzeit, was für die geforderte Unaufdringlichkeit der intelligenten Umgebung ein Gewinn ist.

5.1.5.2 Austauschstrategie

In Hinblick auf eine mögliche Strategie, Planer während der Synthese des Prozessmodells auszutauschen, wurde untersucht, ob eine solche Strategie sinnvoll sein kann. Für die anschließende Evaluierung wurden die Daten der Experimente aus Abschnitt 5.1.3 verwendet. Daher können auch nur die Laufzeiten von blackbox, LPG und SGP betrachtet werden.

Eine Austauschstrategie kann sich als Round-Robin-Verfahren mit nur einem Durchlauf vorgestellt werden. Dabei befinden sich die verfügbaren Planer in der Warteschlange. Existiert eine entsprechende Heuristik,[13] kann die Ordnung der Planer priorisiert sein. Weiterhin muss die Länge des Zeitraums a festgelegt werden, die jedem Planer, zur Lösung des Problems zur Verfügung gestellt wird. Auch hierbei sind adaptive Zeiten denkbar. Es soll zunächst von einem statischen Wert für a ausgegangen werden. In der Warteschlange befinden sich blackbox und LPG (in dieser Reihenfolge, da blackbox in den Experimenten deutlich schneller war). SGP wird nicht betrachtet, da er nie eine schnellste Lösung erzielen konnte (siehe Tabelle 5.2). Mit dieser Ordnung wird für die Strategie natürlich eine Heuristik verwendet, die sich auf die Erfahrungen aus den Experimenten stützt. Werden unbekannte Planer auf unterschiedlich rechenstarken Geräten verwendet, muss diese Heuristik mit entsprechenden Verfahren angelernt werden.

Damit die Strategie bei zwei Planern insgesamt innerhalb der eingangs geforderten 1000 ms terminiert, wird a nicht größer als 500 ms gewählt.

Interessant ist jetzt, ob eine solche Austauschstrategie gegenüber einer Strategie, die nur einen Planer, in diesem Fall blackbox laufen lässt, in Summe mehr Lösungen und zusätzliche Laufzeiteinsparungen mit sich bringt.

Dazu müssen zunächst die Fälle betrachtet werden, in denen die Laufzeit von blackbox $t(blackbox)$ die Zeitspanne a überschreitet. Die Mächtigkeit dieser Menge kann mit

$$|t(blackbox) > a| = c_{blackbox}$$

und die Anzahl der in dieser Menge gefundenen Lösungen mit

$$|res(t(blackbox) > a)| = res_{blackbox}$$

ausgedrückt werden. Sollte die Laufzeit von blackbox a überschreiten, wird dessen Planung abgebrochen und das Problem dem zweiten Planer, LPG, zur Lösung gegeben. Die Menge der Läufe, in denen LPG dann innerhalb von a

[13] Eine solche globale Heuristik konnte in den Experimenten nicht gefunden werden, ist jedoch vorstellbar.

terminiert, lässt sich mit

$$|t(LPG|t(blackbox) > a) < a| = c_{LPG}$$

sowie die hierbei gefundenen Lösungen mit

$$|res((LPG|t(blackbox) > a) < a)| = res_{Austausch}$$

beschreiben. Um eine statistische Aussage über den Erfolg bzw. Misserfolg der Strategie treffen zu können, ist weiterhin die insgesamte Laufzeit des ersten Planers, wenn er nicht nach a abgebrochen worden wäre, von Interesse:

$$\sum t(blackbox|t(blackbox) > a) = sum_{blackbox}$$

Die Summe der zusätzlichen Laufzeiten, die durch den zweiten Planer angefallen sind, ergeben sich aus:

$$\sum t((LPG|t(blackbox) > a)|t(LPG) < a) = sum_{LPG}$$

Die gesamte Laufzeit der Austauschstrategie lässt sich dann mit folgender Formel berechnen:

$$c_{blackbox} * a + sum_{LPG} + (c_{blackbox} - c_{LPG}) * a = sum_{Austausch}$$

Die Differenz aus diesem Wert und den alleinigen Laufzeiten des ersten Planers ohne Abbruch nach a ergibt sich so aus:

$$sum_{blackbox} - c_{blackbox} * a + sum_{LPG} + (c_{blackbox} - c_{LPG}) * a = t_{spared}$$

Entscheidend für die Güte der Strategie ist wie schon bei der Schwellwertmethode das Verhältnis der verpassten zu den gewonnenen Lösungen. Mit der Formel

$$(res_{Austausch} - res_{blackbox})/res_{blackbox} = \delta_{sol}$$

kann dargestellt werden, wie viele Lösungen die Austauschmethode im Verhältnis zur Singlemethode erzielt hat.[14]

Am Beispiel von $a = 500$ ms, welches in der Tabelle grau hinterlegt ist, soll kurz erläutert werden, wie die einzelnen Werte berechnet wurden.

Bei $a = 500$ ms überschreitet blackbox 10344 mal die Grenze ($c_{blackbox}$) woraufhin LPG in 4992 von diesen Fällen (c_{LPG}) im Anschluss eine Lösung innerhalb

[14] Hier ist lediglich der gesamte Saldo der gefundenen Lösungen angegeben. Es ist durchaus möglich wenn auch äußerst unwahrscheinlich, dass vereinzelt Lösungen, die blackbox noch gefunden hätte, von LPG im Anschluss nicht gefunden wurden.

a	50 ms	80 ms	100 ms	200 ms	300 ms	400 ms	500 ms
$c_{blackbox}$	33133	18663	16786	12978	11558	10904	10344
c_{LPG}	0	4384	7049	6075	5544	5248	4992
$sum_{blackbox}$	21307 s	20348 s	20172 s	19652 s	19306 s	19078 s	18826 s
sum_{LPG}	0 s	329 s	596 s	593 s	581 s	568 s	566 s
$sum_{Austausch}$	3313 s	2964 s	3248 s	4569 s	5852 s	7192 s	8414 s
t_{spared}	84,5%	85,4%	83,9%	76,8%	69,7%	62,3%	55,3%
$res_{blackbox}$	8913	5323	4660	2872	2045	1659	1332
$res_{Austausch}$	0	2375	5596	5883	5418	5159	4928
δ_{sol}	-100%	-55%	20%	105%	165%	211%	270%

Tabelle 5.7: Evaluierung der Austauschstrategie

von 500 ms findet. Diese 10344 Läufe bilden daher die Grundgesamtheit der betrachteten Durchführungen für $a = 500$ ms.
Aufaddiert ergibt die Laufzeit der 10344 Läufe von blackbox eine Dauer von 18826 Sekunden $sum_{blackbox}$.[15] Die kumulierte Laufzeit von LPG für die 4992 Läufe beträgt 566 Sekunden (sum_{LPG}). In den übrigen Läufen überschreitet auch LPG den Grenzwert von 500 ms. So ergibt sich eine Gesamtlaufzeit von $10344 * a + 566s + (10344 - 4992) * a = 8414s$ ($sum_{Austausch}$) und eine daraus folgende Einsparung von 55,3% (t_{spared}) bei angewandter Austauschstrategie. Ohne Austausch hätte blackbox in 1332 von 10344 Fällen ($res_{blackbox}$) noch eine Lösung ermittelt. Bei angewandter Austauschstrategie werden 4928 der 10344 Läufe gelöst ($res_{Austausch}$) und dabei 270% mehr Lösungen (δ_{sol}) bei einer Einsparung von über 55% Laufzeit gefunden.
Die Tabelle 5.7 zeigt die Ergebnisse für weitere Werte für a. Es ist zu beachten, dass jede Spalte dabei von einer unterschiedlichen Grundmenge (1. Zeile) ausgeht.
Für größere Zeiträume a von über 100 ms lässt sich eine signifikante Steigerung der Anzahl der gefundenen Lösungen bei gleichzeitiger Verringerung der insgesamten Laufzeiten beobachten. Die Austauschstrategie kann so einen erheblichen Beitrag zur Verringerung der Laufzeiten der Planer und dadurch zur Gewährleistung der Unaufdringlichkeit der Umgebung leisten.
Aus den Ergebnissen in Tabelle 5.7 ist auch zu erkennen, dass die Strategie für Werte von a <80 ms schlechtere Ergebnisse als eine alleinige Lösung mit blackbox liefert. Grund dafür sind die in Abschnitt 5.1.3.3 ermittelten Offsets der Planer (18 ms für blackbox und 75 ms für LPG). LPG liefert aufgrund seiner Implementierung erst nach frühestens 75 ms Ergebnisse, während zu diesem Zeitpunkt blackbox schon sehr viel Zeit zur effektiven Berechnung des Problems hatte. Wird LPG früh abgebrochen (z.B. bei $a = 50$ ms) sind daher

[15] Es ist zu beachten, dass die maximale Laufzeit der Experimente auf 2000 ms begrenzt war, so dass hier ohne eine Beschränkung von noch höheren Werten ausgegangen werden kann, und der genannte Wert eine untere Grenze darstellt.

von diesem Planer keine Ergebnisse zu erwarten. Nach Überschreitung des Offsets kann LPG jedoch mit blackbox konkurrieren und ist in nicht wenigen Fällen dann schneller als dieser. Es ist daher anzunehmen, dass mit steigender Anzahl von Geräten in einer intelligenten Umgebung, die zu höherer Komplexität der Probleme und absolut höheren Laufzeiten der Planer führen, auch der Einsatz einer Austauschstrategie vorteilhafter wird.
Es soll noch erwähnt werden, dass die Gesamtzahl der ausgewerteten Läufe bei über 5.5 Millionen lag, so dass sich angesichts der betrachteten gut 10.000 Läufe die absolute Einsparung der Strategie stark relativiert. Bei der Untersuchung der Austauschstrategie wurden zudem keinerlei Betrachtungen des Overheads vorgenommen. So ist beispielsweise der Aufruf eines zweiten Planers mit zusätzlichen Latenzen verbunden. Bei der durchgeführten Evaluierung stand die Überprüfung der prinzipiellen Tauglichkeit der Strategie im Mittelpunkt.
Die hier präsentierten Ergebnisse basieren auf einer umfangreichen Evaluierung. Daher ist eine direkte Anwendung in einer intelligenten Umgebung nicht ohne Weiteres möglich. Die Erkenntnis, dass eine Austauschstrategie die Offsets der Planer berücksichtigen muss, ermöglicht es jedoch, eine einfache adaptive Strategie zur Bestimmung jeweils individueller, von Planer zu Planer unterschiedlicher, Werte für a zu formulieren. Den Offset eines unbekannten Planers zu bestimmen ist trivial. Dem Planer sollte, wie in Abschnitt 5.1.3.3 beschrieben, ein einfach lösbares und ein einfach unlösbares Problem zur Planung übergeben werden. Die Laufzeit dieser Läufe sollte annähernd gleich sein und entspricht dem Offset (*offset*) des Planers. Eine Austauschstrategie muss dann für jeden Planer ein $a = \mathit{offset} + \epsilon$ verwenden. Die Ordnung der Planer könnte aufsteigend nach *offset* gewählt werden. Mit entsprechenden, lernenden Verfahren könnte die Ordnung sowie ϵ zur Laufzeit der intelligenten Umgebung verfeinert werden.

These 10 *Die Umsetzung einer Austauschstrategie kann sowohl zur Steigerung der Anzahl der gefundenen Pläne als auch zur Verringerung der insgesamten Planungsdauer führen.*

5.1.5.3 Neustartstrategie

Diese Strategie wird anhand der Daten, die mit den Beispielszenarien gewonnen wurden, evaluiert. Hierbei werden immer nur die Laufzeiten eines einzelnen Planers betrachtet. Es zeigte sich, dass sich diese bei der Lösung desselben Problems sehr stark unterscheiden können (siehe Abschnitt 5.1.4). Diese Vola-

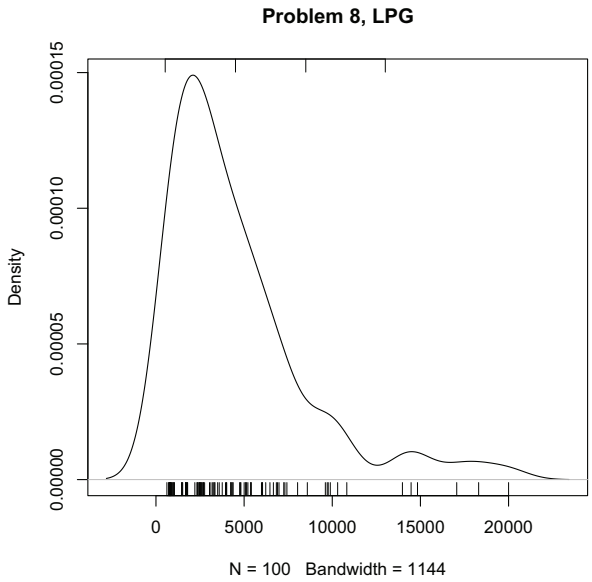

Abbildung 5.11: Verteilung der Laufzeiten von LPG für das Problem 8. Die Bandbreite (bandwidth) zur Glättung der Kennlinie ist automatisch generiert. Am unteren Rand des Diagramms sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std abgetragen. Alle 100 Läufe pro Problem konnten dabei von LPG gelöst werden und ein Plan wurde erzeugt.

tilität der Planungslaufzeiten liegt die Idee der Neustartstrategie zu Grunde. Ein Abbruch und anschließendes Neustarten von Suchalgorithmen hat sich in der Vergangenheit bereits als vorteilhaft gezeigt (Richter und Westphal, 2008).

Um die Neustartstrategie zu bewerten, sollen sich die Laufzeiten eines Planers aus den Experimenten in Abschnitt 5.1.4 anhand der sich ergebenden Dichtefunktion verdeutlicht werden. Abbildung 5.11 zeigt die Dichtefunktion von LPG bei der Lösung von Problem 8.

An den an der X-Achse abgetragenen, diskreten Werten der einzelnen Laufzeiten ist zu erkennen, dass es Ausreißer gibt. Daneben ist aber auch zu erkennen, dass der Großteil der Läufe deutlich schneller war. Eine möglicherweise erfolgversprechende Strategie, um das Ziel der Unaufdringlichkeit zu erreichen, könnte also sein, einen langen Planungslauf abzubrechen, um ihn im Anschluss in der Hoffnung auf eine kürzere Laufzeit noch einmal zu starten.

Unterschiedliche Laufzeiten für gleiche Probleme können in jedem Fall bei nicht-deterministischen Planern auftreten. Möglicherweise können aber auch

deterministische Planer diese Laufzeitunterschiede zeigen. In Howe u. a. (1999) wurde die starke Sensibilität der Planer gegenüber einfachen syntaktischen Änderungen (z.B. Umsortierung der Operatoren in der Domänenbeschreibung oder die Umsortierung der Propositionen der PEs) beschrieben, die sich für die Anwendung der Strategie auch bei deterministischen Planern zunutze gemacht werden könnte. Dazu müsste ein Planungsproblem nach einem Abbruch durch eine entsprechende Komponente entsprechend umsortiert werden, bevor es dem Planer wieder zur Lösung übergeben wird. An der Untersuchung von Howe u. a. (1999) ist allerdings zu kritisieren, dass keine Untersuchung der Planer auf nicht-deterministische Elemente durchgeführt wurde. Ein nicht-deterministischer Planer wird in verschiedenen Läufen sehr wahrscheinlich unterschiedliche Laufzeiten für ein gleiches Problem haben, auch ohne syntaktische Änderungen des Problems. Kommen beide Effekte (Nicht-Determinismus des Planers und syntaktische Änderungen des Problems) zusammen, kann keine Aussage mehr darüber getroffen werden, ob die syntaktischen Änderungen zu den unterschiedlichen Laufzeiten geführt haben oder sich der Laufzeitunterschied allein durch den Nicht-Determinismus des Planers erklären ließe.

Wesentlich für eine Neustartstrategie ist zunächst ein gut gewählter Zeitpunkt für einen Abbruch. Der Zeitpunkt eines Abbruchs sollte Ausreißer von normalen Läufen separieren. Es muss also eine Definition der minimalen Laufzeit eines Ausreißers gefunden werden. Orientierung dafür geben Mittelwert (μ) und Varianz (σ). Bei einer normalverteilten Zufallsvariablen liegen knapp 70% der Durchführungen im Intervall $\mu \pm \sigma$ und bereits über 95% im Intervall $\mu \pm 2\sigma$. Alle Werte jenseits des jeweiligen Intervalls sind als Ausreißer zu betrachten. Da keine passende Verteilungsfunktion für die Ergebnisse gefunden werden konnte, soll zunächst vom Intervall $\mu + \sigma$ als Abgrenzung zwischen normalen Läufen und Ausreißern ausgegangen werden, auch wenn die experimentell ermittelten Dichtefunktionen (siehe Anhang E) deutliche Abweichungen zur Normalverteilung zeigen. Eine Neustartstrategie würde, vorausgesetzt die Dichtefunktion samt Mittelwert und Standardabweichung wären bekannt,[16] nachdem ein Planungslauf die Dauer von $\mu + \sigma$ überschritten hätte, diesen abbrechen, ggf. umsortieren und dann sofort wieder neu starten. Als grobe Orientierung soll angenommen werden, dass der neue Lauf dann im Mittel eine Zeit von σ benötigt, um zu einem neuen Ergebnis zu kommen. Die insgesamte Laufzeit einer Neustartstrategie wäre damit $\sigma + \mu + \sigma = 2\sigma + \mu$. Die Strategie ist insgesamt erfolgversprechend, wenn ausreichend viele Ausreißer

[16] Da intelligente Umgebungen auch in unbekannten Konstellationen funktionieren sollen, ist die Anforderung nach bekannter Verteilungsfunktion der Planungszeiten eines Planers kritisch zu sehen. Diese statistischen Daten sind erst nach vielen Läufen verfügbar. Dennoch lässt sich an den Diagrammen erkennen, dass sich die Dichtefunktionen für alle Planer ähneln.

jenseits der Schwelle $2\sigma + \mu$ liegen, da sich für sie die insgesamte Planungszeit verkürzt.
Um den zu erwartenden Erfolg einer Neustartstrategie abzuschätzen, wurden vier statistisch signifikante Werte in folgender Reihenfolge als Striche am oberen Rand des Diagramms 5.11 sowie der Diagramme im Anhang E abgetragen:

1. $\sigma - \mu$ - Untere Grenze für Ausreisser[17]
2. σ - Mittelwert der Durchführungen
3. $\sigma + \mu$ - Grenze die Ausreißer von normalen Läufen trennt
4. $2\sigma + \mu$ - Grenze ab der ein Neustart von Läufen insgesamt Laufzeit spart

Es ist zu erkennen, dass der vierte Wert $2\sigma + \mu$ in vielen Diagrammen nicht mehr im Wertebereich liegt und daher nur drei Striche eingezeichnet wurden. In diesen Fällen würde die Abbruch-Neustart Strategie keinen Aussreisser identifizieren und nicht aktiv werden. In Fällen, in denen der vierte Wert zu sehen ist (z.B. bei den Problemen 6, 7 und 8 für LPG), ist zu erkennen, dass nur wenige Laufzeiten jenseits der Schwelle $2\sigma + \mu$ liegen und mehr Läufe zwischen $\sigma + \mu$ und $2\sigma + \mu$ zu finden sind. Wird ein zwischen den Grenzen $\sigma + \mu$ und $2\sigma + \mu$ liegender Lauf abgebrochen, ist die Neustartstrategie jedoch unvorteilhaft, da ein Weiterlaufen des Planers im Mittel früher zu einem Ergebnis geführt hätte, als der Neustart. Auch eine Erhöhung der Schwelle zur Definition eines Ausreißers auf $\mu + 2\sigma$ würde für die vorliegenden Verteilungen zu keiner Verbesserung der Ergebnisse der Strategie führen.
Die Anwendung der vorgestellten Neustartstrategie für die Beispielszenarien ergibt daher nur in wenigen Fällen eine Reduktion der insgesamten Laufzeit. Die Anzahl der Läufe, die sich aufgrund der Anwendung der Strategie verlängern, ist im Vergleich dazu höher, sodass eine Anwendung der Strategie zunächst nicht vorteilhaft erscheint.
Für eine lohnende Verwendung der Neustartstrategie müssten die Laufzeiten der Ausreißer sehr viel größer sein und wenige Läufe zwischen Grenzen $\sigma + \mu$ und $2\sigma + \mu$ liegen. Die Dichtefunktion einer solchen Verteilung würde im hinteren Bereich ein zweites deutliches lokales Maximum aufweisen.

5.1.5.4 Race

Die Umsetzung dieser Strategie setzt, wie auch die Austauschstrategie mehrere in der Umgebung verfügbare Planer voraus. Zur Lösung eines Problems

[17] Da die untere Grenze der Laufzeiten der Planer durch deren Offset (siehe Abschnitt 5.1.3.3) bestimmt ist, ist dieser Wert ohne praktische Relevanz und wird nur zur besseren Orientierung eingezeichnet.

werden mehrere oder alle verfügbaren Planer gleichzeitig gestartet. Auf diese Art und Weise sind schnellste Ergebnisse möglich. Allerdings ist zu beachten, dass die Planer aus verschiedenen Gründen falsch negative Ergebnisse zurückgeben. Daher sollte eine Race-Strategie nicht mit der ersten Rückmeldung eines Planer abgebrochen werden, denn dies könnte ein falsch negatives Ergebnis [18] sein, sondern erst mit dem ersten Plan. Dann allerdings ist die Race-Strategie bei Problemen, die keine Lösung haben, immer so langsam, wie der langsamste teilnehmende Planer. Um die Gefahr von falsch negativen Lösungen zu minimieren, jedoch auch die Laufzeit zu beschränken, kann ein negatives Ergebnis als wahr angenommen werden, wenn es schon von einigen aber nicht allen Planern als nicht lösbar identifiziert wurde.

Bei Betrachtung der Experimente mit den Beispielszenarien würde eine Race-Strategie immer zum Erfolg führen, wenn mindestens drei Ergebnisse abgewartet werden, bis eine negative Lösung akzeptiert wird, da blackbox in den Problemen 2 bis 8 und FF beim Problem 4 sehr schnell ein falsch negatives Ergebnis lieferten. Die Antwortzeit der Strategie entspräche der des jeweils schnellsten Planers, der eine Lösung zurückgibt, zu dem Preis, dass bei jedem Problem fünf Planer arbeiten und Rechenleistung beanspruchen. Die Race-Strategie ist damit schnell aber sehr ressourcenintensiv.

Daneben kann die Verwendung der Race-Strategie zur initialen Generierung einer Heuristik für die weitere Verwendung der Planer im Rahmen einer Austauschstrategie verwendet werden. Auf diese Art und Weise können schnell die Leistungsmerkmale der in der Umgebung befindlichen Planer gesammelt werden.

Neben den Vorteilen hinsichtlich Laufzeit und der potentiellen Steigerung der Anzahl der Lösungen haben sowohl die Race- als auch die Austauschstrategie noch einen entscheidenden architektonischen Vorteil. Intelligente Umgebungen wurden als dynamisch charakterisiert. Die Verfügbarkeit von Geräten der Umgebung kann sich jederzeit ändern. Werden im Rahmen der Race- oder der Austauschstrategie mehrere in der Umgebung verteilte Planer verwendet, bleibt das Verhalten des Composers auch dann robust, wenn einer der verwendeten Planer während der Planung ausfällt. In diesem Fall verstreicht zwar bei der Austauschstrategie die Zeitspanne a des Planers ungenutzt. Dennoch können die verbleibenden Planer weiter an der Lösung des Problems beteiligt werden. Voraussetzung dafür ist eine asynchrone Implementierung der Archi-

[18] In den Ergebnissen der Experimente mit den generierten Problemen finden sich 3948 Fälle, in denen blackbox schneller als LPG war, jedoch eine negative Lösung zurückgab und LPG im Anschluss noch einen validen Plan gefunden hat. In den Experimenten mit den Beispielszenarien gab blackbox aufgrund der fehlerhaften Implementierung in den Problemen 2-8 immer sehr schnell (im Bereich von 70 ms) ein negatives Ergebnis zurück, jeweils bevor die anderen Planer trotzdem einen validen Plan lieferten.

tektur, die im Abschnitt 7.4 näher erläutert wird.

5.2 Funktionale Eigenschaften

Neben der Effizienz der Algorithmen sind auch deren Implementierung und die funktionalen Eigenschaften der Planer wesentliche Einflussfaktoren für die Anwendbarkeit in intelligenten Umgebungen. PDDL unterstützt eine große Anzahl an Funktionalitäten. Jedoch ist kaum ein Planer in der Lage, die gesamte Bandbreite der Funktionalitäten von PDDL anzubieten. In den Beispielszenarien in Abschnitt 4.1 wurde sich bei der Modellierung der Domänen und Probleme auf die Elemente von PDDL1.2 beschränkt. Bereits für diese einfachen Domänen zeigte sich (siehe Abschnitt 5.1.4), dass nicht alle Planer funktional in der Lage waren, die Probleme zu lösen. Um die Evaluierung der Planer zu vervollständigen, werden daher in diesem Abschnitt einige Tests beschrieben, mit deren Hilfe Planer auf ihre Funktionalität getestet werden können.

5.2.1 Anforderungen an Planer

Mit dem Element :requirements verfügt PDDL bereits über eine gute Möglichkeit die Anforderungen einer Domäne an einen Planer zu beschreiben. In der Praxis sollen Planer so in die Lage versetzt werden, bereits im Vorfeld der eigentlichen Planung überprüfen zu können, ob sie in der Lage sind, Probleme dieser Domäne zu lösen. In der im letzten Abschnitt bereits erwähnten Studie von Howe und Dahlman (2002) wurde auch untersucht, inwieweit verschiedene Planer die Anforderungen (:requirements) von PDDL unterstützen. Hier stellte sich heraus, dass bis auf eine Ausnahme kein Planer alle getesteten Anforderungen[19] von PDDL unterstützte. Jede Anforderung bringt eigene syntaktische Elemente in eine Domänen- bzw. Problembeschreibung ein. Eine Überprüfung, ob ein Planer für eine Domäne geeignet ist, ließe sich somit auch auf Parser- bzw. Lexerebene durchführen. Die explizite Nennung und Auswertung der Anforderungen über :requirements ist jedoch eleganter.

Die Eigenschaften von Planern, die bei der IPC teilgenommen haben und teilnehmen, lassen sich in den begleitenden Veröffentlichungen der jeweiligen IPC meist in übersichtlicher, tabellarischer Form finden. Die aktuellste Tabelle ist unter der Adresse http://ipc.informatik.uni-freiburg.de/Planners verfügbar.

[19] Es wurden die Anforderungen :derived−predicates, :conditional−effects, :disjunctive−preconditions, :equality, :existential−preconditions, :safety−constraints, :strips, :typing und :universal−preconditions aus der PDDL-Version 1.2 untersucht. Diese sind bis auf :safety−constraints auch Bestandteil der Versionen 2.2 und deren Nachfolger.

5.2.2 Fähigkeitstests

Bei der Modellierung der Szenarien stellte sich heraus, dass zunächst zwei Anforderungen von PDDL erforderlich sind, :strips und :typing. Wenngleich :strips obligatorisch ist, soll es dennoch erwähnt werden. Als weitere wichtige, wenn auch für die beschriebenen Szenarien nicht zwingend erforderliche Anforderung erscheint :derived−predicates (siehe Abschnitt 4.2.6).
Darüber hinaus können weitere funktionale Anforderungen formuliert werden, die einige Planer nicht erfüllen und dennoch wesentlich sind, um die Szenarien zu modellieren. Die Anforderungen lassen sich durch gezielt modellierte Testprobleme überprüfen, indem die Rückgaben eines Planers mit erwarteten Lösungen verglichen werden. Entspricht die Rückgabe eines Planers nicht der erwarteten Lösung, unterstützt der Planer die entsprechende Fähigkeit nicht. Eine Sammlung dieser Tests findet sich im Anhang B.
Es ist zu beachten, dass im Folgenden die Fähigkeiten einzelner Planer überprüft werden. Diese Fähigkeitstests haben nur bedingten Bezug zu den Modellierungsrichtlinien in Abschnitt 4.2.9. Dort wurden Richtlinien formuliert, die eine verteilte aber konsistente Modellierung von Planungsproblemen ermöglichen sollen. In diesem Abschnitt werden die Planer selbst auf konsistenten Umgang mit spezifischen syntaktischen und semantischen Mustern in Planungsproblemen überprüft.
Zusätzlich zu den bisher untersuchten Planern werden in diesem Abschnitt auch die Fähigkeiten von LAMA untersucht. LAMA ist ein sequentieller, heuristischer Planer und Gewinner der letzten IPC (Richter und Westphal, 2008).

Nicht instanziierte Typen Wird in Ad-hoc-Umgebungen Dienstekomposition angewendet, muss eine Planungsdomäne, die die jeweils aktuelle Umgebung beschreibt, zur Laufzeit aus den verfügbaren Dienstbeschreibungen generiert werden (siehe Abschnitt 7.2). Im Anschluss wird basierend auf den Zielen der Intentionsanalyse eine Problembeschreibung generiert. Dabei besteht die Möglichkeit, dass nicht alle Typen, die durch die Operatoren in der Typenhierarchie der Domäne hinzugefügt wurden, auch in der Problembeschreibung verwendet werden. Eine solche Situation könnte insbesondere dann auftreten, wenn Dienstebeschreibungen unabhängig von Intentionen gespeichert werden, um eine Domänenbeschreibung in einer statischen Umgebung für verschiedene Probleme wiederverwenden zu können. So können Prädikate und Operatoren existieren, die Typen verwenden, die bei der Instanziierung der Objekte in der Problembeschreibung nicht verwendet werden. Diese nicht instanziierten Typen sind für die Lösung des Planungsproblems irrelevant und sollten keinen negativen Einfluss auf die Planung haben und den Planer daher nicht stören. Ein entsprechendes Testproblem, mit dem ein Planer auf die Akzeptanz

nicht instanziierter Typen überprüft werden kann, findet sich im Anhang in Abschnitt B.1.

Vererbung von Typen In den Definitionen von PDDL ist ausdrücklich die Möglichkeit der Vererbung von Typen vorgesehen, was die Beschreibung einer Typentaxonomie innerhalb der Domänenbeschreibung erlaubt. Dadurch wird es möglich, generische Operatoren wie beispielsweise (switchOn ?d − device) zu verwenden, die für alle Objekte anwendbar sind, deren Obertyp device ist. Generische Operatoren helfen die Domänenbeschreibung lesbar und damit wartbar zu halten. Das Planungsproblem B.2 veranschaulicht die Vererbung von Typen.

Objekte in Domänenbeschreibung Die Verwendung von Objekten in der Beschreibung der Planungsdomäne widerspricht der Idee der Unabhängigkeit der Domänen- von der Problembeschreibung und damit der Trennung von t-box und a-box.
Um den Suchraum des Planungsproblems zu beschränken, kann es dennoch sinnvoll sein, Operatoren zu definieren, deren Vorbedingungen oder Effekte in den Pradikaten auch Objekte verwenden. Ein Testproblem, das Objekte in Operatorbeschreibungen verwendet, findet sich im Anhang in Abschnitt B.3.

Partiell geordnete Pläne In den vorangegangen Kapiteln zeigte sich schon mehrfach der Wunsch, partiell geordnete Pläne als Lösung zu erhalten. Im Anhang B.4 ist ein einfaches Problem hinterlegt, in dem je zwei unabhängige Operatoren verwendet werden müssen, um das Ziel des Planungsproblems zu lösen. Ein paralleler (oder partial-order) Planer ist in der Lage diese Unabhängigkeit der Operatoren zu erkennen und einen Plan zurückzugeben, der parallel ausführbare Aktionen enthält (POP). Sequentielle (oder total-order) Planer sind dazu nicht in der Lage und geben immer eine reine Sequenz (TOP) als Lösung zurück.

5.2.3 Auswertung der Tests

Die Ergebnisse der Durchführung der oben beschriebenen Tests mit den Planern sind in Tabelle 5.8 dargestellt.
Anhand der Ergebnisse in Tabelle 5.8 ist zu erkennen, dass kein Planer allen funktionalen Anforderungen entspricht. So ist blackbox, trotz sehr guter Ergebnisse im Laufzeitverhalten, durch seine fehlende Unterstützung für Typenvererbung für die Dienstekomposition in intelligenten Umgebungen praktisch nicht anwendbar. In Kombination mit den Ergebnissen der Laufzeituntersuchung schneiden LPG und FF am besten ab. Da FF als spezielle Konfiguration

Problem/Planer	LPG	FF	blackbox	SGP	LAMA	MIPS-XXL
PDDL-Anforderungen						
(:strips)	OK	OK	OK	OK	OK	OK
(:typing)	OK	OK	OK	OK	OK	OK
(:derived-predicates)	OK	OK	-	-	OK	-
Funktionale Eigenschaften						
Leere Typen	-	-	OK	OK	OK	OK
Vererbung von Typen	OK	OK	-	OK	OK	OK
Objekte in Domäne	OK	OK	OK	OK	OK	OK
POP	OK	OK	OK	OK	-	-

Tabelle 5.8: Fähigkeiten der Planer

von LPG verwendet wurde, sind die funktionalen Eigenschaften von LPG und FF hier gleich. Beide sind im Gegensatz zu allen anderen überprüften Planern nicht in der Lage, mit leeren Typen umzugehen und geben eine Fehlermeldung zurück, falls in einer Domäne leere Typen enthalten sind. Für LAMA konnten leider keine Laufzeitexperimente durchgeführt werden.[20] Funktional hat LAMA den Nachteil, lediglich sequentielle Pläne zu generieren. Der Planer kann aber ansonsten durch großen Funktionsumfang punkten. SGP und MIPS-XXL zeigten in unseren Experimenten schlechtes Laufzeitverhalten und können durch fehlende Unterstützung abgeleiteter Prädikate keinen Boden gutmachen. Für die Lösung von Problemen aus den überprüften Szenarien sind sie daher schlechter geeignet.

5.2.4 Erfahrungen aus den Experimenten

Alle Planer stellten sich als robust heraus, es gab während der gesamten Experimente (auch aus Abschnitt 5.1) keine abnormalen Abstürze eines Planers, was für die sehr gute Qualität der Implementierungen spricht. Dennoch gab es Probleme.

Es war für die Planer LPG und blackbox nicht möglich, Domänen mit mehr als 341 Operatoren zu verarbeiten. In einem solchen Fall wurde ein Syntaxfehler von den Planern zurückgegeben. Eine wahrscheinliche Ursache für dieses Verhalten, ist die gemeinsame Verwendung eines fehlerhaften Parsers.

In sehr seltenen Fällen lieferte blackbox nachweislich falsche Ergebnisse, was höchstwahrscheinlich auf einen fehlerhaften Nachverarbeitungsschritt zurückzuführen ist. Um falsche Komposition zu vermeiden, sollten Pläne daher im

[20] Die Laufzeitexperimente wurden unter Windows durchgeführt. Leider war es nicht möglich LAMA, der unter Linux entwickelt wurde, vollständig für Windows zu compilieren und so automatisierte Experimente zu ermöglichen. Erste händische Versuche unter Linux zeigten jedoch sehr schnelle Planungszeiten von LAMA, sodass eine weitere Untersuchung von LAMA zu empfehlen ist.

Anschluss an die Planung immer validiert werden. Ein Validierungsalgorithmus ist dabei simpel. Gegeben den initialen Zustand der Welt wendet der Algorithmus schrittweise die Operatoren des Plans an und vergleicht das Ergebnis mit dem verlangten Zielzustand. Sind beide nicht gleich, ist der Plan falsch. Die Komplexität dieses Evaluierungsalgorithmus ist linear zur Länge des resultierenden Planes und daher nicht problematisch im Vergleich zum vorhergegangenen Planungslauf.
Es muss weiterhin festgestellt werden, dass die unterschiedliche Implementierung der Planer und deren teilweise unterschiedliche Interpretation der Sprache PDDL eine einheitliche Modellierung erschwert. Es kommt immer wieder vor, dass einige Planer Domänen und Probleme, ohne spezifische Fehlermeldungen zu geben, nicht parsen. Die im letzten Abschnitt vorgestellten Tests können Hinweise geben, warum Planer bestimmte Probleme nicht parsen, nicht lösen oder Fehler zurückgeben und sind damit ein hilfreicher Beitrag für die Modellierung von Planungsdomänen aus dem Bereich der Dienstekomposition in intelligenten Umgebungen. Des Weiteren sind die Testprobleme Grundlage für eine automatische problemspezifische Planerauswahl.
Planer sind in verschiedensten Sprachen auf verschiedenen Plattformen unter Zuhilfenahme verschiedener Werkzeuge entwickelt worden. Was eine gemeinsame, integrierte Verwendung erschwert. Es ist nötig, für jeden Planer einen eigenen Wrapper zu schreiben (siehe Abschnitt 7.3). Dabei ist es mühselig und stellenweise nicht möglich, Planer auf andere Plattformen zu portieren, da u.a. Binärdateien verwendet werden, die nicht als Quellen mitgeliefert werden.
Da die Syntax von Plänen nicht durch PDDL geregelt ist, müssen auch die Rückgaben der Planer individuell geparst werden. Zwar hat sich die Heterogenität der Planer bezüglich Compilierung, Aufruf und Rückgabe durch die Anforderungen der IPC verbessert, sodass für alle neueren Planer der Quellcode frei verfügbar ist und die compilierten Versionen mit einem einheitlichen Shell-Script unter Linux aufrufbar sind. Allerdings gibt es noch immer starke Unterschiede.
Weiterhin können viele Planer unterschiedlich parametrisiert werden. Dabei gibt es verschiedenste Parameter, die erheblichen Einfluss auf das Laufzeitverhalten eines Planers haben können (Howe und Dahlman, 2002). Die Parameter sind hauptsächlich vom intern verwendeten Algorithmus abhängig und lassen zum Beispiel eine Kontrolle über die Anzahl von Iterationen oder die Rekursionstiefe von Suchen zu oder spezifizieren die maximal zu erzeugende Größe des Suchgraphen bei GRAPHPLAN-basierten Planern. Auf Grund ihrer spezifischen Wirkung ist eine allgemeine für verschiedene Planer gültige und problembezogene Justierung der Parameter nicht möglich. Während der im Kapitel 5 beschriebenen Experimente wurden alle Planer mit ihren Standardeinstellungen verwendet. Einzige Ausnahme ist hier die Neukonfigu-

ration von LPG, um eine alleinige Verwendung des enthaltenen Planers FF zu erzwingen.

These 11 *Die funktionalen Eigenschaften verfügbarer Planer unterscheiden sich zum Teil stark. Für eine automatisierte Komposition ist daher eine funktionale Überprüfung der verwendeten Planer zu empfehlen.*

Kapitel 6

Integration KI-planungsbasierter Komposition

6.1 Grundlegende Anforderungen

Nachdem in den letzten Kapiteln die Anwendbarkeit von klassischer KI-Planung zur Dienstekomposition in intelligenten Umgebungen hinsichtlich der Problemmodellierung und der Effizienz untersucht wurde, widmet sich dieses Kapitel den Möglichkeiten der Integration KI-planungsbasierter Komposition in intelligenten Umgebungen.
Wie im zweiten Kapitel beschrieben, setzt sich eine Dienstekomposition aus der Generierung des Prozessmodells und dessen anschließender Instanziierung zusammen. Die Generierung des Prozessmodells übernimmt dabei ein *Planer*. Bei der Anwendung reiner KI-Planung wird ein Planer durch einen *Controller* und stellenweise durch einen *Scheduler* ergänzt. Diese Komponenten übernehmen die Instanziierung des Prozessmodells (siehe Abbildung 6.1).
Während der Planer die kausalen Aktionsketten ermittelt, ist es Aufgabe des Controllers, diese Aktionen auszuführen. Ein Scheduler kann verwendet werden, um die Aktionen vor der Ausführung auf vorhandene Ressourcen zu verteilen. Während die Verwendung von Planer und Controller auch in intelligenten Umgebungen unzweifelhaft nötig ist, stellt sich die Frage, inwieweit die Verwendung eines Schedulers für die Dienstekomposition in intelligenten Umgebungen sinnvoll ist. Die Beispielszenarien Toms Routenplanung, Janes Email und Barts Anweisung können als Probleme angesehen werden, bei denen allein kausale Aktionsketten gesucht werden. Eine anschließende Verteilung der Pläne auf Ressourcen und somit die Verwendung eines Schedulers scheint zunächst nicht erforderlich, da die verwendeten Operatoren direkt einzelnen Dienstaufrufen zugeordnet werden konnten.

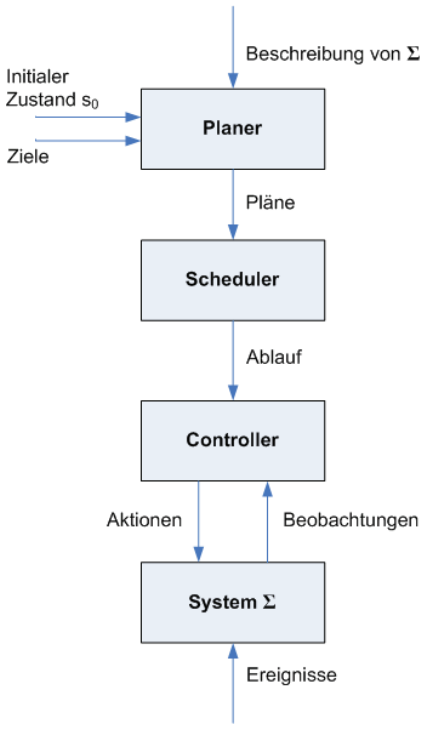

Abbildung 6.1: Konzeptionelles Modell der KI-Planung (nach Nau (2007))

Im Beispiel Modellierung der SmartLab-Szenarios wurde zunächst die Darstel-

lung eines Dokumentes auf einer Leinwand modelliert. Hier wird ebenfalls eine kausale Aktionsverkettung gesucht. In erweiterten Versionen dieses Problems (siehe Anhang A.5) wurden Pläne für die gleichzeitige Darstellung von bis zu vier verschiedenen Dokumenten gesucht. Neben der Suche nach kausalen Aktionsketten zur Anzeige der Dokumente, beinhalten diese Probleme auch einen Teil Ressourcenmanagement. Allerdings wurde auch hier kein zusätzlicher Scheduler benötigt, da es in diesem Fall möglich war, dessen Aufgaben direkt in das Planungsproblem zu integrieren und so allein von einem Planer lösbar zu machen. Der Einsatz eines Schedulers erscheint erst dann sinnvoll, wenn Diensttypen und nicht konkrete Dienstinstanzen komponiert werden, was in den Beispielszenarien nicht der Fall war.[1]

Die von einem Planer erzeugten Pläne sind allerdings in keinem Fall direkt ausführbar, sodass es in jedem Fall erforderlich ist, einen Controller zu verwenden, der Pläne in Dienstaufrufe umwandelt. Da diese Komponente immer im Anschluss an die eigentliche Planung Verwendung findet, können darin bei Bedarf auch Funktionalitäten eines Schedulers untergebracht werden.

Im Rahmen dieser Arbeit wird sich auf die Generierung des Prozessmodells konzentriert. Daher werden im Anschluss die architektonischen Implikationen der Planung untersucht, ohne jedoch die weiteren Schritte (Scheduler und Controller) aus den Augen zu verlieren.

Viele Architekturentwürfe zur Komposition von Diensten aus der Literatur spiegeln die Zweiteilung in Generierung des Prozessmodells und dessen anschließender Instanziierung wieder. So verwenden Chakraborty u. a. (2005) (vgl. auch Chakraborty u. a. (2002)) zur Umsetzung einer mobilen Dienstekomposition in pervasiven Umgebungen eine Layerstruktur, in der eine eindeutige architektonische Trennung zwischen der Erstellung des Prozessmodells (service composition layer) und der Ausführung der Komposition (service execution layer) beschrieben ist.

Auch bei der Beschreibung von Agarwal u. a. (2008), bei der ausdrücklich KI-Planung als Kompositionsmethode zugrunde gelegt wird, kommt die Trennung zwischen Generierung des Prozessmodells und dessen Instanziierung zum Ausdruck. Agarwal u. a. (2008) differenzieren weiter in *überlappende*, *monolithische*, *abgestufte* und *templatebasierte* Kompositionen. Dabei unterscheiden sie zwischen den Mengen von Diensttypen, angemeldeten Dienstinstanzen und real laufenden Diensten, wobei die Menge der real laufenden Dienste eine echte Teilmenge der angemeldeten Dienstinstanzen ist. Agarwal u. a. (2008) versuchen durch die Unterscheidung der beiden letzten Mengen, das mögliche spontane Ausfallen von Diensten abbildbar zu machen. Da in den Szenarien

[1] Ein Beispiel für die getrennte Komposition von Diensttypen und Dienstinstanzen geben Agarwal u. a. (2008) mit der „abgestuften Komposition“ (siehe unten).

im Kapitel 4 eine Unterscheidung zwischen der Komposition von Diensten und Diensttypen nicht gemacht wurde, sind daraus entstehende Unterscheidungen der Architektur im Rahmen dieser Arbeit zu vernachlässigen.
Bei *überlappenden* Kompositionsmethoden sind nach Agarwal u. a. (2008) die Generierung des Prozessmodells, die Instanziierung und die Ausführung der Dienste untrennbar miteinander verwoben und werden zur Laufzeit ausgeführt. Hierbei kann nachträglich nicht in die Generierung des Prozessmodells oder der Aktionssequenz eingegriffen werden. Die überlappende Komposition ruft direkt eine Sequenz von real laufenden Diensten auf. Diese Methode eignet sich für den Einsatz in Domänen mit einer großen Anzahl von Unsicherheiten, die erst zur Ausführungszeit determiniert werden können und ist somit vor allem für Online-Planer geeignet. Theoretisch kann in einer überlappenden Komposition jedoch auch ein Offline-Planer zur Generierung des Prozessmodells verwendet werden (siehe hierzu auch Abschnitt 6.2.2).
Bei der *monolithischen* Komposition ist die Erstellung des Prozessmodells von der Ausführung der Komposition entkoppelt, wobei das Prozessmodell „offline“ also vor Ausführung der Komposition erstellt wird. Bei dieser Art der Komposition besteht die Möglichkeit, mehrere Instanzen eines Prozessmodells zu erstellen, zwischen denen erst bei der Ausführung ausgewählt wird.
Bei der *abgestuften* Komposition unterscheiden Agarwal u. a. (2008) im Vergleich zur monolithischen Komposition zusätzlich in eine logische und eine physikalische Komposition. Die logische Komposition arbeitet nur auf Grundlage der Diensttypen. Bei der pysikalischen Komposition wird der bei der vorangegangenen logischen Komposition erzeugte Plan auf die im Repository publizierten Dienste angewendet. Im letzten Schritt wird dieser Plan dann mit den real verfügbaren Diensten ausgeführt. Die abgestufte Komposition entspricht der klassischen Einsatzweise von Planern mit an die Planung angeschlossenem Scheduling (siehe Abbildung 6.1). Bei Veränderungen der Umgebung kann der erzeugte Plan unter Umständen wiederverwendet werden, was eine Neugenerierung des Prozessmodells erspart.
Mit der *templatebasierten* Komposition werden Methoden bezeichnet, bei denen generalisierte Vorlagen von Prozessmodellen verwendet werden, die anhand von Regeln ausgesucht und instanziiert werden. Hier sind große Teile des Prozessmodells vorgegeben und müssen so nicht neu generiert werden. Diese Art der Komposition kann durch HTN-Planung (siehe Abschnitt 3.1.3) umgesetzt werden.
Zunächst kommen alle von Agarwal u. a. (2008) beschriebenen Methoden für die Komposition von Diensten in intelligenten Umgebungen in Frage. Ein klassischer KI-Planer kann insbesondere bei der monolithischen und der abgestuften Komposition als Kompositionsmethode verwendet werden. Eine überlappende Komposition scheint vor allem für Online-Planer geeignet zu sein.

Template-basierte Komposition kommt für HTN-Planer in Frage, soll aber hier nicht weiter betrachtet werden, da in diesem Fall aktionsbasierte Ziele (siehe Abschnitt 3.2.3) zu verwenden wären. Wenn bei der Komposition nicht zwischen Diensttyp und Dienst unterschieden wird, gleicht die monolithische der abgestuften Komposition, sodass die im Rahmen dieser Arbeit verwendete Methode einer monolithischen bzw. abgestuften Komposition entspricht.

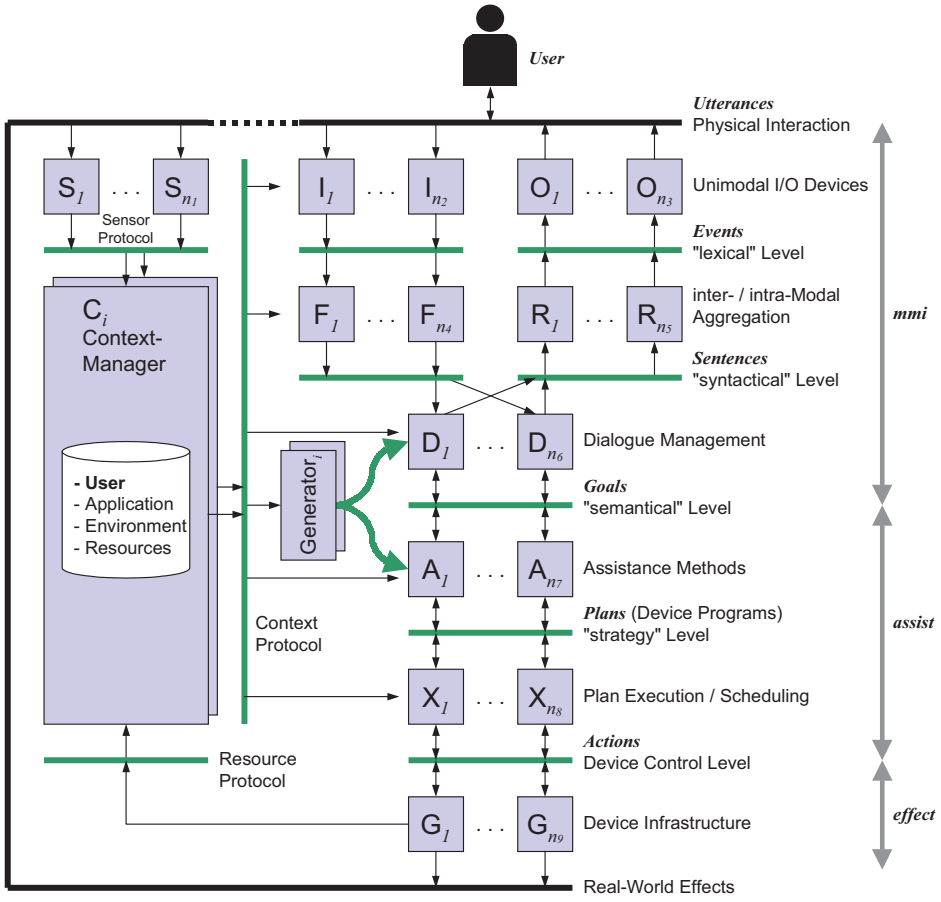

Abbildung 6.2: Konzept der EMBASSI-Architektur (aus Heider (2009))

In der Architektur des Projektes EMBASSI (Herfet und Kirste, 2002; Heider, 2009) ist die beschriebene Zweiteilung in Generierung des Prozessmodells und dessen anschließender Instanziierung ebenfalls wiederzufinden. Wesentliche Vorarbeiten für eine Architektur zur Nutzerassistenz in intelligenten Umgebungen stammen aus dem Projekt EMBASSI. Die Architektur des EMBASSI Projektes wurde im GRK MuSAMA, in dessen Rahmen diese Arbeit entstanden ist, durch verschiedene Arbeiten aufgegriffen und erweitert.
Der EMBASSI-Architektur (siehe Abbildung 6.2) liegt eine kanalbasierte Konzeption zugrunde, welche aus mehreren Schichten besteht. Diese Schichten un-

terteilen sich in drei Gruppen (Multimodale Interaktion („mmi“), Assistenz („assist“) und Effekte („effect“)).
Die Gruppe der Assistenz umfasst dabei die zwei Schichten Assistenzmethoden („Assistance Methods“) und Planausführung („Plan Execution“).
Eine Komponente der Plangenerierung wird in der EMBASSI-Architektur als „A“-Komponente bezeichnet. Die Eingaben dieser Komponente sind Ziele sowie Informationen des Kontext-Managers. Ausgaben der Komponente sind Pläne. Die Kommunikation zwischen den Schichten wird in Abbildung 6.2 bidirektional beschrieben, was die Möglichkeit von Rückfragen zwischen den Komponenten benachbarter Schichten offen lässt. Generell lässt sich der Workflow jedoch als sequentiell betrachten (vgl. Abbildung 1.3). Für die Ausführung der Dienste sind „X“-Komponenten verantwortlich. Sie formen die Pläne in Aktionssequenzen um und führen diese auf den Geräten aus.
Es kann zusammengefasst werden, dass die im Rahmen dieser Arbeit beschriebene Dienstekomposition für intelligente Umgebung in jeder Architektur zum Einsatz kommen kann, die eine seperate Generierung und Ausführung des Prozessmodells unterstützt. Das trifft sowohl für die durch Agarwal u. a. (2008) allgemein beschriebene monolithische bzw. abgestufte Komposition als auch für die konkreten Entwürfe von Chakraborty u. a. (2005) und Heider (2009) zu.

6.2 Entwurf eines Composers

Es stellt sich die Frage, wie eine entsprechende Komponente zur Anwendung klassischer KI-Planung als Kompositionsmethode in intelligenten Umgebungen umzusetzen ist. Sie soll als *Composer* bezeichnet werden.
Grundlegende Eingaben der Komponente sind die Ziele eines Nutzers, Dienstbeschreibungen der Umgebung sowie Sensordaten (vgl. Abbildungen 3.1). Die Ausgabe ist ein Plan. Der Composer ist damit für Durchführung des eigentlichen Planungsprozesses, den Datenaustausch mit der intelligenten Umgebung sowie die Transformation der Daten verantwortlich. In der EMBASSI-Architektur sind die Empfänger der von „A“-Komponenten generierten Plänen die „X“-Komponenten. Auch im konzeptionellen Modell von Nau (2007) schließt sich der Controller, verantwortlich für die Planausführung, an die Plangenerierung an.
Weitere Anforderungen an die Integration ergeben sich aus den in Abschnitt 1.5 genannten Punkten. Die Komposition soll in Ad-hoc-Umgebungen verwendet werden, d.h. robust gegenüber Änderungen der Umgebung sein. Der Ablauf der Komposition soll automatisch ablaufen, was teilweise schon durch die Verwendung von Planung gewährleistet ist. Die Komposition sollte weiter-

hin ressourcenschonend erfolgen. Daraus folgt, dass im Gegensatz zum konzeptionellen Modell von Nau (2007) (siehe Abbildung 6.1) die Beschreibung der Umgebung in Form von Dienstbeschreibungen aus der Umgebung selbst kommt. Abbildung 6.3 zeigt eine angepasste Variante der Konzeption aus Abbildung 6.1.

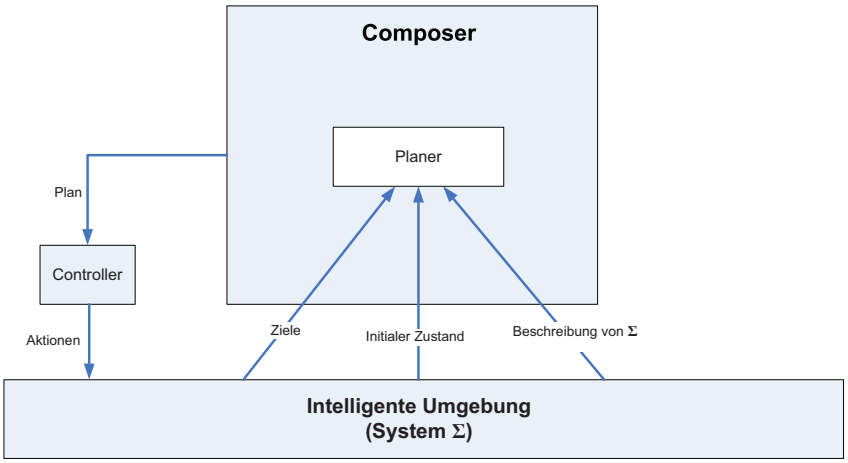

Abbildung 6.3: Konzept eines Composers, Schritt I

Um das in Abschnit 3.2 beschriebene Mapping umzusetzen (vgl. Abbildung 3.1 aus Abschnitt 3.2) ist eine Erweiterungen des Konzeptes um eine Komponente erforderlich, die aus den verschiedenen Eingaben ein Planungsproblem erstellen kann. Diese Komponente wird Assembler [2] genannt. Sie erstellt aus den Eingaben des Composers (siehe oben) zwei PDDL-Dokumente, die Domänenbeschreibung und die Problembeschreibung, die dem Planer als Eingabe dienen. Abbildung 6.4 veranschaulicht einen um eine Domänenassemblierung erweiterten Composer.
Um das Konzept des Composers zu verfeinern, sollen im Anschluss weitere Eigenschaften der Integration geklärt werden.

6.2.1 Zentral kontra dezentral

Da klassische KI-Planung als Kompositionsmethode zur Erzeugung des Prozessmodells verwendet wird, soll sich zunächst der Anwendbarkeit von Planung in verteilten Szenarien gewidmet werden. Eine Einteilung verteilter Planungsansätze gibt Durfee (2001). Die Unterteilung ist zweidimensional. Die

[2] to assemble - zusammenstellen
Es besteht hierbei kein Zusammenhang zum Begriff Assembler, der aus Programmcode Maschinencode erzeugt. Bei der hier beschriebenen Assemblierung wird aus Fragmenten ein Ganzes erstellt aber keine Übersetzung der Elemente vorgenommen.

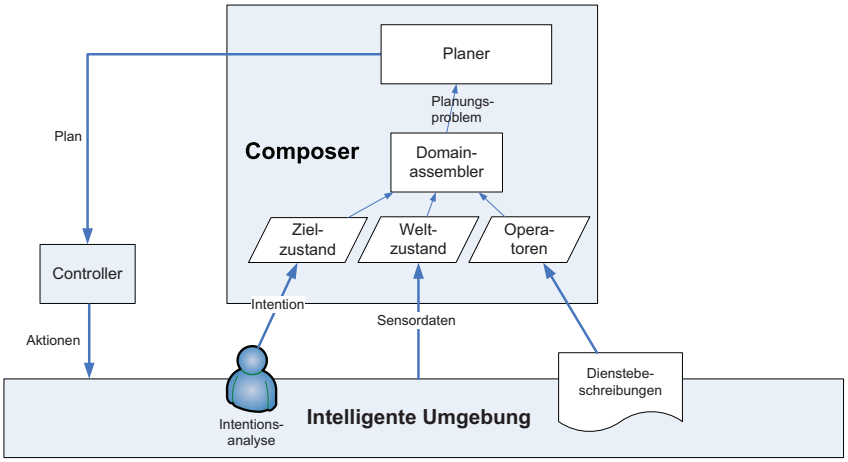

Abbildung 6.4: Konzept eines Composers, Schritt II

betrachteten Dimensionen sind auch hier die Erzeugung des Planes und dessen Ausführung. Beides kann sowohl zentral als auch dezentral erfolgen. Durfee (2001) unterscheidet neben der eigentlichen klassischen KI-Planung, die zentral einen zusammenhängenden Plan sucht, drei verschiedene Konzepte der Verteilung.

1. Zentralisiertes Planen für verteilte Pläne
   Hierbei wird von einer zentralen Planungsinstanz ein Plan erstellt, der sich in verschiedene unabhängige Teilpläne zerlegen und verteilt ausführen lässt.

2. Verteiltes Planen für zentrale Pläne
   Bei dieser Variante wird der Planungsprozess auf verschiedene Agenten verteilt. Jeder beteiligte Agent trägt einen Teil zur Lösung des Problems bei. Diese Art der Planung benötigt Mittel, um ein Planungsproblem in Teilprobleme aufzuteilen.

3. Verteiltes Planen für verteilte Pläne
   In dieser Version der verteilten Planung ist es nicht mehr nötig, einen zusammenhängenden Plan zu generieren. Verschiedene Agenten haben Wissen über verschiedene Teile des Plans, wobei sich die verteilte Ausführung der Planfragmente nicht gegenseitig behindert. Im Idealfall sind die Agenten in der Lage, sich bei der Ausführung der Teilpläne zu unterstützen.

Eine Umsetzung des ersten Konzeptes ist in intelligenten Umgebungen basierend auf klassischer KI-Planung, die die zentrale Planung übernimmt, möglich.

Neben dem Planer muss ein entsprechender Controller vorhanden sein, der die Verteilung des Planes auf die Dienste übernimmt. Eine einfache Möglichkeit der verteilten Ausführung eines Planes soll kurz beschrieben werden. Im Anschluss an die Plangenerierung werden alle Aktionsausführungen im Plan mit ihren Preconditions annotiert. Diese Informationen sind im Plan enthalten (z.B. in Form von ordering constraints) oder können, gegeben den Plan und das zugrundeliegende Planungsproblem, inferiert werden. Die Dienste bekommen im Anschluss an die Planung ihre Aktion mit den jeweiligen Vorbedingungen zugeschickt und müssen selbst überprüfen, wann diese erfüllt sind, um dann die entsprechende Aktion auszuführen. Diese Methode weist allerdings Nachteile auf. Zunächst müssen alle Geräte in der Lage sein, den Zustand der Umgebung entsprechend umfassend zu überwachen, was nicht vorausgesetzt werden kann. Des Weiteren kostet die Überwachung des Weltzustandes durch verschiedene Dienste kontinuierlich Ressourcen, was die Anfordung nach einer ressourcenschonenden Integration gefährdet.
In der Literatur zur Komposition von Webservices werden die zentrale bzw. dezentrale Ausführung eines vorhandenen Prozessmodells oft mit Orchestrierung und Choreographie bezeichnet. Beide Ansätze beruhen auf einer zentralen Generierung des Prozessmodells und charakterisieren lediglich dessen Ausführung. Bei der Orchestrierung gibt es eine zentrale Instanz, die die Ausführung der Dienste des Plans veranlasst und überwacht. Diese Komponente entspricht im Sinne der KI-Planung einem u.U. durch einen Scheduler ergänzten Controller. Bei der Choregraphie wird der erhaltene Plan an alle beteiligten Dienste geschickt. Die Ausführung des Plans wird durch wechselseitige Kommunikation der Dienste ermöglicht. Der Controller ist dabei nicht mehr als singuläre Komponente vorhanden. Seine Funktionen werden verteilt realisiert.
Aufgrund der hohen Komplexität der Interaktionsbeschreibung zwischen den Diensten und der fehlenden theoretischen Formalisierung (Wolf, 2009) ist die Umsetzung einer Choreographie im Gegensatz zur Umsetzung einer Orchestrierung anspruchsvoller. Daneben erzeugt ein auf Choreographie basierender Ansatz das für dezentrale Lösungen typische, gegenüber zentralen Lösungen deutlich erhöhte Kommunikationsaufkommen, was aufgrund der Anforderung nach ressourcenschonender Integration der Komposition kritisch zu sehen ist. Bei der Orchestrierung wiederum stellt der Controller einen „Single-Point-of-Failure“ dar. Fällt er aus, kann die Komposition nicht ausgeführt werden, bis wieder der alte oder ein neuer Controller verfügbar ist.
Beim Ausfall einzelner an der Komposition beteiligter Dienste sind beide Ansätze in der Lage dies zu erkennen und angemessen darauf zu reagieren.
Die beiden letzten Konzepte von Durfee (2001) (verteiltes Planen für zentrale Pläne und verteiltes Planen für verteilte Pläne) verteilen den eigentlichen

Planungsprozess und erfordern so andere Planungsalgorithmen als die in dieser Arbeit untersuchten klassischen KI-Planer. Beispiele für die Umsetzung solcher verteilten Lösungen geben Fauvet u. a. (2001) sowie Reisse und Kirste (2008). Wesentliche Kriterien bei der Bewertung der Tauglichkeit solcher Verfahren sind deren Effizienz hinsichtlich der Reaktionszeit und des Ressourcenverbrauches (sowohl benötige Netzwerkbandbreite und Netzwerktraffic als auch Rechenlast auf den beteiligten Agenten) sowie deren Korrektheit. Im Rahmen dieser Arbeit muss offen bleiben, ob die Tauglichkeit dieser Verfahren in intelligenten Umgebungen gegeben ist. Dahingehend wird auf die ebenfalls im GRK MuSAMA entstandene Dissertation von Plociennik (2010) verwiesen. Im Rahmen dieser Arbeit erfolgt die Generierung des Prozessmodells zentral. Die Ausführung des Plans ist sowohl zentral als auch dezentral möglich. Eine zentrale Variante der Ausführung wurde umgesetzt (siehe Kapitel 7).

6.2.2 Online kontra Offline

Online-Planung unterscheidet sich von der klassischen Planung, die in diesem Zusammenhang auch Offline-Planung genannt werden könnte, dahingehend, dass der Planungsprozess nicht der Einschränkung 8 (siehe Abschnitt 3.1.2) unterliegt. Ein Online-Planer kann während der Suche im Zustandsraum Ereignisse aus der Umgebung in den Planungsprozess einfließen lassen und so zusätzlich gezielt Informationen anfordern. Online-Planung ist in Umgebungen sinnvoll, in denen die Modellierung von Domänen mit allen möglichen Verzweigungen und die Generierung von Plänen darin aufgrund von fehlendem Wissen unmöglich oder zu umfangreich ist.
Kuter u. a. (2004) nennen zwei Punkte, die eine Verwendung von Online-Planung zur Dienstekomposition erforderlich machen.[3]

1. Unvollständiges Wissen des Planers über den aktuellen Zustand der Welt
2. Zu viele verfügbare Informationen, deren gänzliche Erfassung nicht möglich ist

Diese Anforderungen wurden in Hinblick auf Webservice-Umgebungen formuliert. Wie bereits in Abschnitt 2.4 erwähnt, wird in intelligenten Umgebungen

[3]Der von Kuter u. a. (2004) beschriebe Planer ENQUIRER ist ein echter Online-Planer. Er erzeugt keine nicht-deterministischen Pläne mit Aktionen, die zur Ausführungszeit Informationen sammeln und daraufhin entscheiden, welcher Weg im Plan zu gehen ist, sondern versucht, diese Informationen während der Planungszeit zu akquirieren. Im Gegensatz zum Offline-Planen ist bei der Verwendung eines Online-Planers eine Unterscheidung zwischen weltverändernden Diensten und informationsliefernden Diensten hilfreich, da hier eine gefahrlose Ausführung von Diensten, die während der Planung unvollständiges Wissen vervollständigen, essentiell ist (vgl. Abschnitt 3.2.2).

von weitaus weniger Diensten und Informationen ausgegangen, sodass die genannten Punkte nur bedingt zutreffen und eine Online-Planung in intelligenten Umgebungen zunächst nicht erforderlich machen. Des Weiteren wurden in der von Kuter u. a. (2004) beschriebenen Evaluierung Laufzeiten im mittleren zweistelligen Sekundenbereich benötigt, um die Testprobleme zu lösen. Auch wenn die dort verwendeten Probleme von denen einer intelligenten Umgebung abweichen, kann bei Laufzeiten dieser Länge nicht mehr von unaufdringlicher Komposition (vgl. Abschnitt 5.1) gesprochen werden.
Vorteilhaft bei der Verwendung eines Online-Planers im Vergleich zu einem Offline-Planer ist die Möglichkeit, während der Planung auf kritische Änderungen der Umgebung reagieren zu können. **Kritische Änderungen** sind Änderungen kritischer Zustände, die Einfluss auf die Validität des erzeugten Plans haben. Die Menge der kritischen Zustände eines Plans setzt sich aus den Vorbedingungen der Aktionen, die nicht Effekte der Aktionen des Plans sind, zusammen, wobei die Vorbedingungen, die ausschließlich in bereits abgeschlossenen Aktionen vorkommen, nicht mehr zu den kritischen Zuständen zählen.[4] Ein Offline-Planer muss bei einer kritischen Änderung der Umgebung eine Neuplanung[5] veranlassen, da der generierte Plan u.U. nicht mehr geeignet ist, das aktuelle Problem in der veränderten Umgebung zu lösen (vgl. Abschnitt 5.1). Diese Neuplanung kostet Zeit und Ressourcen. Dennoch ist die zusätzliche Laufzeit der Neuplanungen bei durchschnittlichen Planungszeiten im Sekundenbereich vergleichsweise gering. Im Rahmen dieser Arbeit wird von sehr schnellen Plangenerierungen im einstelligen Sekundenbereich für schwierige Probleme und im unteren dreistelligen Millisekundenbereich für einfache Probleme ausgegangen (siehe Kapitel 5), was den zeitlichen Verlust einer wiederholten Generierung des Prozessmodells bei auftretenden Änderungen der Umgebung gegenüber dem durch zusätzliche Kommunikation entstehenden Zeitaufwand einer Online-Planung gering hält.
Eine architektonische Konsequenz, die sich aus der Verwendung von Offline-Planung ergibt, ist die Notwendigkeit einer Komponente, die einen Plan vor und während seiner Ausführung bei jeder Veränderung der Umgebung validieren kann. Die dafür benötigte Komponente muss zunächst in der Lage sein, kritische Änderungen der Umgebung zu erkennen. Bei einer Änderung eines kritischen Zustands[6] ist der Plan nicht mehr gültig und eine Neuplanung muss

[4] Hierbei sei darauf hingewiesen, dass viele Aktionen nicht direkt nach ihrem Aufruf abgeschlossen sind (vgl. Abschnitt 4.2.4).

[5] In der Literatur hat sich gezeigt, dass die Planreparatur verglichen mit der Neuplanung eines Problems eine gleich hohe, unter Umständen sogar höhere Komplexität hat (Nebel und Koehler, 1995), sodass eine Planreparatur hier nicht in Erwägung gezogen wird.

[6] Dieser Aussage liegt eine boolesche Betrachtung der Umgebung zugrunde. Werden kontinuierliche Zustandsvariablen verwendet, sind hier die Über- oder Unterschreitung von Schwellwerten zu überwachen.

erfolgen.
Wird diese selektive Überwachung der kritischen Zustände nicht umgesetzt, wodurch jede Zustandsänderung der Umgebung zu einer Neuplanung führt, ist es bei einer hohen Anzahl von Änderungen der Umgebung möglich, dass nie eine Komposition erstellt werden kann, da der Planer fortwährend plant.
Auch in der Architektur von Rao und Su (2004) wird zusätzlich zum Planer ein sogenannter *Evaluator* verwendet, dessen Aufgabe es jedoch vor allem ist, QoS-Anforderungen in die Komposition einfließen zu lassen.
Abbildung 6.5 zeigt ein um eine Validierungskomponente erweitertes Konzept einer Integration eines Composers in eine intelligente Umgebung.

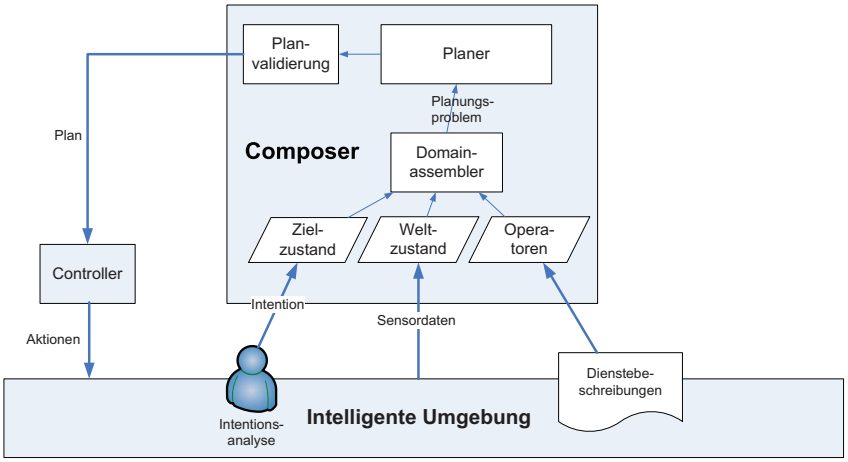

Abbildung 6.5: Konzept eines Composers, Schritt III

6.2.3 Planermanagement

Im Kapitel 5 wurde beschrieben, dass Planungsläufe Laufzeitspitzen haben können, die die Unaufdringlichkeit einer intelligenten Umgebung gefährden. Daneben wurden verschiedene Strategien vorgestellt und anhand der Experimentdaten evaluiert (siehe Abschnitt 5.1.5). Zur Umsetzung dieser Strategien wird zusätzlich zum Planer eine Managementkomponente benötigt, die die Laufzeit eines oder mehrerer Planer beobachten und die Planung bei Bedarf unterbrechen und neustarten kann. Das Management der Planung ist eine Kernaufgabe des in dieser Arbeit vorgestellten Composers.
Die im Abschnitt 5 durchgeführte Evaluierung der Planer konnte weiterhin zeigen, dass es keinen einzelnen Planer gibt, der für alle Probleme geeignet ist. Um das Ziel schnellstmöglicher Komposition zu erreichen, kann es daher von Vorteil sein, aus verschiedenen Planern auszuwählen, welcher für die aktuelle Umgebung am besten geeignet ist. Dazu ist es zunächst erforderlich,

einer intelligenten Umgebung mehrere Planer zur Verfügung zu stellen. Im Metaplaner von Howe u. a. (1999) werden die verschiedenen Planer statisch eingebunden. Zwar ist es so schneller möglich, die Planer zu starten und zu beenden, allerdings müssten alle Planer auf einem Gerät laufen. Für die Umsetzung einer Race-Strategie ist dies jedoch ungünstig.
Aus den Experimenten wurde allerdings auch ersichtlich, dass eine solche Auswahl nicht deduktiv, basierend auf a priori Wissen erfolgen kann, sondern auf einen lernenden Algorithmus angewiesen ist, unter anderem auch deshalb, weil die Leistung der Rechner auf denen die Planer in der Umgebung vorhanden sind, nicht als bekannt vorausgesetzt werden kann. Eine Möglichkeit, eine solche lernende Auswahl umzusetzen wird in Fink u. a. (1998) beschrieben. Darüber hinaus zeigten die Planer funktionale Differenzen (siehe Abschnitt 5.2), sodass neben einer effizienzbasierten Auswahl auch eine funktionalitätsbasierte Auswahl erfolgen sollte.
Neben der Überwachung und dem möglichen Abbruch von Planungsläufen fällt daher die bestmögliche Auswahl eines konkreten Planers in den Aufgabenbereich des Composers. Ein detaillierterer Überblick über die Umsetzung der Planerauswahl und der Abbruchstrategie wird im nächsten Kapitel in Abschnitt 7.4 gegeben.
Abbildung 6.6 stellt das vorläufig endgültige Konzept eines Composers zur automatischen Dienstekomposition in intelligenten Umgebungen dar. Der Composer wurde um eine intelligente Planerauswahl erweitert. Weiterhin wurden als Dienste verfügbare Planer hinzugefügt. Eine Vorgängerversion dieses Konzeptes wurde in Marquardt und Uhrmacher (2009a) veröffentlicht.

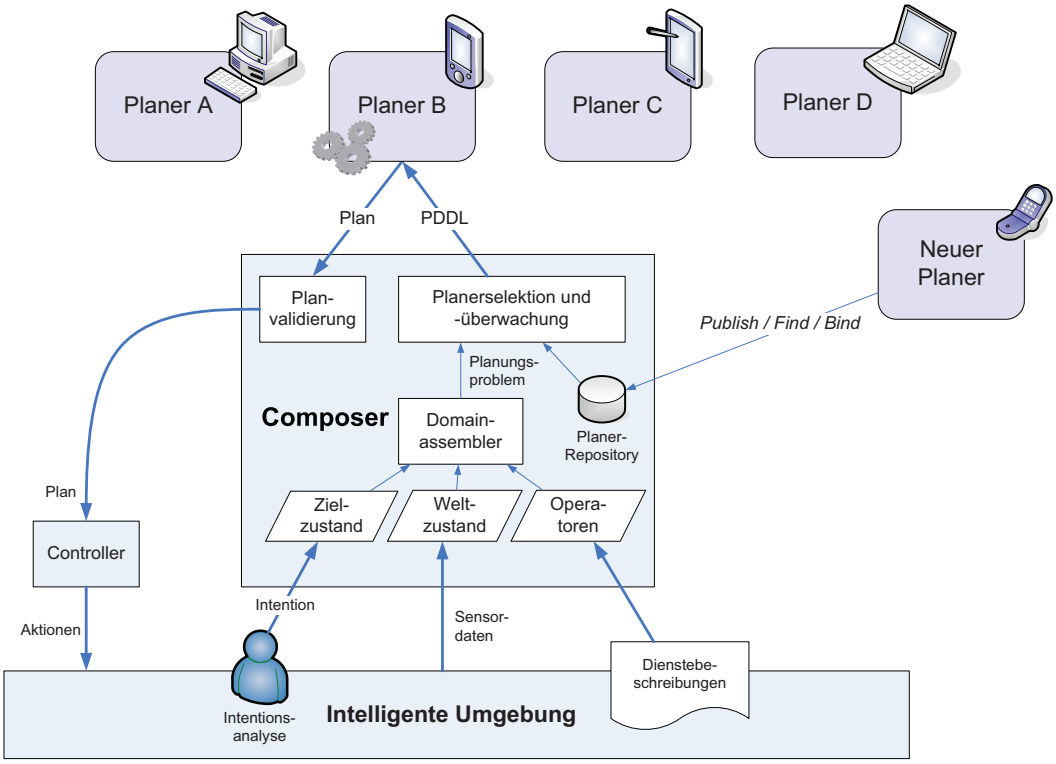

Abbildung 6.6: Konzept eines Composers, Schritt IV

6.3 Ablauf einer Komposition

Aufbauend auf der im letzten Abschnitt entworfenen Komponente zur Integration KI-planungsbasierter Komposition in intelligenten Umgebungen, soll in diesem Abschnitt der Ablauf der Erstellung eines Prozessmodells skizziert werden.
Bezugnehmend auf die im Abschnitt 2.3.1 definierten Kompositionsphasen (1. Auffinden der vorhandenen Dienste (*discovery*), 2. Generierung eines Prozessmodells (*composition*), 3. Instanziierung des Prozessmodells (*deployment*) und 4. Ausführung des komponierten Dienstes (*execution*)) setzt der im letzten Abschnitt entworfene Composer den zweiten Punkt die „Generierung eines Prozessmodells“ um. Als Ausgangssituation sind an diesem Punkt alle vorhandenen Dienste bekannt. Ziel ist es, ein Prozessmodell, im Falle der Verwendung von KI-Planung also einen Plan, zu generieren.

Assemblierung der Domäne Im ersten Schritt wird aus den vorhandenen Dienstbeschreibungen die Planungsdomäne zusammengesetzt. Dieser Schritt wird Assemblierung der Domäne genannt. Sie wird im Abschnitt 7.2.1 genauer beschrieben.

Ziel anfordern Im Anschluss kann das aktuelle Ziel der Komposition von der Intentionsanalyse abgerufen werden. Das Ziel muss dabei als Satz in einer Beschreibungslogik verfasst sein, die sich auf PDDL abbilden lässt (vgl. Abschnitt 3.2.3). Der umgesetzte Composer akzeptiert Ziele, die in PDDL-Syntax formuliert sind.

Initialen Weltzustand prüfen Dieser Schritt kann je nach Größe der Umgebung sehr umfangreich werden. Neben den aktuellen Werten der Umgebungssensoren müssen hier die gegenwärtigen Zustände aller Geräte bzw. deren Dienste abgefragt werden. Um die Menge der durchzuführenden Abfragen zu begrenzen, kann sich auf die zu diesem Zeitpunkt bekannte Menge der Propositionen beschränkt werden, die in den Vorbedingungen der Operatoren der bereits generierten Domäne oder in der Zielbeschreibung vorhanden sind. Andere Propositionen sind für das jeweilige Planungsproblem irrelevant.

Assemblierung des Planungsproblems Mit dem Wissen um den aktuellen Weltzustand, die Ziele des Nutzers und mit der bereits erstellten PDDL-Domäne lässt sich eine PDDL-Problembeschreibung erstellen. Dabei sollte überprüft werden, ob jedes der im Ziel verwendeten Prädikate entweder in der Domänenbeschreibung, im initialen Weltzustand oder in den Effekten der

Operatoren (bzw. der abgeleiteten Prädikate) enthalten ist. Ist dies nicht der Fall, kann die Komposition an dieser Stelle abgebrochen werden, da eine Lösung des Problems in der gegebenen Umgebung dann nicht möglich ist. Eine detailliertere Beschreibung der Assemblierung des Problems wird in Abschnitt 7.2.2 gegeben.

Planerauswahl Die Planerauswahl ist Teil der Metaplanerfunktionalität des Composers. Sie erfolgt problemspezifisch, einerseits hinsichtlich der Effizienz der Planer und andererseits hinsichtlich deren Fähigkeiten. Voraussetzung für eine echte Auswahl ist das Vorhandensein mehrerer Planer. Anhand der erstellten PDDL-Domäne und des PDDL-Problems lassen sich die Anforderungen des aktuellen Planungsproblems an die Fähigkeiten des Planers ableiten und können mit den Fähigkeiten der in der Umgebung vorhandenen Planern abgeglichen werden. Neben der funktionalen Auswahl kann die Planerauswahl auch basierend auf der Effizienz der vorhandenen Planer erfolgen. Im Abschnitt 5.1.5 wurden mehrere mögliche Strategien vorgestellt. Wird nur ein Planer verwendet oder eine Austauschstrategie benutzt, sollte während der Planerauswahl eine effizienzbasierte Auswahl bzw. Priorisierung der Planer erfolgen.

Planen Wurden ein oder mehrere Planer ausgewählt, wird das assemblierte Planungsproblem entsprechend der gewählten Strategie an die jeweiligen Planungsdienste geschickt und die Planung gestartet.
Um die Unaufdringlichkeit der Umgebung zu wahren, kann die Dauer eines Planungslaufes überwacht und nach Überschreitung eines Schwellwertes abgebrochen werden.
Da die Planer verteilt verfügbar sind, ist es möglich in diesem Schritt mehrere Planer zu verwenden. Die Auslagerung der Planer vermindert zudem die Komplexität des Composers und verteilt die Fehleranfälligkeit der Komposition auf verschiedene Geräte.

Plan validieren Falls die Planung zu einem Plan führt, muss dieser vor der Ausführung validiert werden (siehe Abschnitt 6.2.2). Auch während der Ausführung des Plans ist eine Validierung angeraten, allerdings ist dafür dann die ausführende Komponente („controller“) verantwortlich. Im Falle einer verteilten Ausführung übernehmen die Geräte selbst die Validierung.

6.4 Start von Kompositionsläufen

Für die Komposition stellt sich nun die Frage, wann eine Planung und damit eine Komposition (neu)gestartet werden muss. Eingangs wurde eine intelligente Umgebung als zielorientiert charakterisiert (vgl. auch Heider und Kirste (2002)). Demnach reagiert die Umgebung, wenn sich die Ziele in der Umgebung, also die Intentionen der Nutzer ändern. Ein Plan und damit eine Komposition kann aber auch ungültig werden, wenn sich ein weiterer Bestandteil des Tripels $\mathcal{P} = (\Sigma, s_0, G)$, welches ein Planungsproblem definiert, ändert, also bei Änderungen der Operatoren oder Änderungen des gegenwärtigen Weltzustands.
Es kann der Fall eintreten, dass eine Intention des Nutzers zunächst nicht umgesetzt werden konnte, da ein bestimmter Dienst in der Umgebung fehlte oder der gegenwärtige Zustand der Umgebung die Ausführung eines essentiellen Dienstes unmöglich macht. Nach einiger Zeit wird aber der fehlende Dienst ausführbar, wodurch die Intention des Nutzers erfüllt werden könnte. Dieser ist dann allerdings in seinem Tun schon fortgeschritten und bedarf der Komposition u.U. nicht mehr. An diesem Punkt gibt es zwei mögliche Verfahrensweisen. Die Umgebung kann eine Information an den Nutzer ausgeben, dass sein altes Ziel nun erreichbar ist und nachfragen, ob die entsprechende Komposition geplant und ausgeführt werden soll. Dies würde allerdings die geforderte Unaufdringlichkeit der Umgebung aufgeben, da die Umgebung den Nutzer zum aktiven Handeln veranlassen würde. Alternativ hält sich die Umgebung an das Gebot der Unaufdringlichkeit und bleibt passiv. Dadurch wird der Nutzer in seiner neuen Tätigkeit nicht gestört, kann aber die Möglichkeiten der neuen Komposition nicht nutzen. Es sind allerdings auch Szenarien denkbar, in denen eine spätere Neuplanung und anschließende unaufgeforderte Ausführung im Hintergrund gewünscht sein können, da sie den Nutzer nicht stören, z.B. wenn durch eine neue Komposition Datenübertragungsraten gesteigert werden. Eine weitere Möglichkeit bestünde in der Angabe einer Gültigkeitsdauer für Ziele. Tritt innerhalb dieser Dauer eine Änderung ein, die das Ziel erfüllbar macht, kann auch verspätet noch eine Komposition veranlasst werden. Ansonsten bleibt die Umgebung passiv und erstellt keine neue Komposition.
Die Entscheidung ist an dieser Stelle immer eine Abwägung zwischen der Nützlichkeit einer (neuen) Komposition und den negativen Auswirkungen der dadurch verursachten Irritation des Nutzers, wobei eine automatische Entscheidungsfindung hier schwierig ist. Eine automatische Abschätzung würde eine Art Störmaß der Dienste und eine Einschätzung der Störtoleranz der Umgebung benötigen, die zum gegenwärtigen Zeitpunkt nicht verfügbar sind. Da für die hier vorgestellten Szenarien weder Gültigkeitsdauern von Zielen

noch ein Störmaß vorhanden waren, sollte im Zweifelsfall immer der Unaufdringlichkeit Vorrang gegeben werden und der Kompositionsprozess nur bei einer neuen Intention gestartet werden.

Kapitel 7

Umsetzung des Smart-Composers

7.1 Implementierung

Nach der Vorstellung des Konzeptes eines Composers im letzten Kapitel werden in diesem Kapitel einige Details der Implementierung des Composers erläutert. Dabei wird sich auf die Assemblierung der Domänen- und Problembeschreibungen und die Umsetzung der Planungsdienste sowie deren Verwendung durch den Metaplaner konzentriert.
Die Implementierung erfolgte in Java. Im Rahmen der Umsetzung wurde der Open-Source PDDL-Parser von Zeyn Saigol[1] verwendet. Dieser Parser wurde in Zusammenarbeit mit Christoph Burghardt, der ebenfalls Stipendiat im GRK MuSAMA ist, für die eigenen Anforderungen erweitert und angepasst. Die Webservicekomponenten, die für die Umsetzung des Metaplaners benötigt wurden, verwenden die AXIS2 Bibliotheken der Apache Software Foundation.[2]
Die Anforderungen einer intelligenten Umgebung stellen sich auch an die Implementierung des Composers und der weiteren umgesetzten Komponenten. Daher wurden alle wesentlichen Funktionalitäten modular in seperaten Klassen und Packages umgesetzt.
Die wichtigsten Zusammenhänge der Implementierung sind zum leichteren Verständnis in den entsprechenden Abschnitten als UML-Klassendiagramme dargestellt.

[1] http://www.cs.bham.ac.uk/~zas/software/graphplanner.html
[2] http://ws.apache.org/axis2/

7.2 Assemblierung des Planungsproblems

Treffen verschiedene zu komponierende Dienste ad hoc aufeinander, müssen die einzelnen Dienstbeschreibungen, die Nutzerintention und die aktuelle Situation der Umgebung zu einem Planungsproblem zusammengefasst werden. Dabei wird PDDL als intermediäre Austauschsprache verwendet. Im Abschnitt 6.3 wurde bereits deutlich, dass zunächst die PDDL-Domäne und im Anschluss das PDDL-Problem generiert werden.

7.2.1 Assemblierung der Domänenbeschreibung

Eine PDDL-Domäne besteht aus zwei Teilenm dem Kopf und der Operatorliste. Die Operatorbeschreibungen können aus gegebenen Dienstbeschreibungen extrahiert werden. Der Kopf muss anhand der Informationen aus den Operatoren erzeugt werden.

Im Abschnitt 3.3 wurden verschiedene Beschreibungssprachen vorgestellt. Dabei wurde ersichtlich, dass eine direkte Verwendung der Sprachen im Rahmen der Planung nicht möglich ist und in eine Umwandlung in PDDL erfolgen muss. Bei der Umsetzung wurde dieser Umwandlungsschritt vorausgesetzt, sodass angenommen wird, dass die Dienstbeschreibungen bereits als einzelne PDDL-Operatoren vorliegen. Aufgabe der Assemblierungskomponente ist es, aus dieser Menge von PDDL-Operatoren eine PDDL-Domäne zu erstellen.

Im Kopf der Domäne werden der Name sowie die Anforderungen der Domäne an einen Planer hinterlegt. Welche Anforderungen hier vermerkt werden, hängt von den syntaktischen Elementen ab, die in den Operatoren verwendet werden.[3] Um die Anzahl der Planer, die zur Lösung des Problems verwendet werden können, so wenig wie möglich zu reduzieren, empfiehlt es sich, die Anforderungsliste so klein wie möglich zu halten.

Weiterhin werden im Kopf die verwendeten Typen in einer Taxonomie hinterlegt.[4] Schließlich werden die in den Operatoren verwendeten Prädikate deklariert. Es können noch weitere Elemente wie zum Beispiel Konstanten oder abgeleitete Prädikate im Kopf der Domänen beschrieben werden. Auf diese Elemente wird im Folgenden jedoch nicht näher eingegangen, da sie bei der

[3] Die automatische Generierung der Anforderungsliste aus den Operatorbeschreibungen wurde nicht umgesetzt, ist jedoch einfach zu realisieren, da jedes PDDL-Requirement durch charakteristische Schlüsselwörter in den Operatorbeschreibungen zu identifizieren ist (z.B. erfordert ein forall die Einbindung von :universal−precondition in den Requirements). Im Rahmen der Implementierung wurden immer die Requirements :strips und :typing angenommen.

[4] Es ist anzumerken, dass es auch Domänen ohne Typendeklaration geben kann. Dies ist der Fall, wenn alle Operatoren auf eine Typendeklaration verzichten. Zur besseren Modellierung ist allerdings eine Verwendung von getypten Objekten ratsam.

Modellierung der Beispielszenarien (siehe Abschnitt 4.1) nicht benötigt wurden.

Algorithm 1 DomainAssembler

```
services ← getServices()
for all services do
  operators.add(service.getPDDLDescription())
types ← getTypHierarchie()
predicates ← getPredicates(operators)
return new domain(types, predicates, operators)
```

Der Algorithmus zur Assemblierung einer PDDL-Domäne aus gegebenen Operatorbeschreibungen bildet zunächst die Operatorenliste durch einfache Aneinanderreihung der einzelnen Operatoren. Bei einer Zusammenführung der Operatoren muss auf Duplikate überprüft werden. Angenommen es gibt in einer Umgebung drei Lampen, die alle über den Dienst lightOn(?lamp) verfügen. Die entsprechenden Operatoren der Dienste wären identisch und unterschieden sich lediglich in ihrer Instanziierung, die aber erst durch die Objekte der Problembeschreibung erfolgt. Da auf eine Verwendung von Objekten in den Operatorbeschreibungen verzichtet wird (siehe Abschnitt 4.1.6), sind mehrfach vorhandene Operatoren in einer Domäne überflüssig und werden bei der Assemblierung herausgefiltert. Es ist zu beachten, dass in PDDL die Identität von Operatoren lediglich über den Namen des Operators nicht aber über dessen Signatur definiert ist. Es kann demzufolge innerhalb einer Domäne nur einen Operator namens lightOn geben. Im Assemblierungsalgorithmus wird die Überprüfung auf Duplikate durch die Verwendung eines entsprechenden Datentypes (z.B. einer Liste) realisiert.
Um die Liste der verwendeten Prädikate einer Domäne zu erhalten, wird die Methode *getPredicates(operators)* aufgerufen. Dazu soll sich nochmals der Aufbau eines PDDL-Operators vergegenwärtigt werden. Der nachfolgend dargestellte Operator ProjectToSurface stammt aus dem einfachen SmartLab-Szenario (siehe Anhang A.4).

```
(:action ProjectToSurface
  :parameters (?p - projector ?s - surface ?d - data)
  :precondition (and (isDown ?s)(isPointingTo ?s ?p)(isActive ?d ?p))
  :effect (and (isActive ?d ?s)))
```

Im :parameters-Element des Operators werden die verwendeten Parameter (?p, ?s und ?d) samt ihrer Typen (projector , surface und date) deklariert. In den Elementen :precondition und :effect werden diese Parameter dann in den Argumenten der Prädikate verwendet, allerdings ohne die entsprechenden Typen.

Ziel der Methode *getPredicates()* des Domainassemblers ist es nun, aus den Operatorbeschreibungen eine getypte Liste der verwendeten Prädikate zu extrahieren, die einen obligatorischen Teil der Domänenbeschreibung darstellt. Auch hier sind Dopplungen möglich und müssen herausgefiltert werden. Werden getypte Variablen verwendet, müssen den Variablen der Prädikate die entsprechenden Typen aus der Parameterdefinition zugeordnet werden. Die Rückgabe der Funktion für den obigen Operator sollte demnach die folgende Liste von Prädikaten sein:

```
(isDown ?s - surface)
(isPointingTo ?s - surface ?p - projector)
(isActive ?d - data ?p - projector)
```

Algorithm 2 getPredicates(*operators*)

```
for all operators do
  predicates ← precond.getPredicates() ∪ effects.getPredicates()
  for all predicates do
    for all arguments do
      for all parameters do
        if parameter.name == argument.name then
          newArgument.setType(parameter.type)
      typedPredicate.add(newArgument)
    if !predicateList.contains(typedPredicate) then
      predicateList.add(typedPredicate)
    else
      getCommonType(typedPredicate, predicateListPredicate)
return predicateList
```

Bei der Verwendung von Typen kann ein weiteres Problem auftreten. So kann in einem Operator das Prädikat (isOn ?l) einen Parameter vom Typ lamp verwenden. Ein anderer Dienst nutzt (isOn ?p) dagegen mit einem Parameter vom Typ projector . Da in der Liste der Prädikate im Kopf der PDDL-Domäne nur ein Prädikat gleichen Names vorhanden sein kann, muss der vorliegende Typenkonflikt aufgelöst werden, indem ein gemeinsamer Obertyp[5], in diesem Fall zum Beispiel der Typ device, gesucht wird. Die Methode *getCommonType(firstType, secondType)* leistet diese Konfliktlösung. Dafür ist allerdings eine vorhandene Typenhierarchie nötig, in der der gemeinsame Obertyp gesucht werden kann. Zwar ist es vorstellbar, eine Typenhierarchie aus einer bestehenden T-box automatisch zu extrahieren, allerdings nur dann, wenn diese

[5]In der umgesetzten PDDL-Bibliothek sind Typen jeweils mit einem Verweis auf ihren Obertyp (parent) gespeichert. In der Definition von PDDL (Edelkamp und Hoffmann, 2004) ist der Typ object als absoluter Obertyp festgelegt, d.h. alle Typen haben object als direkten oder indirekten Obertyp.

Beschreibung bezüglich der Typen konsistent und eindeutig ist. Kommt es zu einem Konflikt, wie er hier beschrieben ist, wird die automatische Generierung einer solchen Hierarchie nicht mehr ohne Weiteres möglich sein. Im Rahmen der Umsetzung wird daher von gegebenen Typenhierarchien ausgegangen.

Algorithm 3 getCommonType($firstType$, $secondType$)

```
if firstType == secondType then
  return firstType
else
  currType ← globalHierarchie.get(firstType)
  while currType ≠ "object" do
    tempHierarchie.add(currType)
    currType ← currType.getParentType()
  currType ← hierarchie.get(secondType())
  while currType ≠ "object" do
    if tempHierarchie.contains(currType) then
      return currType
    else
      currType ← currType.getParentType()
return "object"
```

Konnten alle Prädikate aus den Operatoren konfliktfrei extrahiert werden, ist die Domänenbeschreibung vollständig.

7.2.2 Assemblierung der Problembeschreibung

Für die Assemblierung einer Problembeschreibung werden die aktuelle Situation der Umgebung, die Intention des Nutzers und die Liste der verfügbaren Objekte (Geräte) benötigt. Zusätzlich wird die Domäne benötigt, für die das Problem zusammengestellt werden soll. Theoretisch kann ein Problem auch ohne Kenntnis der Domäne assembliert werden. Dann jedoch besteht keine Sicherheit, dass die verwendeten Prädikate und ggf. Typen auch in der entsprechenden Form in der Domäne deklariert wurden. Es sollte daher die Domänenbeschreibung vor der Assemblierung des Problems vorhanden sein.
Die Beschreibung der initialen Situation muss als eine Menge von Belegungen der Prädikate der Domäne formuliert sein. Das Ziel kann als prädikatenlogischer, funktionsfreier Satz bestehend aus den Prädikaten der Domäne angegeben werden. Dabei ist zu beachten, dass eine Verwendung spezieller prädikatenlogischer Elemente, Allquantor oder Existenzquantor, eine PDDL-Anforderung darstellt und zu einem entsprechenden Eintrag in den :requirements der Domäne führt, wodurch sich die Menge der möglichen Planer zur Lösung des Problems u. U. verkleinert.
Der Algorithmus zur Assemblierung erzeugt aus der Liste der in der Umgebung vorhandenen Geräte die Liste der Objekte des PDDL-Problems. Zusammen mit dem initialen Weltzustand und dem Ziel kann eine Problembeschreibung kreiert werden.

Algorithm 4 ProblemAssembler(*domain*)

```
for all devices do
    objects.add(device)
initWS = env.getInitWS()
goal = env.getGoal()
return new Problem(d, objects, initWS, goal)
```

7.3 Wrapperklassen für die Planer

Bei der Umsetzung des Composers konnte auf Arbeiten, die im Rahmen der Experimente (siehe Abschnitt 5.1.3) durchgeführt wurden, zurückgegriffen werden. Zur automatisierten Durchführung der Experimente war eine Javainterne allgemeine Ansteuerung für jeden der untersuchten Planer nötig. Daneben war es erforderlich, die Laufzeit der Planungsprozesse durch einen zweiten, pro Planer parallel laufenden Thread zu überwachen und auf 2.000 ms zu beschränken. Der überwachende Thread wurde Wrapper genannt.
Um den Abbruch eines Planungslaufes implementieren zu können, musste der Planeraufruf asynchron erfolgen und aufgrund der unterschiedlichen Aufrufe für jeden Planer einzeln implementiert werden. Die ensprechenden Klassen heißen *SGPPlanner* für SGP, *LPGPlanner* für LPG usw. Die jeweiligen Interfaces zur Verwendung der Planer sind *IPlannerWrapper* und *IPlannerThread*.
Bei den verwendeten Planern können zwei Typen unterschieden werden. Der überwiegende Teil ist über Konsolenaufrufe steuerbar. Die Rückgabe dieser Planer erfolgt über die Konsolenausgabe oder teilweise über eine Dateirückgabe. Andere Planer wie z.B. UCPOP oder SGP sind in Lisp geschrieben und können nur in einer Lisp-Umgebung gestartet werden. Bei der Umsetzung wurde hier Common Lisp für Windows in der Version clisp-2.44 verwendet. Auch die Rückgaben der Lisp-basierten Planer konnte über die Konsole abgegriffen werden.
Der externe Programmaufruf der Planer ist zwar in den eigentlichen Planerklassen, die eine Umsetzung von *java.util.Thread* sind, implementiert. Der Start des Planerthreads und die Laufzeitüberwachung ist jedoch in der abstrakten Klasse *AbstractPlannerWrapper* definiert, von der alle Wrapper erben.
Die maximale Laufzeit, die einem Planer für die Lösung eines Problems eingeräumt wird, ist im Konstruktor des zugehörigen Wrappers anzugeben. Sollte die maximale Laufzeit (*threshold*) überschritten sein, wird die statische Methode *killPlanningProcess()*, die jeder Planer vom Typ *IPlannerThread* implementieren muss, aufgerufen. Da alle verwendeten Planer letztlich auf Kommandozeilenaufrufen beruhten, konnte diese Funktionalität in einer eigenen abstrakten Klasse (*RuntimeExecBasedPlanner*) gekapselt werden.
Neben dem Start und dem Abbruch von Planungsläufen muss den Planern das konkrete zu lösende Planungsproblem samt der dazugehörigen Domäne übergeben werden. Alle verwendeten Planer akzeptierten zwei getrennte PDDL-Dateien, eine Domänen- und eine Problembeschreibung, als Eingaben. Zusätzlich konnten oder mussten je Planer noch weitere Parameter angegeben werden. Da die Planer letztendlich über einen Kommandozeilenaufruf gestartet wurden, konnten die Daten der PDDL-Domäne und des PDDL-Problems

nicht als Objekte oder Streams übergeben werden. Sie werden stattdessen für jeden Planungslauf in ein Arbeitsverzeichnis, auf das der Planer Zugriff erhält, gespeichert. Dem Planer werden beim Aufruf lediglich die Pfade der zwei Dateien (PDDL-Domäne und PDDL-Problem) übergeben. Die Methoden *writeDomain()* und *writeProblem()* der Klasse *AbstractPlannerWrapper* übernehmen diese Aufgabe.
Letztlich wurde in den Planerklassen auch das Parsing der Ergebnisse der Planer realisiert. Diese Ergebnisse werden von den Planern zum Teil als Konsolen- oder als Dateiausgaben zurückgegeben. Jeder Planer verfügt über eine statische Methode *parseSolution()*, die die Rückgabe des jeweiligen Planers analysieren und die wesentlichen Ergebnisse (resultierender Plan, Grund bei Misserfolg der Planung, u.a.) extrahieren und sie mitsamt zusätzlich erhobener Metadaten wie Laufzeit oder Validität des resultierenden Plans in einer einheitlichen Datenstruktur vom Typ *Result* zurückgeben kann. Die eigentlichen Pläne werden als Objekt vom Typ *PartialOrderedPlan* zurückgegeben. Die Verwendung der Klasse *PartialOrderedPlan* erlaubt es, den Plan auch als TOP, als S-Expression mit separaten Ordnungsparametern oder als einfachen String auszugeben. In den Abbildungen 7.1 und 7.2 sind die Abhängigkeiten der beschriebenen Klassen als UML-Klassendiagramm dargestellt.

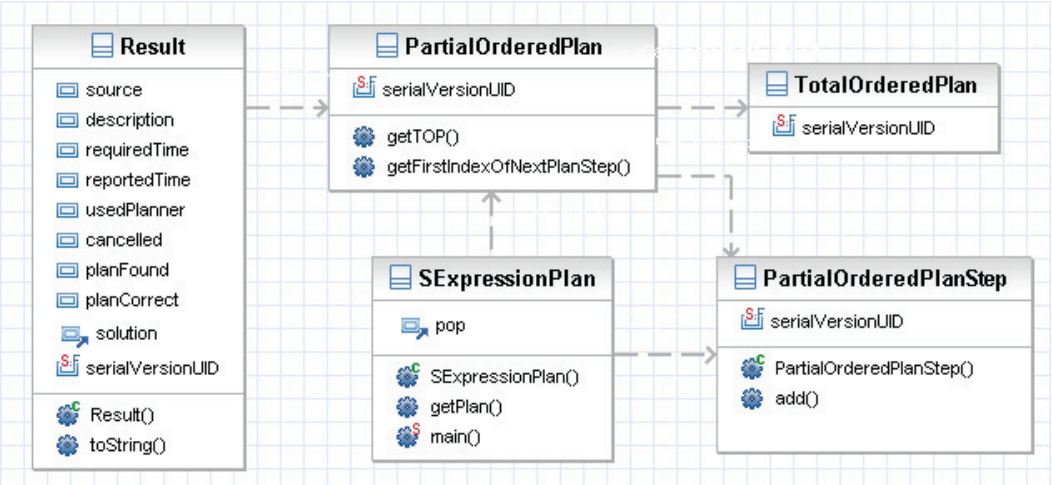

Abbildung 7.1: UML-Diagramm der Klasse *Result* zur Repräsentation eines Planungsergebnisses

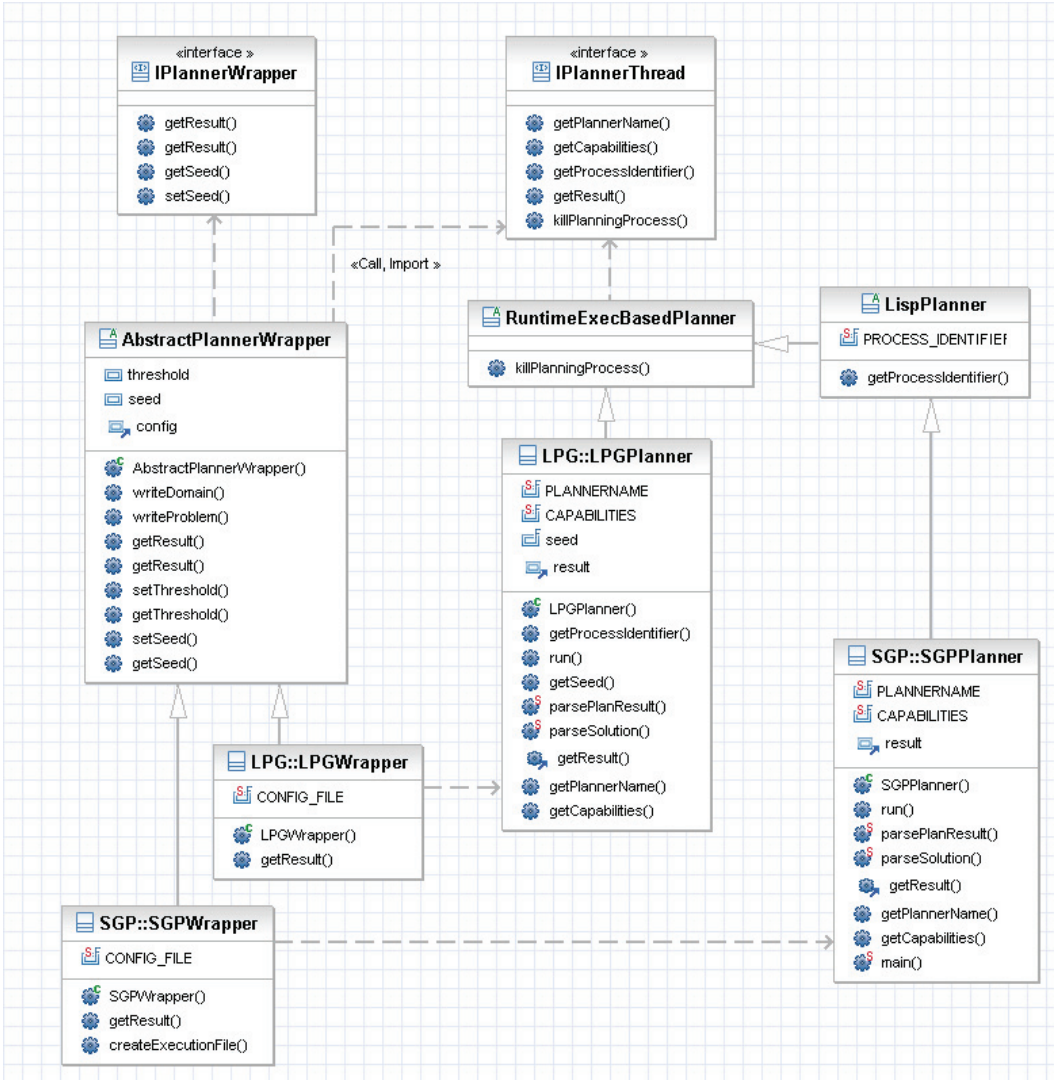

Abbildung 7.2: UML-Diagramm der Wrapper und der Planer am Beispiel von LPG und SGP

7.4 Composer

Im Kapitel 6 wurde eine Komponente zur Auswahl und Steuerung von Planern gefordert. Diese Komponente wird als Composer bezeichnet. Sie leistet neben der Überwachung und dem möglichen Abbruch eines Planungslaufes auch die Auswahl eines Planers in einer intelligenten Umgebung sowie die Umsetzung verschiedener Ausführungsstrategien (vgl. 5.1.5).
Im letzten Abschnitt 7.3 wurde gezeigt, wie die Laufzeitüberwachung und der Abbruch von Planern realisiert wurde. Im aktuellen Abschnitt wird die Umsetzung der Planer als Dienst und deren Verwendung innerhalb des Composers beschrieben.
Bevor die Umsetzung der Ausführungsstrategien beschrieben wird, soll zunächst die Planungsdienstschnittstelle vorgestellt werden. Darauf aufbauend wird im Anschluss die Umsetzung als Dienst erläutert.

7.4.1 Schnittstelle eines Planungsdienstes

Soll ein Planer als Dienst umgesetzt werden, ist es nötig, die Schnittstelle des Dienstes zu beschreiben. Aufgrund der Umsetzung als Webservice wird hierfür WSDL verwendet. Durch die Benutzung der Werkzeuge des Axis2 Paketes konnte die Erzeugung der WSDL-Beschreibungen als auch die Generierung der für einen Webservice nötigen Stubs und Skeletons automatisiert werden.
Durch die einheitliche Handhabung der Planer mit Hilfe von Wrappern (siehe Abschnitt 7.3) ist der Entwurf einer gemeinsamen Dienstschnittstelle einfach. Ein Planungsdienst hat vier Funktionen:

- *setDomainAndProblem(String d, String p)*
  Mit dieser Methode werden dem Planungsdienst die in PDDL formulierte Domäne und das Problem übergeben. Um Unfairness bei sehr langen Domänen- oder Problembeschreibungen zu vermeiden, wurde der Start des Planers vom Prozess des Sendens von Domäne und Problem an den Planer getrennt.

- *startPlanning()*
  Diese Methode gibt das Ergebnis der Planung zurück. Der Aufruf ist synchron implementiert. Die nötige Asynchronität muss durch den aufrufenden Composer umgesetzt werden, was durch die bereits erläuterten Wrapperklassen erreicht wird.

- *stopPlanning()*
  Der parameter- und rückgabelose Aufruf dieser Methode stoppt den aktuellen Planungsprozess des Dienstes. Diese Funktion kann unabhängig

davon, ob aktuell ein blockender Aufruf von *startPlanning()* läuft, aufgerufen werden.

- *getCapabilities()*
  Diese Funktion liefert die Fähigkeiten des Planers zur Zeit als einfachen String von PDDL-Requirements zurück. Sie ist für die Berücksichtigung neuer Planer im Planungsauswahlprozess wesentlich.

Die Umsetzung der Dienstklassen ähnelt sehr stark den im letzten Abschnitt vorgestellten Wrappern. Auch hier gibt es eine Zweiteilung zwischen Dienst und Planerklasse, um die asynchrone Erreichbarkeit der Dienstmethoden zu ermöglichen.
Um die Dienste seitens des Composers ansprechen zu können, ist es vorteilhaft, Dienstclients zu implementieren. Zwar wird von den AXIS2-Werkzeugen anhand der WSDL-Beschreibung ein Webservice-Stub generiert, der den Zugriff auf den Dienst ermöglicht, allerdings bestehen diese Stubs aus über 2.500 Zeilen automatisch generiertem, schlecht zu überblickendem Code, der zudem bei jeder Änderung neu generiert wird und manuelle Erweiterungen ignoriert und überschreibt.
In den Dienstclients werden beim Aufruf eines Planers sogenannte Callback-Objekte erzeugt, die aktiv werden, wenn der entsprechende Dienst eine Antwort sendet. Damit ist der eigentliche Dienstaufruf blockend, also synchron. Die Umsetzung im Composer ist jedoch nicht-blockend und damit asynchron. Daher muss für jeden verwendeten Planer ein zusätzlicher Thread, in dem das Callback-Objekt läuft, den blockenden Aufruf von *startPlanning()* ausführt und auf dessen Antwort wartet, gestartet werden.

7.4.2 Ablauf Composer

Wird der Composer gestartet, arbeitet er iterativ die folgenden Schritte ab. Dabei ist zur Zeit die Race-Strategie implementiert. Durch die Kapselung der verschiedenen Funktionalitäten ist eine Umsetzung der anderen Strategien ohne Weiteres möglich.[6]

1. *init()*
   Im ersten Schritt werden PDDL-Domäne und -Problem erstellt. Dazu werden die Daten der Umgebung an den Assembler delegiert, welcher daraus die Domäne und das Problem generiert (vgl. Abbildung 6.6).

[6] Einen einzelnen Planer auszuwählen und zu starten ist trivial genau wie das sukzessive Starten von Planern anhand einer Liste, was einer Umsetzunf der Austauschstrategie entspricht.

2. *discoverPlanner()*
   Hier wird eine Liste der in der Umgebung verfügbaren Planer aus dem Planerrepository erstellt. Anhand der Anforderugen des Planungsproblems können die Planer auf ihre funktionale Tauglichkeit zur Lösung des Problems überprüft werden. Dafür besitzen die Dienstclients der Planer die boolsche Methode *canHandle()*, die eine Liste von PDDL-Requirements als Parameter akzeptiert. Weiterhin können in diesem Auswahlschritt noch weitere Anforderungen an die Planer (z.B. die Generierung von POPs) überprüft werden.

3. *sendRequests()*
   Stehen die verfügbaren Planer fest, kann das Planungsproblem verschickt werden. Dabei werden zuerst alle Domänen- und Problembeschreibungen an die Planer gesendet. Sobald dieser Vorgang abgeschlossen ist, können die Planer parallel gestartet werden, was faire Voraussetzungen für die Bewertung der Antwortzeiten der Planer schafft.[7]

4. *waitForResponses()*
   Wurden die Planer gestartet, wartet der Composer auf den Aufruf der *update()*-Methode der Callback-Objekte der PlanerClients. Wurde ein valider Plan gefunden, wird die Planung aller anderen Planer mittels der Methode *stopPlanning()* abgebrochen.

Der letztendlich erhaltene Plan kann im Anschluss nochmals validiert und dann an die Umgebung übergeben werden. Zur Ausführung des Plans im aktuellen SmartLab ist eine Übermittlung des Plans als S-Expression nötig. Die entsprechende Umwandlung realisiert die Klasse *Result*.
Abbildung 7.3 verdeutlicht das Zusammenspiel der beschriebenen Komponenten und enthält weiterhin die Klassen *DomainAssembler* und *ProblemAssembler* (siehe Abschnitt 7.2).

[7] Ist ein Dienst ausgefallen und reagiert nicht auf das Senden der Daten, kann diese faire Variante unter Umständen zu unnötig langen Wartezeiten führen, da hier auf alle Planerdienste gewartet wird. Eine Entscheidung, wie in diesem Schritt am besten zu verfahren ist, sollte in der Praxis getroffen werden.

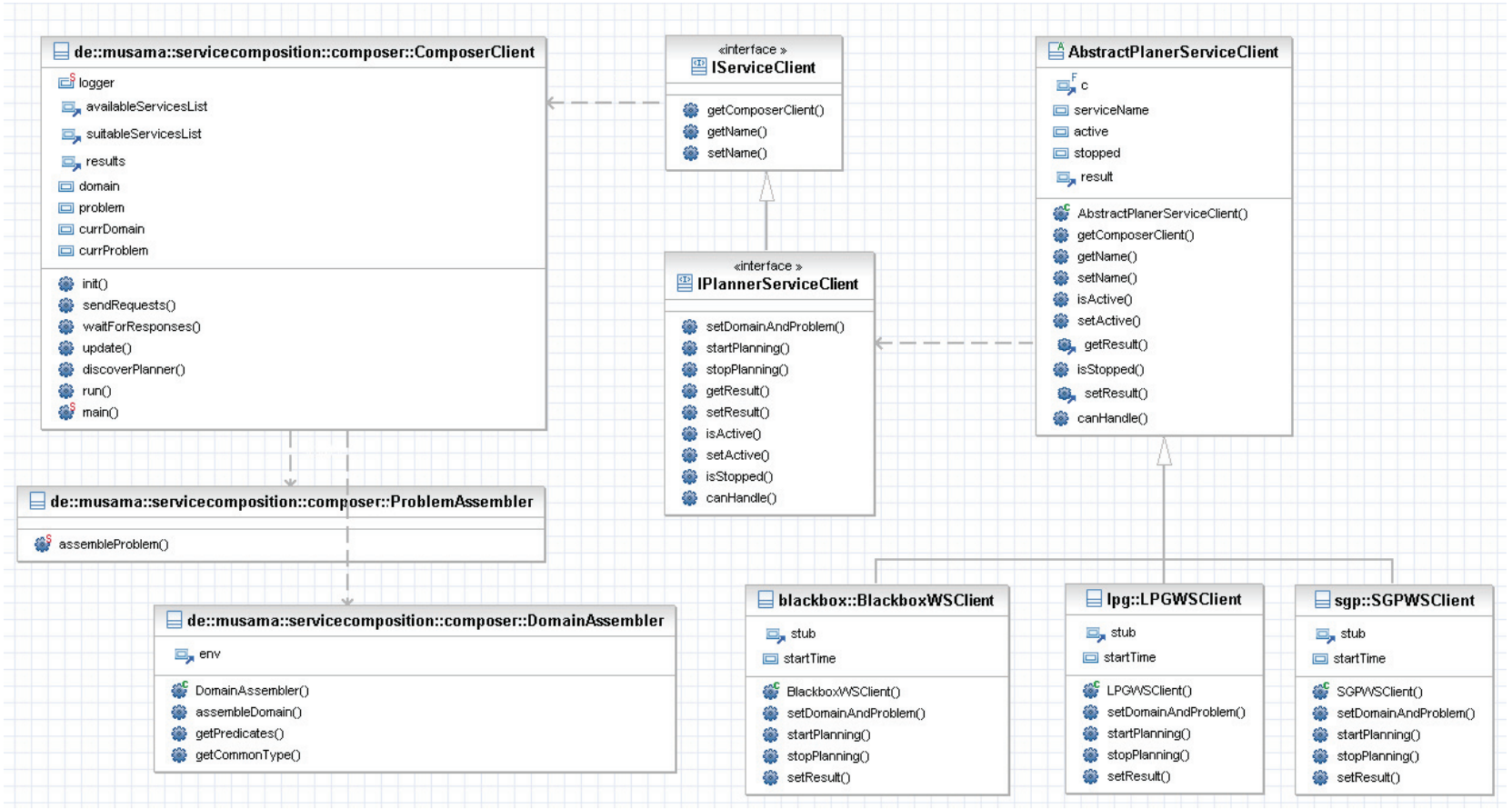

Abbildung 7.3: UML-Diagramm des Composers mit den Clients der Planungsdienste am Beispiel von blackbox, LPG und SGP

7.4.3 Einbinden eines neuen Planers

Durch die Etablierung einer einheitlichen Schnittstelle für Planer, wird das dynamische Hinzufügen und Entfernen von Planern durch die Umsetzung der Mechanismen einer SOA möglich.
Dabei ist es sinnvoll, in den Registrierungsprozess eines neuen Planers einige Tests zu integrieren. Hier könnten anhand der im Abschnitt 5.2.2 beschriebenen Testprobleme die Fähigkeiten des Planers überprüft werden. Auf diese Weise kann sichergestellt werden, dass alle zur Verfügung stehenden Planer konsistente funktionale Eigenschaften aufweisen.
Mit Hilfe der *getCapabilities()* ist es möglich, die Fähigkeiten des Planers zu beurteilen, was dem Composer eine funktionsbasierte Auswahl der Planer ermöglicht. Weiterhin sollte im Rahmen des Anmeldeprozesses die Effizienz des Planers getestet werden. So lässt sich beispielsweise der Offset des neuen Planers (vgl. Abschnitt 5.1.3.3) relativ schnell automatisch ermitteln, was eine gute Orientierung bei der Einordnung der Leistungsfähigkeit des neuen Planers im Vergleich zu den vorhandenen Planern ermöglicht. Daneben kann die Umgebung funktionale Anforderungen an die Planer haben (siehe Tabelle 5.8). So ist es beispielsweise denkbar, nur Planer zu verwenden, die partiell geordnete Pläne erzeugen können. Mit Hilfe der im Anhang B aufgelisteten Testprobleme lässt sich auch eine Überprüfung dieser Eigenschaften im Rahmen des Anmeldeprozesses umsetzen.
Sollten die entsprechenden Ressourcen vorhanden sein, können im Rahmen der Registrierung weiterhin die Laufzeitqualitäten eines neuen Planers anhand der in der Umgebung aufgetretenen Probleme in größerem Umfang getestet werden.
Auf Seiten der Implementierung wird die Repository-Funktionalität für die Planer zur Zeit durch den Composer zur Verfügung gestellt. Diese könnte und sollte bei einer weiterführenden Implementierung ausgelagert werden, um so die Komplexität des Composers weiter zu verringern.

7.4.4 Integration und Auslagerung der Planer

Anhand der durchgeführten Experimente (siehe Abschnitt 5) lässt sich erkennen, dass es zwar Unterschiede zwischen den Planern gibt, aber dennoch Planer existieren, die deutlich besser geeignet erscheinen, die gestellten Probleme zu lösen. Daher ist es zu überdenken einige Planer direkt in den Composer zu integrieren. Diese Vorgehensweise hätte den Vorteil, dass in einer unbekannten Umgebung immer ein Planer zur Verfügung steht. Weiterhin kann dessen Effizienz bereits im Vorfeld überprüft worden sein, was gute Laufzeitabschätzungen zuließe.

Allerdings müssen sich dann Planer und Composer die gleichen Ressourcen teilen, was aufgrund der starken CPU-Auslastung des Planungsprozesses problematisch werden kann, da u.U. die Reaktionszeit des Composers vermindert wird.
Werden Planer in den Composer integriert, können einzelne Planer zur Lösung eines Problems herangezogen werden. Auch die Austauschstrategie könnte umgesetzt werden. Eine Race-Strategie ist allerdings nicht realisierbar. Grund ist auch hierfür die Notwendigkeit, gemeinsame Ressourcen zu teilen, was selbst bei mehreren Prozessoren nicht immer fair umzusetzen ist.
Allerdings hat auch die Verteilung der Planer Nachteile. So ist nicht zu überprüfen, ob die Planungsergebnisse eines Planers fair sind. Es ist im Sinne der Agententheorie denkbar, dass es in einer intelligenten Umgebung egoistische Agenten/Geräte gibt. Ziel dieser Geräte könnte es sein, eigene Ressourcen zu sparen aber dennoch ihre Intentionen in vollem Umfang umzusetzen. Es ist vorstellbar, dass ein Planer, der im Sinne eines solchen egoistischen Gerätes handelt, Dienste des eigenen Gerätes bewusst nicht einsetzt, um so die eigenen Ressourcen zu schonen und nur dann einzusetzen, wenn es nicht anders möglich ist.
Ist in der Umgebung lediglich ein egoistischer Planer vorhanden, kann dem nicht begegnet werden. Wird jedoch davon ausgegangen, dass der egoistische Planer mit mindestens einem anderen, vergleichbar leistungsfähigen fairen Planer konkurriert und beide Planer versuchen, schnellstmöglich ein Ergebnis zu liefern, kann argumentiert werden, dass die vom egoistischen Planer angestrebten Einsparungen dem Planungsproblem zusätzliche Komplexität hinzufügen. Da der faire Planer ein einfacheres Problem zu lösen hat, stehen seine Chancen, schneller eine Lösung zu finden, dementsprechend besser, sodass es insgesamt wahrscheinlicher ist, dass ein fairer Plan zur Anwendung kommt.

Kapitel 8

Diskussion und Ausblick

8.1 Zusammenfassung

In der vorliegenden Arbeit wurde die Eignung KI-planungsbasierter Dienstekomposition für den Einsatz in intelligenten Umgebungen als Mittel zur Umsetzung von Nutzerassistenz und automatischer Gerätekooperation untersucht. Besonderes Augenmerk lag dabei auf der Unaufdringlichkeit der Komposition und der Fähigkeit in dynamischen Ad-hoc-Umgebungen einsetzbar zu sein.

Dienstekomposition Basierend auf einer Literaturrecherche wurden Kompositionsmethoden anhand einer temporalen und einer modalen Dimension klassifiziert. Dabei wurde die Erstellung des Prozessmodells als auch die Erstellung der Aktionssequenz betrachtet. Intelligente Umgebungen erfordern Kompositionsmethoden, die dynamisch und automatisch sowohl bei der Erzeugung des Prozessmodells als auch bei der Erzeugung der Aktionssequenz sind. Nur wenige Ansätze aus der Literatur entsprechen diesen Anforderungen. Von den geeigneten Ansätzen wird hauptsächlich KI-Planung zur Erstellung des Prozessmodells angewendet. Allerdings konnten keine Angaben über deren Laufzeitverhalten gefunden werden. Auch die Modellierung realer Szenarien wird in den entsprechenden Veröffentlichungen nicht thematisiert.

KI-Planung zur Komposition Aufbauend auf einer Formalisierung klassischer KI-Planung wurde beschrieben, dass der Anwendbarkeit von KI-Planung verschiedene Annahmen zugrunde liegen. Intelligente Umgebungen genügen nicht allen diesen Anforderungen. Einige Anforderungen von intelligenten Umgebungen werden auch von anderen Anwendungsgebieten der KI-Planung gestellt. So hat auch die Robotik einen starken physikalischen und räumlichen Bezug. Im Semantik Web muss mit offenen Systemen und Unsicherheiten geplant werden. Anpassungen dieser Gebiete können auch in intelligenten Umgebungen verwendet werden. Aufgrund der hohen Dynamik ist insbesondere die Annahme der Offline-Planung in intelligenten Umgebungen kritisch zu sehen. Durch sehr schnelle Planungsläufe kann dieser Anforderung jedoch begegnet werden.
Um Dienstekomposition in intelligenten Umgebungen durchführen zu können, müssen die Kompositionsprobleme so formuliert sein, dass sie von Planern verstanden und gelöst werden können. Als intermediäre Austauschsprache für Planungsprobleme wird PDDL verwendet, da fast alle Planer diese Sprache unterstützen. Durch den räumlichen Bezug intelligenter Umgebungen ist jedem Dienst ein Gerät in der Umgebung zuzuordnen, dabei hat jede Dienstausführung Effekte und Nebeneffekte, die die Umgebung beeinflussen. Anders als bei Webservices kann von dieser lokalen Komponente nicht abstrahiert werden. Daher müssen die Vorbedingungen und insbesondere die Effekte

eines Dienstes explizit deklariert sein. Nur so kann eine korrekte Komposition erreicht werden. Eine alleinige Verwendung einer syntaktischen Beschreibung der Ein- und Ausgänge eines Dienstes ist nicht ausreichend. Somit ist die Möglichkeit der Formulierung von PEs eine wesentliche Anforderung an eine Dienstbeschreibungssprache für intelligente Umgebungen in denen KI-planungsbasierte Komposition eingesetzt werden soll.
Bei der anschließenden Evaluierung aktueller Dienstbeschreibungssprachen stellte sich heraus, das keine Sprache direkte verwendbar war, jedoch OWL-S, WSMO als auch DSD geeignet sind, PEs in Dienstbeschreibungen zu hinterlegen. Für die Weiterverarbeitung durch einen Planer ist eine Umwandlung der Dienstbeschreibungen in PDDL-Operatoren dennoch unumgänglich.
Eine intelligente Umgebung ist zielorientiert, daher müssen neben den Diensten auch die Ziele formalisiert werden. Da KI-Planung auf einer Beschreibung der Welt als Zustandsübergangssystem beruht, bieten sich hier auch zustandsbasierte Ziele an. Eine Verwendung aufgabenbasierter Ziele ist nicht direkt möglich.

Modellierung Um die Anwendbarkeit von PDDL zur Beschreibung realer Anwendungen in intelligenten Umgebung zu belegen, wurden drei Szenarien aus der Literatur sowie ein eigenes Szenario, dass sich am SmartLab der Universität Rostock orientiert, als Kompositionsprobleme in PDDL modelliert.
Die Vision intelligenter Umgebung geht davon aus, dass in einer Ad-hoc-Umgebung verschiedene Geräte verschiedener Hersteller spontan zusammentreffen und dennoch in der Lage sind, sinnvoll miteinander zu interagieren. Bezogen auf die Komposition dieser Dienste bedeutet das, dass die Operatoren mit denen die Dienste der Geräte beschrieben sind, sinnvoll zu einer Domäne zusammengefasst werden können. Dieses Zusammenführen der Operatoren zu einer gemeinsamen Domäne geschieht dynamisch und erst zur Laufzeit der Umgebung. Um eine verteilte Modellierung von Operatoren zu gewährleisten, ist neben der Etablierung gemeinsamer Ontologien eine Modellierungsrichtline erforderlich. Eine solche Richtline wurde entworfen.

Laufzeitexperimente Um unaufdringliche Komposition zu ermöglichen, sind schnelle Planungszeiten erforderlich. Aus ersten Experimenten war zu sehen, dass Planer gleiche Probleme verschieden schnell lösen. Weiterhin sind Planer für die Lösung unterschiedlicher Probleme verschieden gut geeignet. Daher wurden Planer anhand von generierten Problemen als auch anhand der im Kapitel 4 erstellten Beispielszenarien evaluiert, um das Laufzeitverhalten der Planer bei der Verwendung in intelligenten Umgebungen abzuschätzen. Das Laufzeitverhalten wurde anhand der Parameter o (Anzahl der Opera-

toren), n (Anzahl der Zustandvariablen), i (Anzahl der wahren initialen Zustandsvariablen), g (Anzahl der Zielzustände) und r (Anzahl der Vorbedingungen und Effekte der Operatoren) untersucht. Es zeigte sich, dass die Anzahl der Zustandsvariablen den größten Einfluss auf die Planungsdauer hat.
Eine interssante Beobachtung dabei war, dass einzelne Planungsläufe sehr viel länger als vergleichbare Läufe benötigen. Diese Ausreißer oder Peaks gefährden die Unaufdringlichkeit einer intelligenten Umgebung. Grund für die Peaks ist das Phänomen der Phasentransition. Sie sind daher unvermeidlich.
Es wurden drei Hypothesen aufgestellt, die mit Hilfe der Experimente zu überprüfen waren:

1. Wenn ein Planungslauf in eine Laufzeitspitze gerät, ist es sehr wahrscheinlich, dass kein Plan gefunden wird und der Lauf daher vorzeitig abgebrochen werden kann.

2. Verschiedene Planer haben verschiedene Laufzeitspitzen.

3. Es kann eine signifikante Problemregion identifiziert werden, die stellvertretend für eine größere Region steht, um Vorhersagen über die Effizienz von Planern darin zu treffen.

Davon mussten die erste und die letzte Hypothese auf Grundlage der Experimentdaten verworfen werden. Es konnte jedoch nachgewiesen werden, dass verschiedene Planer unterschiedliche Peaks aufweisen, wodurch die zweite Hypothese angenommen werden konnte.

Eignung der Planer Die Evaluierung von fünf aktuellen Planern anhand der modellierten Beispielszenarien ergab, dass die Planer FF und LPG gut geeignet waren, Kompositionsprobleme zu lösen. Während FF sehr schnelle Lösungen generierte, jedoch eins der acht gestellten Problem nicht lösen konnte, war LPG in der Lage, alle Probleme zu lösen, allerdings insbesondere bei den schwierigeren Problemen in erheblich langsamerer Zeit als FF. Es war auch zu sehen, dass für Probleme die lediglich kausale Aktionsketten suchen, die Geschwindigkeit aktueller Planer ausreichend ist. Ist Ressourcenplanung in den Planungsproblemen enthalten, steigen die Laufzeiten der meisten Planer spürbar an.
Anhand der Laufzeitexperimente wurde weiterhin ersichtlich, dass sich Planer nicht nur anhand ihrer Effizienz sondern auch funktional unterscheiden. Bei der Synthese von verteilten Problemen ist es wichtig, die Eigenschaften der Planer genau zu kennen. Einfachste Problembeschreibungen, die von vier Planern problemlos geparst und gelöst werden, können beim fünften Planer zu einer Fehlermeldung oder keiner Lösung führen. Es konnten verschiedene

syntaktische und semantische Muster isoliert werden, die bei einigen Planern zu Problem führten. Beispiele dafür sind der korrekte Umgeng mit getypten Variablen oder nicht instanziierten Typen. Es wurden Testprobleme erstellt, die es erlauben, die funktionale Eignung von Planern zu überprüfen.
Werden die Erfahrungen aus Laufzeitexperimenten und funktionaler Überprüfung zusammen betrachtet, empfiehlt sich eine Verwendung von LPG bzw. FF für den Einsatz als Planer zur Dienstekomposition in einer intelligenten Umgebung. Ein weiterer aussichtsreicher Kandidat ist der Planer LAMA, der leider nicht mehr im Rahmen der Laufzeitexperimente evaluiert werden konnte, aber dennoch in Einzeltests sehr gute Leistungen zeigte. Im Gegensatz zu den beiden vorher genannten ist LAMA ein rein sequentieller Planer und so nicht in der Lage POPs zu erzeugen, was aus Sicht der Planausführung kritisch zu sehen ist.

Strategien Aufgrund der Beobachtungen, die während der Laufzeitexperimente gesammelt worden, wurden drei verschiedene Strategien zur Verringerung der Laufzeit der Planungsläufe vorgeschlagen und untersucht: Austausch-, Neustart- und Race-Strategie. Insbesondere die Austauschstrategie erwies sich hierbei als vorteilhaft. Dabei werden die in einer intelligenten Umgebung verfügbaren Planer zunächst nach ihrer voraussichtlichen Lösungsgeschwindgkeit in einer Liste geordnet. Dann bekommen die Planer ein bestimmtes Zeitfenster zur Lösung des aktuellen Problems. Kann innerhalb dieser Spanne keine Lösung gefunden werden, wird der Planungslauf abgebrochen und dem nächste Planer aus der Liste wird das Problem zu Lösung übergeben.
Bedingung für einen erfolgreichen Einsatz der Strategie ist eine gute Heuristik, die einerseits zu einer effektiven Ordnung der Planer führt und andererseits die Zeitfenster, die pro Planer unterschiedlich sein können, möglichst optimal berechnet. Basierend auf den Offsets der Planer kann eine solche Heuristik in einer Ad-hoc-Umgebung schnell initiiert werden kann. Sie sollte allerdings während der Laufzeit der intelligenten Umgebung angepasst werden.

Entwurf und Umsetzung des Composers Um KI-planungsbasierte Komposition von Diensten in intelligenten Umgebungen durchführen zu können, bedarf es einer Komponente, die diese Aufgabe in der Umgebung übernimmt. Die entwickelte Komponente, der Composer, wurden für die prototypische Umsetzung im Rahmen des GRK MuSAMA zur Verfügung gestellt. Der Composer wurde zentral entworfen, da zum einen die Assemblierung der Domänen- und Problembeschreibung nur zentral erfolgen kann. Zum anderen arbeitet auch klassische KI-Planung üblicherweise zentral. Zentrale Komponenten in dynamischen Umgebungen haben Nachteile. Sie können ausfallen und damit

im schlimmsten Fall die Funktionalität der Umgebung negativ beeinflussen. Es muss dabei jedoch beachtet werden, dass die Komposition aus der Generierung des Prozessmodells und der Ausführung der Aktionssequenz besteht. Die verteilte Ausführung von Aktionssequenzen (in Fall der Verwendung von KI-Planungs also Plänen) ist trotz zentraler Erstellung des Plans einfach umzusetzen, sodass lediglich die Assemblierung der Domänen- und Problembeschreibung sowie die anschließende Erstellung des Plans zentral erfolgen muss. Dabei handelt es sich bei den betrachteten Szenarien um Zeiträum, die im Sekundenbereich liegen, wodurch der Verlust beim Ausfall der Kompositionskomponente gering ist.
Weiterhin wurde eine allgemeine Schnittstelle für einen Planungsdienst vorgestellt, was eine Auslagerung der Planer aus dem Composer auf verschieden Geräte der Umgebung ermöglicht, sodass lediglich die Funktionalität des Composers als zentral anzusehen ist, was das Gesamtsystem robuster gegenüber Ausfällen macht. Die geringe Größe des Composers erlaubt weiterhin eine schnelle und einfache Migration. Zwar ist das beschriebene Konzept des Composers prinzipiell immer noch als zentral anzusehen, dennoch wurde durch die Verteilung der Planer versucht, der Dynamik intelligenter Umgebungen gerecht zu werden.
Die Planung erfolgt offline. Während der Planungszeit kann sich der Zustand der Welt jedoch ändern. Betrifft eine solche Änderung die Planung wird von einer kritischer Änderungen gesprochen. Im Falle einer kritischer Änderung muss eine Planung wiederholt werden. Auch hier ist der Verlust aufgrund der anvisierten Planungszeiten im Sekundenbereich gering.
Als Teil des Composers wird eine Komponente zur Planerselektion und Überwachung vorgestellt, deren Aufgabe es ist, die beschriebenen Strategien umzusetzen. Das Konzept der hier vorgestellten Planerauswahl verfügt zusätzlich zur effizienzbasierten auch über eine funktionsbasierte Auswahl und ermöglicht weiterhin die Überwachung von Planungsläufen und stellt so einen signifikanten Fortschritt gegenüber des von Howe u. a. (1999) beschriebenen Metaplaners BUS dar, der sich auf die Auswahl eines bestmöglichen Planers für ein bestimmtes Problem beschränkte. Des Weiteren war BUS eine statische Implementierung mit vier fest integrierten Planern. Durch eine servicebasierte Umsetzung ist der hier beschrieben Composer in der Lage, auch neue unbekannte Planer zu integrieren, was einen zusätzlichen Vorteil darstellt.
Wesentlich für die Unaufdringlichkeit einer Komposition ist neben deren Geschwindigkeit ist auch der Zeitpunkt an dem eine Komposition gestartet und vor allem ausgeführt wird. Wird eine Komposition zum falschen Zeitpunkt gestartet kann sie den Nutzer erheblich stören. Es wurde daher argumentiert, dass eine Komposition aus Gründen der Unaufdringlichkeit nur bei einer neuen Nutzerintention gestartet werden sollte.

8.2 Bewertung der Eignung des Composers

Als Anforderungen an eine Architektur zur Nutzerassistenz in intelligenten Umgebungen nennen Heider und Kirste (2002) die folgenden Punkte, anhand derer sich die Eignung des Composers für einen Einsatz in intelligenten Umgebungen überprüfen lässt. Im Anschluss an die jeweilige Anforderung wird eine entsprechende Bewertung des in dieser Arbeit entworfenen Composers gegeben.

- *Sicherstellung der Unabhängigkeit der Komponenten*
  Eine Unabhängigkeit der Komponenten ist durch die servicebasierte Umsetzung erfüllt. Die Modellierungsrichtlinie erlaubt darüber hinaus unabhängige Modellierung der Gerätebeschreibungen.

- *Dynamische Erweiterbarkeit durch neue Komponenten*
  Da das der Komposition zugrunde liegende Planungsproblem vor jeder Planung neu erstellt wird, ist diese Dynamik gegeben. Auch die Menge der Planer ist durch eine servicebasierte Integration dynamisch erweiterbar.

- *Vermeidung von zentralen Komponenten*
  Diese Forderung kann nicht vollständig erfüllt werden. Durch Trennung von Planern und Planermanagement ist allerdings eine Verteilung dieser Komponenten möglich, was die Robustheit des Systems signifikant erhöht.

- *Unterstützung von verteilter Implementierung*
  Durch die Nutzung von PDDL als allgemein anerkannte und verbreitete Sprache zur Beschreibung von Planungsproblemen sowie die Beschreibung und Verwendung einer allgemeinen Schnittstelle für Planungsdienste wird eine verteilte Implementierung unterstützt. Daneben sind die Modellierungsrichtlinen ein wesentlicher Schritt zur einheitlichen und verteilten Implementierung von Geräten intelligenter Umgebungen.

- *Die flexible Wiederverwendbarkeit*
  Die Verwendung von serviceorientierter Architektur in Verbindung mit standardisierten bzw. akzeptierten Beschreibungssprachen ermöglicht eine flexible Wiederverwendbarkeit aller Komponenten.

- *Austauschbarkeit der Komponenten*
  Durch einheitliche Schnittstellen ist auch die Austauschbarkeit der Komponenten gegeben.

Es kann zusammengefasst werden, dass bis auf die Vermeidung zentraler Komponenten allen von Heider und Kirste (2002) gestellten Anforderungen mit dem hier vorgestellten Composer entsprochen werden konnte.

8.3 Diskussion und Ausblick

Das hier dargestellte Konzept weist Grenzen auf, die kurz genannt werden sollen. Aufgrund der Bearbeitung des Themas entwickelten sich weiterhin neue Fragen und Ideen, deren Beantwortung zukünftigen Arbeiten überlassen werden muss.

8.3.1 Logikbasierte Beschreibung der Umgebung

Wenngleich die Beschreibung von Geräten mit elaborierten logikbasierten Sprachen wie PDDL wesentlich komfortabler als beispielsweise eine Modellierung mit einfachen Hornklauseln ist, so ist sie dennoch nur von einem Spezialisten durchzuführen. Die Modellierung von Gerätefunktionalität beispielsweise durch den Anwender ist nicht realistisch.
Bei der Formulierung von Operatoren für verschiedene Dienste ist zu beobachten, dass sich bestimmte Vorbedingungen und Effekte häufig wiederholen. Diese könnten in abstrakte Operatoren ausgelagert werden, die bestimmten Gerätetypen zuzuordnen wären. Eine solche Zuordnung einzelner Geräte zu vordefinierten Gerätetypen erlaubt beispielsweise UPnP. Die Verwendung abstrakter Operatoren würde einerseits die Beschreibungen der Dienste vereinfachen und so u.U. auch die Modellierung vereinfachen. Sie würde andererseits auch die Möglichkeit eröffnen, abstrakte Pläne zu erzeugen, die wesentlich mehr Freiheitsgrade als instanziierte Pläne hätten und so eine dynamischere Anwendung ermöglichten.
Abstrakte Operatoren könnten auch bei der verteilten Modellierung aller Vorbedingungen und Effekte der von einem Gerät angebotenen Dienste helfen. Diese Modellierung ist zumindest für die Hersteller der Geräte eine nicht unerhebliche zusätzliche Belastung. Abgesehen von der daraus entstehenden Notwendigkeit, sich mit anderen Herstellern auf einheitliche Ontologien zu einigen, müssten alle Effekte und insbesondere Nebeneffekte einer Dienstausführung formalisiert werden. Eine solche Anforderung an die Industrie ist kritisch zu sehen. Ein möglicher Kompromiss könnte darin bestehen, sich auf eine Reihe von Umweltfaktoren, wie Geräuschpegel, Lichtintensität, Wärmeentwicklung u.a. zu einigen, deren Beeinflussung durch eine Dienstausführung immer beschrieben werden müsste. Schon mit diesem ersten Schritt dürften sich interessante Anwendungsmöglichkeiten von KI-planungsbasierter Komposition eröffnen.

Eine weitere kritische Anforderung des in dieser Arbeit vorgestellten Konzeptes zur Dienstekomposition ist die Notwendigkeit, den aktuellen Zustand aller Dienste in der Umgebung verfügbar zu machen. Wurde in der Umgebung eine Lampe eingeschaltet, muss dieser Fakt auch für den Composer verfügbar sein. Denkbar wäre eine zusätzliche Statuskomponente pro Gerät, die den aktuellen Zustand des Gerätes verfügbar macht. Da intelligente Umgebungen auch Klein- und Kleinstgeräte beinhalten, ist dies nicht für alle Geräte realistisch. Für solche Geräte müssten die Zustände zentral gehalten werden, was jedoch dem Gebot der Vermeidung von Zentralität widersprechen würde.
Im Abschnitt 4.2.4 wurden verschiedene locks erwähnt, die zur Modellierung andauernder Aktionen nötig sind. Sie könnten über abstrakte Operatoren realisiert werden. Allerdings entsteht dann das Bedürfnis nach Rechten, da es Situationen gibt, in denen locks aufgelöst werden dürfen, in anderen jedoch nicht. Eine Konfliktlösung kann die Kompositionsinstanz hier nicht leisten. Da eine Rechteverwaltung in PDDL zur Zeit nicht vorhanden ist und auch inhaltlich wenig mit der Sprache korreliert, müssten für Verwaltung von Rechten andere Sprachen zum Einsatz kommen und im Rahmen der Transformation der Dienstbeschreibungen in PDDL aufgelöst werden.

8.3.2 Unterstützung mehrerer Benutzer

Zur Zeit wird nur die Komposition von Zielen für einen Benutzer betrachtet. Daher wird generell davon ausgegangen, dass in einer Umgebung nur ein Ziel gleichzeitig existiert. Sind verschiedene Nutzer mit konkurrierenden Zielen im Raum, können Konflikte zwischen den Zielen auftreten. Eine Möglichkeit diese Konflikte zu lösen, ist eine wiederum zentrale Intentionsanalyse. Seitens der Komposition kann dieser Konflikt nicht gelöst werden, der beschriebene Composer kann nur mit einem Ziel gleichzeitig umgehen.

8.3.3 Planerheuristik

Es konnte keine einfach zu berechnende Maßzahl gefunden werden, anhand der eine schnelle Bewertung der Effektivität der Planer erfolgen könnte. Stattdessen ist die Verwendung lernender Verfahren, die sukzessive die Tauglichkeit verschiedener Planer bewerten können, an dieser Stelle erfolgversprechender. Dies ermöglicht es zudem, neue unbekannte Planer in den Kompositionsprozess mit einzubeziehen.
Auch die vorgestellten Strategien, insbesondere die Austauschstrategie, beruhen auf einer möglichst optimalen Bewertung der Leistungsfähigkeit der Planer. Lässt sich die funktionsbasierte Leistungsfähigkeit eines Planers durch die Evaluierung des Planers anhand der vorgestellten Testprobleme noch rela-

tiv einfach umsetzen, ist die Bewertung des Laufzeitverhaltens eines Planers schwieriger. Der verlässlichste Weg einen effizienten Planer für ein Problem zu finden, ist der empirische Vergleich. Da ein solcher Vergleich in Ad-hoc-Umgebungen nicht zu realisieren ist, sollten hier lernende Verfahren zum Einsatz kommen.
Wenn eine gelernte Heuristik verwendet wird, ist allerdings auch das Problem des Single-Point-of-Failiure kritischer, da mit einem Ausfall des Composers auch die angelernte Heuristik verloren geht. Daher sollte ein robustes System diese Heuristik redundant und verteilt verwalten.

Anhang A

Beispielszenarien

A.1 Toms Routenplanung

PDDL-Domäne

```
(define (domain toms-route-planning)
  (:requirements :strips :typing)
  (:types data printer - object)
  (:predicates
    (isOnline ?p - printer)
    (isStart ?a - data)
    (isDestination ?a - data)
    (isAddress ?a - data)
    (isURL ?d - data)
    (isPrinted ?d - data)
    (containsRoute ?d - data ?a1 ?a2 - data)
    (isEmptyDocument ?d - data)
  )

  (:action switchOn
    :parameters (?p - printer)
    :precondition (not (isOnline ?p))
    :effect (isOnline ?p))

  (:action printDocument
        :parameters (?d - data ?p - printer)
        :precondition (and (isOnline ?p)
                           (not (isEmptyDocument ?d)))
        :effect (isPrinted ?d))

  (:action getAddress
        :parameters (?a - data)
        :precondition (isURL ?a)
        :effect (and (not (isURL ?a))(isAddress ?a)))

  (:action getRoute
    :parameters (?a1 ?a2 - data ?d - data)
    :precondition (and (isStart ?a1)
                       (isDestination ?a2)
                       (isAddress ?a1)
                       (isAddress ?a2)
                       (isEmptyDocument ?d))
    :effect (and (containsRoute ?d ?a1 ?a2)
                 (not (isEmptyDocument ?d))))
)
```

PDDL-Problem

```
(define (problem toms-route-planning-problem)
        (:domain toms-route-planning)
        (:objects TomsAddr RestaurantsAddr Doc - data
        TomsPrinter - printer
    )
        (:init
     (isEmptyDocument Doc)
     (isAddress TomsAddr)
     (isStart TomsAddr)
     (isURL RestaurantsAddr)
     (isDestination RestaurantsAddr)
    )

        (:goal (and (containsRoute Doc TomsAddr RestaurantsAddr)
                 (isPrinted Doc)))
)
```

A.2 Janes Email

PDDL-Domäne

```
(define (domain janes-email)
  (:requirements :strips :typing)
  (:types email laptop thruput location - object)
  (:predicates

    (isat ?loc - location ?l - laptop)
    (isMovable ?l - laptop)
    (currentThruput ?l - laptop ?t - thruput)
    (availableThruput ?loc - location ?t - thruput)
    (requiredThruput ?e - email ?t - thruput)
    (mailSent ?e - email)
    (checkedIn ?loc - location)
  )

  (:action connectWLAN
    :parameters (?l - laptop ?loc - location ?t - thruput)
    :precondition (and (isat ?loc ?l)(availableThruput ?loc ?t))
    :effect (currentThruput ?l ?t))

  (:action sendMail
        :parameters (?e - email ?l - laptop ?t - thruput)
        :precondition (and (requiredThruput ?e ?t)(currentThruput ?l ?t))
        :effect (mailSent ?e))

  (:action move
    :parameters (?from ?to - location ?l - laptop)
    :precondition (and (isMovable ?l)(isat ?from ?l)(not (isat ?to ?l)))
    :effect (and (isat ?to ?l)(not (isat ?from ?l))))

  (:action checkIn
    :parameters (?l - laptop ?loc - location)
    :precondition (and (isat ?loc ?l))
    :effect (and (checkedIn ?loc)(not (isMovable ?l))))
)
```

PDDL-Problem

```
(define (problem janes-email-problem)
        (:domain janes-email)
        (:objects JanesEmail - email
        JanesLaptop - laptop
        HighT MediumT LowT - thruput
        Gate12 Gate5 - location
    )

        (:init
        (isat Gate12 JanesLaptop)
        (isMovable JanesLaptop)
        (requiredThruput JanesEmail HighT)
        (availableThruput Gate12 LowT)
        (availableThruput Gate5 HighT)
    )

        (:goal (and (mailSent JanesEmail)(checkedIn Gate12)))
)
```

A.3 Barts Anweisung

PDDL-Domäne

```
(define (domain barts-order)
  (:requirements :strips :typing)
  (:types
        object
        data person device vehicle location - object
        stream format - data
  )

  (:predicates
    (isat ?loc - location ?l - object)
    (isLocked ?o - object)
    (isMovable ?l - object)
    (hasFormat ?f - format ?d - data)
    (isAvailable ?d - data ?dev - device)
    (isConnected ?dev1 ?dev2 - device)
    (isActive ?dev - device ?d - data ?f - format)
    (supportsFormat ?f - format ?dev - device)
    (isConsuming ?s - stream ?p - person)
    (isPaused ?s - stream ?dev - device)
    (hasConsumed ?s - stream ?p - person)
    (pickedUp ?p1 ?p2 - person ?l - location)
  )

  (:action startStream
    :parameters (?dev - device ?s - stream ?f - format)
    :precondition (and (not (isLocked ?dev))
                       (isAvailable ?s ?dev)
                       (hasFormat ?f ?s)
                       (supportsFormat ?f ?dev))
    :effect (and (isActive ?dev ?s ?f)
                 (isLocked ?dev)))

  (:action stopStream
    :parameters (?dev - device ?s - stream ?f - format)
    :precondition (isActive ?dev ?s ?f)
    :effect (and (not (isActive ?dev ?s ?f))
                 (not (isLocked ?dev))))

  (:action accessStream
    :parameters (?scrDev ?destDev - device ?s - stream ?f - format)
    :precondition (and (isActive ?scrDev ?s ?f)
                       (isConnected ?scrDev ?destDev)
                       (not (isLocked ?destDev))
                       (hasFormat ?f ?s)
                       (supportsFormat ?f ?destDev))
    :effect (and (isLocked ?destDev)
                 (isActive ?destDev ?s ?f)))

  (:action connect
    :parameters (?dev1 ?dev2 - device)
    :precondition (and (not (isLocked ?dev1))
                       (not (isLocked ?dev2)))
    :effect (and (isConnected ?dev1 ?dev2)))

  (:action convertdata
    :parameters (?d - data ?f1 ?f2 - format ?dev - device)
    :precondition (and (supportsFormat ?f1 ?dev)
                       (supportsFormat ?f2 ?dev)
```

```
                        (hasFormat ?f1 ?d)
                        (isActive ?dev ?d ?f1))
    :effect (and (hasFormat ?f2 ?d)
                 (not (hasFormat ?f1 ?d))
                 (isActive ?dev ?d ?f2)
                 (not (isActive ?dev ?d ?f1))))

  (:action move
    :parameters (?from ?to - location ?p - person)
    :precondition (and (isMovable ?p)
                       (not (isLocked ?p))
                       (isat ?from ?p)
                       (not (isat ?to ?p)))
    :effect (and (isat ?to ?p)
                 (not (isat ?from ?p))))

  (:action pickUp
    :parameters (?p1 ?p2 - person ?l - location)
    :precondition (and (isat ?l ?p1)
                       (isat ?l ?p2)
                       (not (isLocked ?p1))
                       (not (isLocked ?p2)))
    :effect (pickedUp ?p1 ?p2 ?l))
)
```

PDDL-Problem

```
(define (problem barts-order-problem)
        (:domain barts-order)
        (:objects
        Bart Client - person
        Home InCar Airport - location
        CarRadio TV PDA - device
        Audio Video - format
        RecordedNews - stream
    )

        (:init
        (isat Home PDA)
        (supportsFormat Audio PDA)
        (supportsFormat Video PDA)

        (isat Home TV)
        (supportsFormat Video TV)
        (isAvailable RecordedNews TV)
        (hasFormat Video RecordedNews)
        (isActive TV RecordedNews Video)
        (isLocked TV)

        (isat InCar CarRadio)
        (supportsFormat Audio CarRadio)

        (isat InCar Bart)
        (isMovable Bart)
    )

        (:goal (isActive CarRadio RecordedNews Audio))
)
```

A.4 SmartLab - einfach

PDDL-Domäne

```
(define (domain simple-smartLab)
  (:requirements :strips :typing)
  (:types device data - object
          format document - data
          notebook projector extron surface lamp - device)

  (:predicates
    (empty)
    (isActive ?d - data ?device - device)
    (isAvailable ?d - document ?device - device)
    (isDown ?s - surface)
    (isLightOn ?l - lamp)
    (isMovable ?p - projector)
    (isPointingTo ?s - surface ?p - projector)
    (isAdjustedTo ?s - surface ?p - projector)
    (isConnected ?d1 ?d2 - device))

  (:action CanvasUp
    :parameters (?s - surface)
    :precondition (isDown ?s)
    :effect (not (isDown ?s)))

  (:action CanvasDown
    :parameters (?s - surface)
    :precondition (not (isDown ?s))
    :effect (isDown ?s))

  (:action LightOn
    :parameters (?l - lamp)
    :precondition (not (isLightOn ?l))
    :effect (isLightOn ?l))

  (:action LightOff
    :parameters (?l - lamp)
    :precondition (isLightOn ?l)
    :effect (not (isLightOn ?l)))

  (:action ShowDocument
    :parameters (?d - document ?n - notebook)
    :precondition (isAvailable ?d ?n)
    :effect (isActive ?d ?n))

  (:action SwitchNotebookOnProjector
    :parameters (?e - extron ?n - notebook ?p - projector ?d - data)
    :precondition (isActive ?d ?n)
    :effect (and (isConnected ?n ?p)(isActive ?d ?p)))

  (:action ProjectToSurface
    :parameters (?p - projector ?s - surface ?d - data)
    :precondition (and (isDown ?s)(isPointingTo ?s ?p)(isActive ?d ?p))
    :effect (and (isActive ?d ?s)))

  (:action MoveProjectorToSurface
    :parameters (?p - projector ?sfrom ?sto - surface)
    :precondition (and (isMovable ?p)(isPointingTo ?sfrom ?p)
                  (isAdjustedTo ?sto ?p))
    :effect (and (isPointingTo ?sto ?p)(not (isPointingTo ?sfrom ?p))))
```

```
  (:action AdjustProjectorToSurface
    :parameters (?p - projector ?sfrom ?sto - surface)
    :precondition (and (isAdjustedTo ?sfrom ?p))
    :effect (and (isAdjustedTo ?sto ?p)(not (isAdjustedTo ?sfrom ?p))))
)
```

PDDL-Problem

```
(define (problem simple-Problem)
    (:domain simple-smartLab)
    (:objects
            LAMP1 LAMP2 - lamp
            NB1 - notebook
            DOC1 DOC2 - document
            NEC - projector
            EXTRON - extron
            LW1 LW2 - surface)
        (:init
        (isAvailable DOC1 NB1)
        (isMovable NEC)
        (isAdjustedTo LW2 NEC)
        (isPointingTo LW2 NEC))

    (:goal (and (isActive DOC1 LW1)(isLightOn LAMP1))))
```

A.5 SmartLab - erweitert mit 4 Dokumenten

PDDL-Domäne

```
(define (domain smartLab)
  (:requirements :strips :typing)
  (:types device data - object
          format document - data
          notebook projector surface lamp - device)

  (:predicates
    (isLocked ?d - device)
    (isActive ?d - document ?device - device)
    (isAvailable ?d - data ?device - device)
    (isDown ?s - surface)
    (isMovable ?p - projector)
    (isPointingTo ?s - surface ?p - projector)
    (isLightOn ?l - lamp)
    (hasFormat ?f - format ?d - data)
    (supportsFormat ?f - format ?dev - device)
    (canConvert ?device - device ?ffrom ?fto - format))

  (:action CanvasUp
    :parameters (?s - surface)
    :precondition (and (not (isLocked ?s)) (isDown ?s))
    :effect (not (isDown ?s)))
  (:action CanvasDown
    :parameters (?s - surface)
    :precondition (and (not (isLocked ?s))(not (isDown ?s)))
    :effect (isDown ?s))
  (:action LightOn
    :parameters (?l - lamp)
    :precondition (not (isLightOn ?l))
    :effect (isLightOn ?l))
  (:action LightOff
    :parameters (?l - lamp)
    :precondition (isLightOn ?l)
    :effect (not (isLightOn ?l)))

  (:action showDocument
    :parameters (?d - document ?f - format ?n - notebook)
    :precondition (and (isAvailable ?d ?n)(hasFormat ?f ?d)
                    (supportsFormat ?f ?n)(not (isLocked ?n)))
    :effect (and (isActive ?d ?n)(isLocked ?n)))
  (:action SwitchNotebookOnProjector
    :parameters (?n - notebook ?p - projector ?d - document)
    :precondition (and (not (isLocked ?p))(isActive ?d ?n))
    :effect (and (isLocked ?p)(not (isActive ?d ?n))(isActive ?d ?p)))

  (:action projectToSurface
    :parameters (?p - projector ?s - surface ?d - document)
    :precondition (and (not (isLocked ?s))(isActive ?d ?p)(isDown ?s)
                    (isPointingTo ?s ?p))
    :effect (and (isLocked ?s)(not (isActive ?d ?p))(isActive ?d ?s)))
  (:action moveProjector
    :parameters (?p - projector ?sfrom ?sto - surface)
    :precondition (and (not (isLocked ?p))(isMovable ?p)(not (isLocked ?sfrom))
                    (not (isLocked ?sto))(isPointingTo ?sfrom ?p))
    :effect (and (isPointingTo ?sto ?p)(not (isPointingTo ?sfrom ?p))))

  (:action moveDocument
    :parameters (?d - document ?f - format ?src ?dest - notebook)
```

```
    :precondition (and (not (isLocked ?src))(hasFormat ?f ?d)
                    (supportsFormat ?f ?dest)(isAvailable ?d ?src))
    :effect (and (not (isAvailable ?d ?src))(isAvailable ?d ?dest)))
  (:action convertDocument
    :parameters (?d - document ?ffrom ?fto - format ?dev - device)
    :precondition (and (canConvert ?dev ?ffrom ?fto)(not (isLocked ?dev))
                    (isAvailable ?d ?dev)(hasFormat ?ffrom ?d))
    :effect (hasFormat ?fto ?d)))
```

PDDL-Problem

```
(define (problem smartLab-Problem-4docs)
    (:domain smartLab)
    (:objects   LAMP1 LAMP2 LAMP3 LAMP4 LAMP5 LAMP6 - lamp
            NB1 NB2 NB3 NB4 - notebook
            DOC1 DOC2 DOC3 DOC4 - document
            PPT PDF - format
            EPS1 EPS3 EPS6 Panasonic NEC-MT1065 - projector
            LW1 LW2 LW3 LW4 LW5 LW6 VD1 VD2 - surface)
        (:init
        (isAvailable DOC1 NB1)
        (isAvailable DOC2 NB2)
        (isAvailable DOC3 NB2)
        (isAvailable DOC4 NB2)
        (isMovable NEC-MT1065)
        (isPointingTo LW2 NEC-MT1065)
        (isPointingTo LW1 EPS1)
        (isPointingTo LW3 EPS3)
        (isPointingTo LW6 EPS6)
        (hasFormat PPT DOC1)
        (hasFormat PPT DOC2)
        (hasFormat PPT DOC3)
        (hasFormat PPT DOC4)
        (supportsFormat PPT NB1)
        (supportsFormat PDF NB1)
        (supportsFormat PDF NB2)
        (supportsFormat PDF NB3)
        (supportsFormat PDF NB4)
        (canConvert NB1 PPT PDF)
    )
    (:goal (and (isActive DOC2 LW1)(isActive DOC1 LW3)
                (isActive DOC3 LW6)(isActive DOC4 LW4)
                (isLightOn LAMP1)(isLightOn LAMP2)))
)
```

Anhang B

Testprobleme

B.1 Test - Nicht instanziierte Typen

PDDL-Domäne

```
(define (domain empty-types)
  (:requirements :strips :typing)
  (:types device - object
          surface lamp - device)

  (:predicates
    (isLocked ?d - device)
    (isDown ?s - surface)
    (isLightOn ?l - lamp))

  (:action CanvasUp
    :parameters (?s - surface)
    :precondition (and (not (isLocked ?s)) (isDown ?s))
    :effect (not (isDown ?s)))

  (:action CanvasDown
    :parameters (?s - surface)
    :precondition (and (not (isLocked ?s))(not (isDown ?s)))
    :effect (isDown ?s))

  (:action LightOn
    :parameters (?l - lamp)
    :precondition (not (isLightOn ?l))
    :effect (isLightOn ?l))

  (:action LightOff
    :parameters (?l - lamp)
    :precondition (isLightOn ?l)
    :effect (not (isLightOn ?l))))
```

PDDL-Problem

```
(define (problem empty-types-problem)
    (:domain empty-types)
    ;No objects of type 'surface' are instantiated, although operators using this
    ;type (CanvasUp and CanvasDown) are declared in the domain.
    ;Therefore LPG can not handle this problem, however SGP and Blackbox can.
    (:objects LAMP1 - lamp)
        (:init
        (isLightOn LAMP1)
    )
    (:goal (and (not (isLightOn LAMP1))))
)
```

Erwarteter Plan

```
0 (LIGHTOFF LAMP1)
```

B.2 Test - Vererbung von Typen

PDDL-Domäne

```
(define (domain typeInferration)
        (:requirements :strips :typing)
        (:types
        object
        device document - object
        notebook - device
)
        (:predicates
                (canConvertPPT2PDF ?a0 - device)
                (isPDF ?a0 - document ?a1 - device)
                (isPPT ?a0 - document ?a1 - device)
)

(:action ConvertPPT2PDF
        :parameters (?d - document ?dev - device)
        :precondition (and (canConvertPPT2PDF ?dev)(isPPT ?d ?dev))
        :effect (isPDF ?d ?dev)))
```

PDDL-Problem

```
(define (problem typeInferration-Problem)
        (:domain typeInferration)
    ;Blackbox can not solve this plan, as it does not infer NB1 to be a device
    ;Thus it can not use the operator ConvertPPT2PDF which is defined for devices
    ;and not only for notebooks
        (:objects DOC1 - document NB1 - notebook)
        (:init
                (isPPT DOC1 NB1)
                (canConvertPPT2PDF NB1))
        (:goal (and (isPDF DOC1 NB1))))
```

Erwarteter Plan

```
0 (CONVERTPPT2PDF DOC1 NB1)
```

B.3 Test - Objekte in Domänenbeschreibung

PDDL-Domäne

```
(define (domain objects-in-domain)
  (:requirements :strips :typing)
  (:types device - object
          lamp - device)

  (:predicates
    (isLightOn ?l - lamp))

  (:action LightOn
    :parameters ()
    :precondition (not (isLightOn LAMP1))
    :effect (isLightOn LAMP1))

  (:action LightOff
    :parameters ()
    :precondition (isLightOn LAMP1)
    :effect (not (isLightOn LAMP1))))
```

PDDL-Problem

```
(define (problem objects-in-domain-problem)
    (:domain objects-in-domain)
    ;Usually PDDL prohibits the usage of objects in the domain description
    ;This problem instantiates LAMP1 as objects of type 'lamp'
    ;LAMP1 is used in the operators 'LightOn' and 'LightOff'
    (:objects LAMP1 - lamp)
        (:init
        (isLightOn LAMP1)
    )
    (:goal (and (not (isLightOn LAMP1))))
)
```

Erwarteter Plan

```
(nil)
```

B.4 Test - Partiell geordnete Pläne

PDDL-Domäne

```
(define (domain partialOrder-domain)
  (:requirements :strips)

  (:predicates
    (LeftSockOn)
    (RightSockOn)
    (LeftShoeOn)
    (RightShoeOn)
    )

 (:action LeftSock
 :parameters ()
 :effect (LeftSockOn))

 (:action RightSock
 :parameters ()
 :effect (RightSockOn))

 (:action LeftShoe
 :parameters ()
 :precondition (LeftSockOn)
 :effect (LeftShoeOn))

 (:action RightShoe
 :parameters ()
 :precondition (RightSockOn)
 :effect (RightShoeOn))
 )
```

PDDL-Problem

```
(define (problem partialOrder-problem)
    (:domain partialOrder-domain)
    (:objects obj)
    (:init )
    (:goal (and (LeftShoeOn)(RightShoeOn)))
)
```

Erwarteter Plan

```
0 (LEFTSOCK)
0 (RIGHTSOCK)
1 (LEFTSHOE)
1 (RIGHTSHOE)
```

Anhang C

Umfrage

C.1 Anschreiben

Dear Prof./Dr. —NAME—

My name is Florian Marquardt and I am PhD Student at the University of Rostock/Germany. Here I am involved in the graduate school MuSAMA (www.musama.de). My research is about service composition in smart environments.
To identify suitable composition algorithms I rely on basic structural information of several different smart environments which is difficult to find in scientific publications. To aquire this information I am now collecting it directly from the involved scientists.
I have written this mail to you because according to your research activity you were —POSITION—/involved in the project —PROJECTNAME— and thus concerned with smart environments. So I would like to ask you four short questions, which should not take too much of your time. If you are not the right contact person for my request, please apologize. I'd appreciate very much if you let me know who I can contact or forward this mail.
The informations I am interested in:

1. How many services (see explanation 1 below) were involved in your smart environment?

2. How many atomic (explanation 2) and composed (explanation 3) services were involved in your smart environment?

3. Regarding the number of services per device, what is the more common case a 1:1 relation (one device one service) or a 1:n relation (one device many services).

4. Did you use some kind of automatic or semi-automatic composition technique?

I hope I could attract your attention and you take a few moments to answer me. I have sent this mail to many scientists and developers on the field of

smart environments to gain best possible results. As a matter of course I will inform you of my analysis results if you are interested. If you have further questions regarding this mail, my work or the project I am involved do not hesitate to ask me.

Thanking you in advance for your help and with kind regards

Florian Marquardt

Explanations:
1) Due to its "smartness" I regard every functionality of an electronical entity in a smart environment as a service, capable of describing itself and answering to requests, or sending messages autonomously. Furthermore all services are accessable through a network. Besides I assume each service to be located at a distinct device. A device is capable of hosting several services.
2) I distinguish atomic and composed services, whereas atomic services can be executed directly and do not rely on any other services. So for example a sensor measuring temperature and propagating it to the environment is an atomic service for me as well as a large scale database answering to complex requests.
3) Composed services depend on other atomic and/or composed services. A composed service must not really be composed during runtime it is just a naming which describes that it could be composed if a suitable composition instance would be available. So for example a registering service that relies on a remote database is a composed service in my terms.

Anhang D

Ergebnisse der gernerierten Laufzeitexperimente

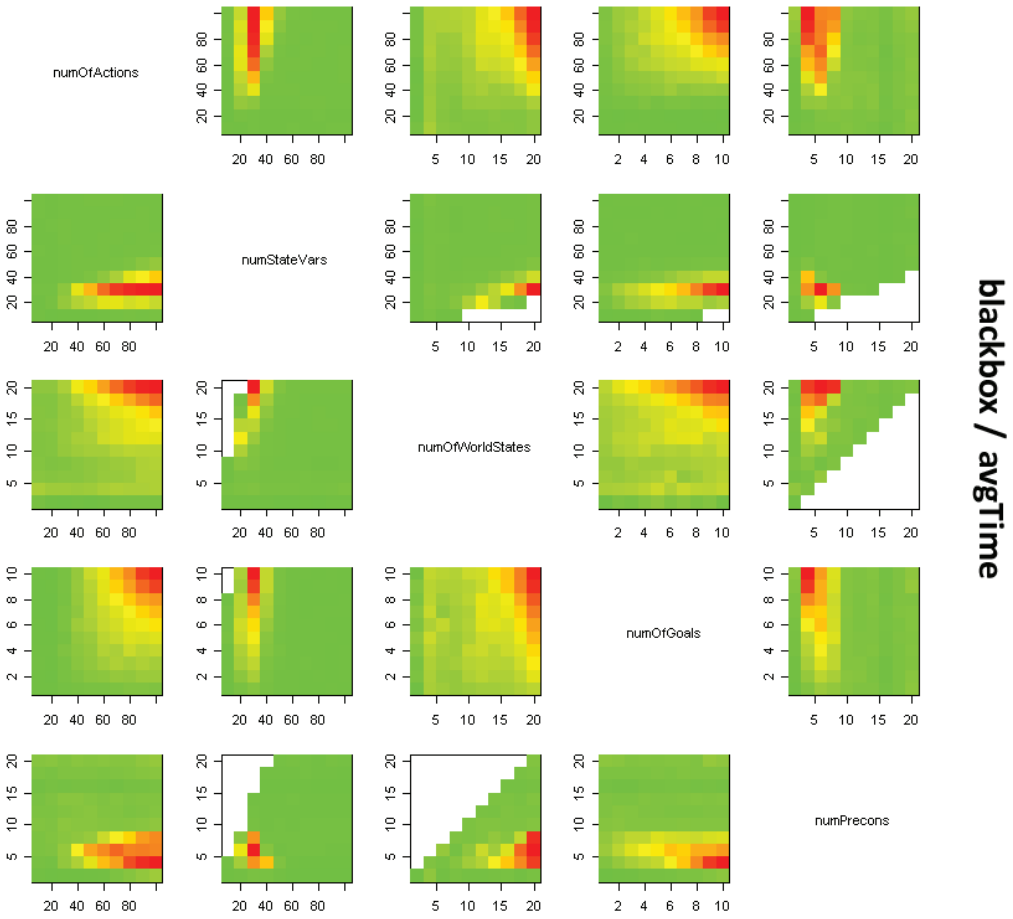

Abbildung D.1: Korrelationsmatrix der durchschnittlichen Planungsdauer; blackbox

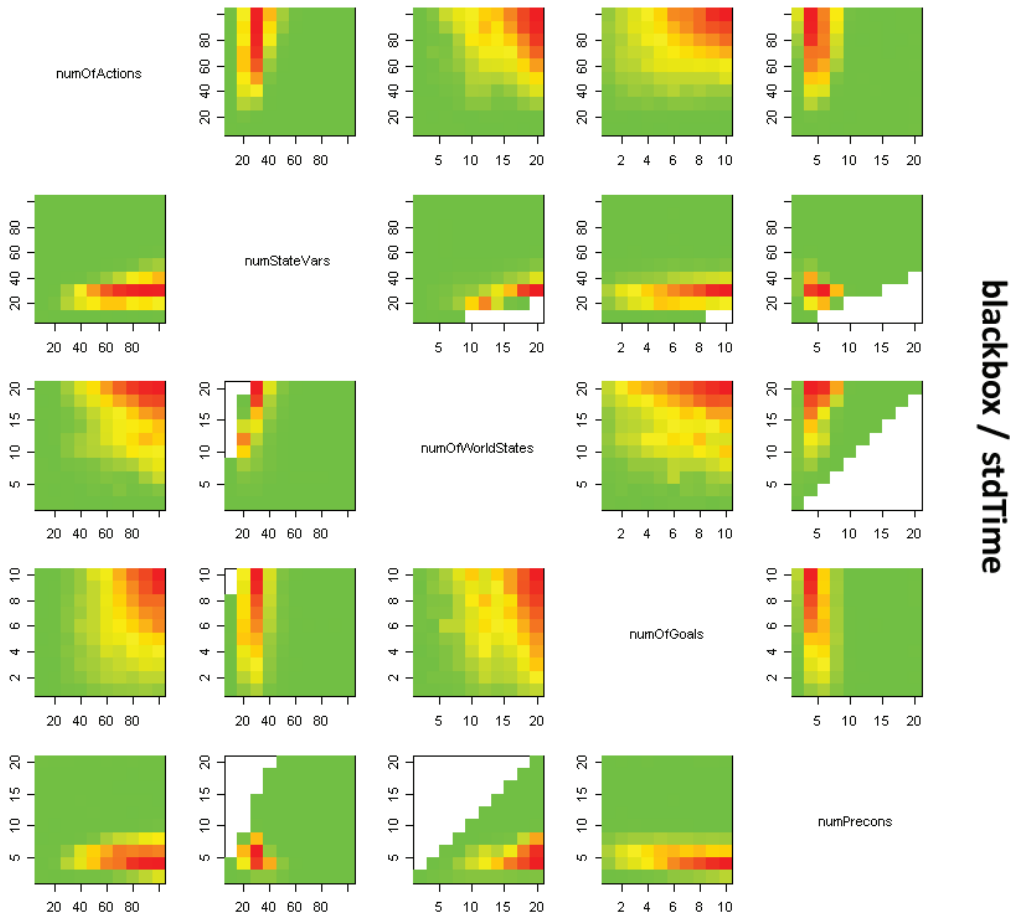

Abbildung D.2: Korrelationsmatrix der Standardabweichung der Planungsdauer; blackbox

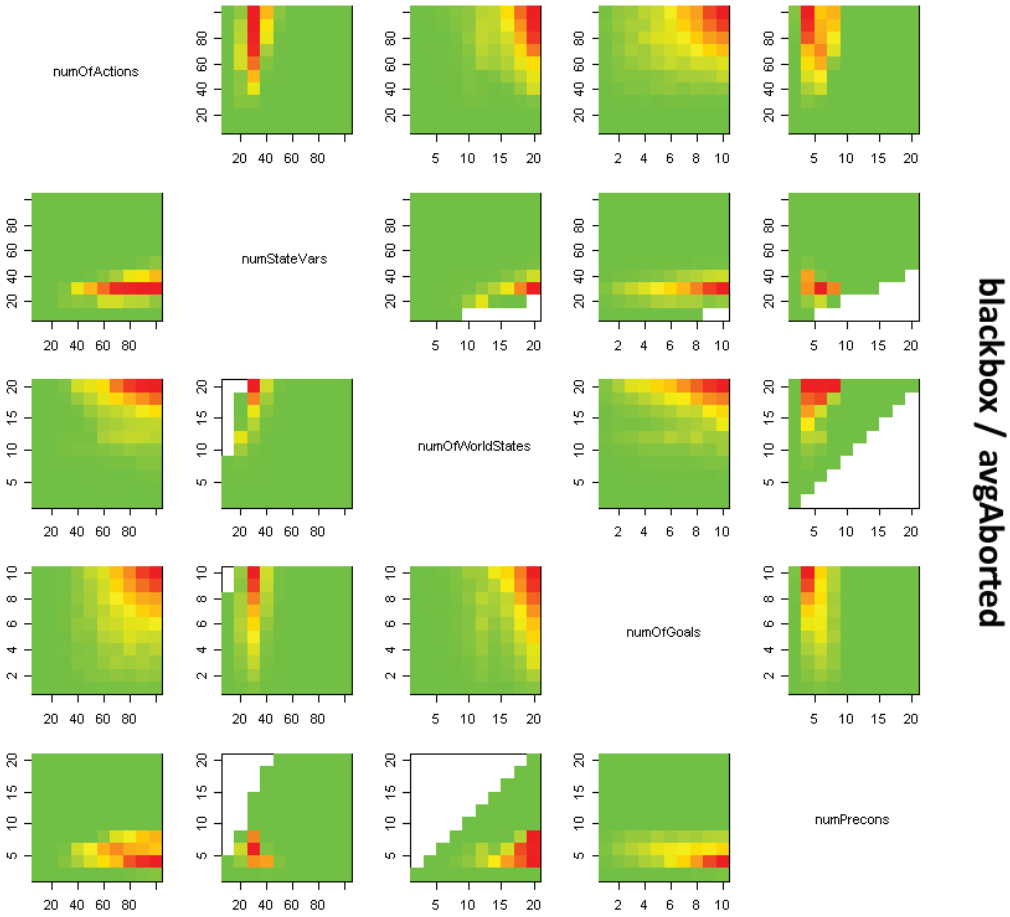

Abbildung D.3: Korrelationsmatrix der durchschnittlich abgebrochenen Läufe; blackbox

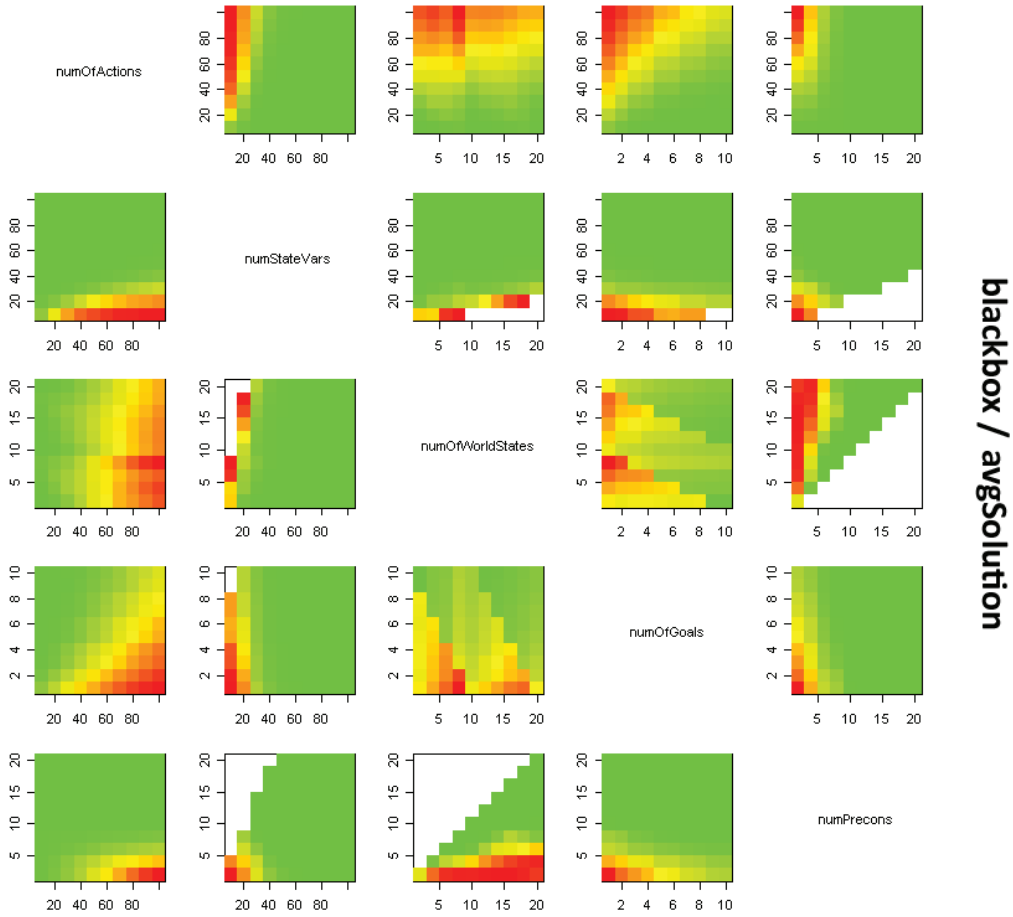

Abbildung D.4: Korrelationsmatrix der Anzahl der gefundenen Lösungen; blackbox

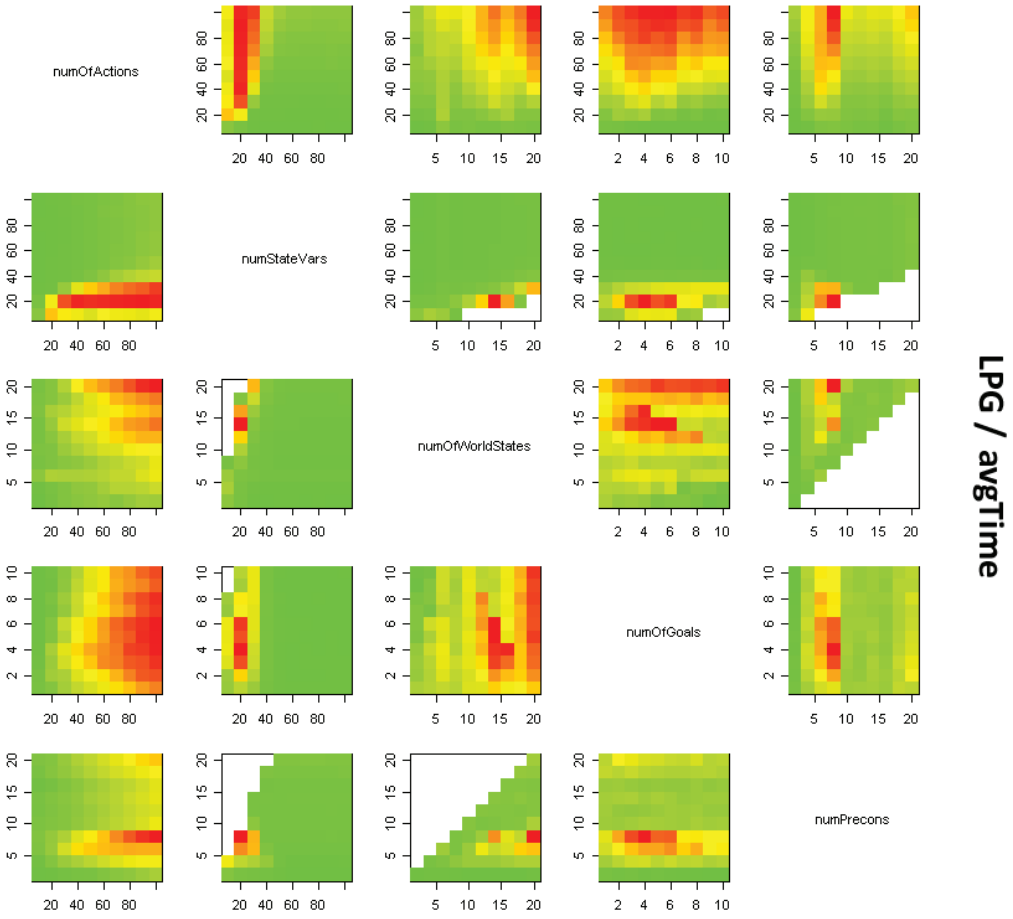

Abbildung D.5: Korrelationsmatrix der durchschnittlichen Planungsdauer; LPG

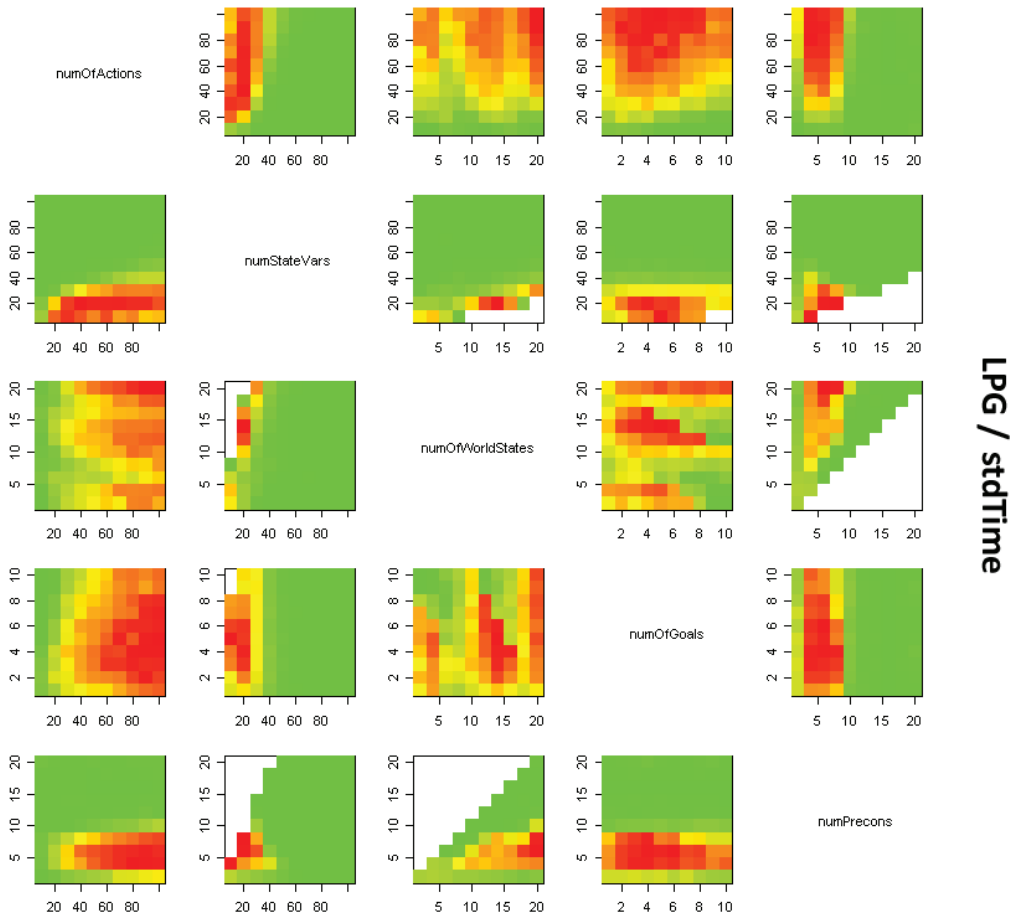

Abbildung D.6: Korrelationsmatrix der Standardabweichung der Planungsdauer; LPG

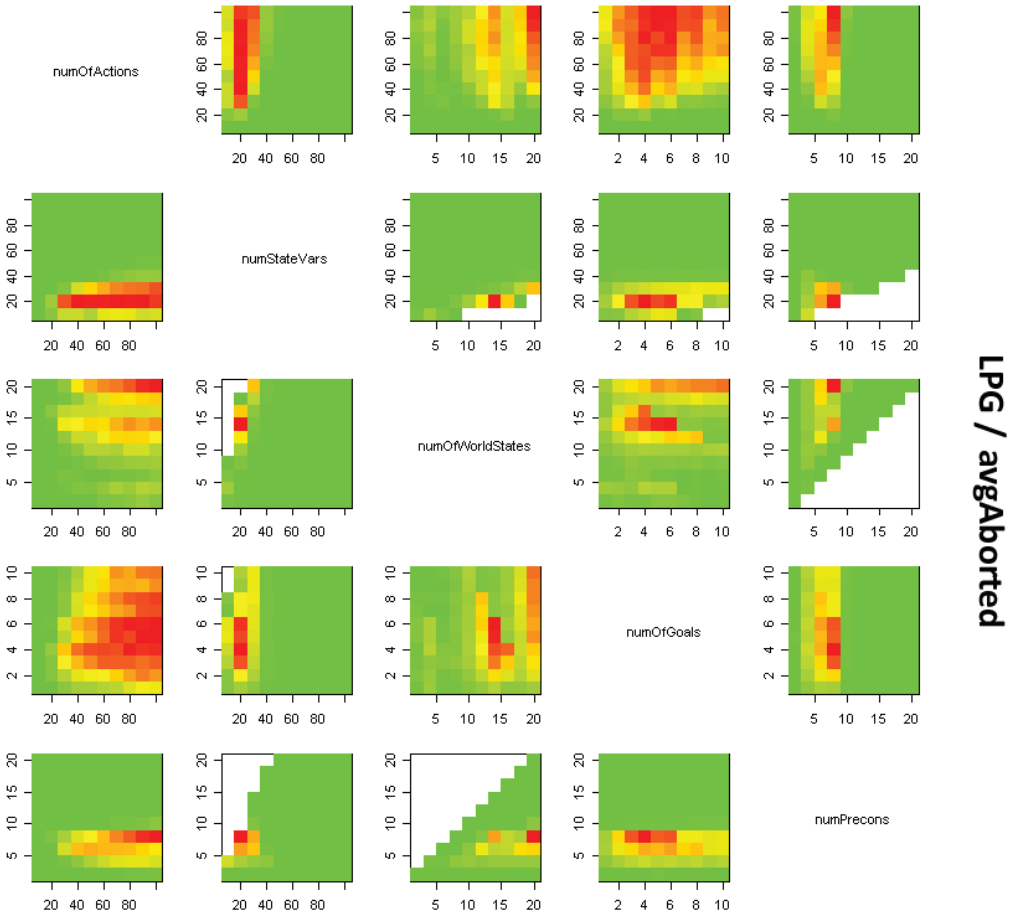

Abbildung D.7: Korrelationsmatrix der durchschnittlich abgebrochenen Läufe; LPG

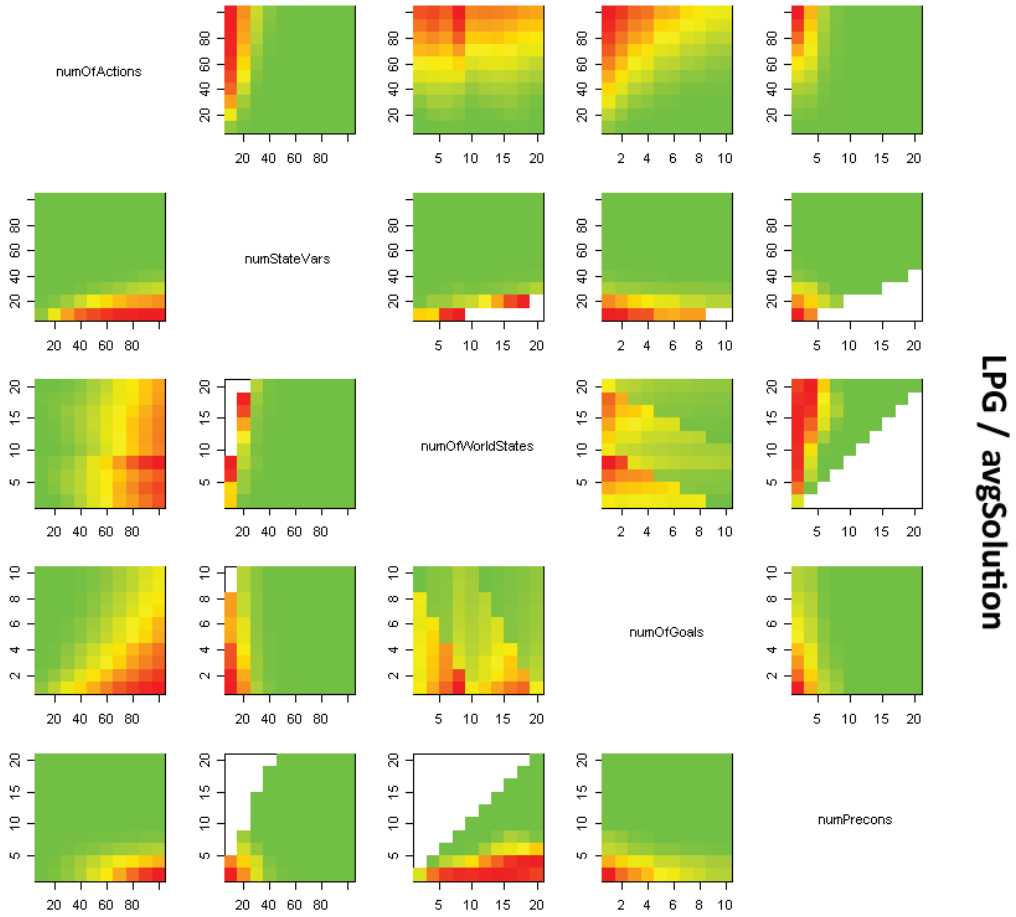

Abbildung D.8: Korrelationsmatrix der Anzahl der gefundenen Lösungen; LPG

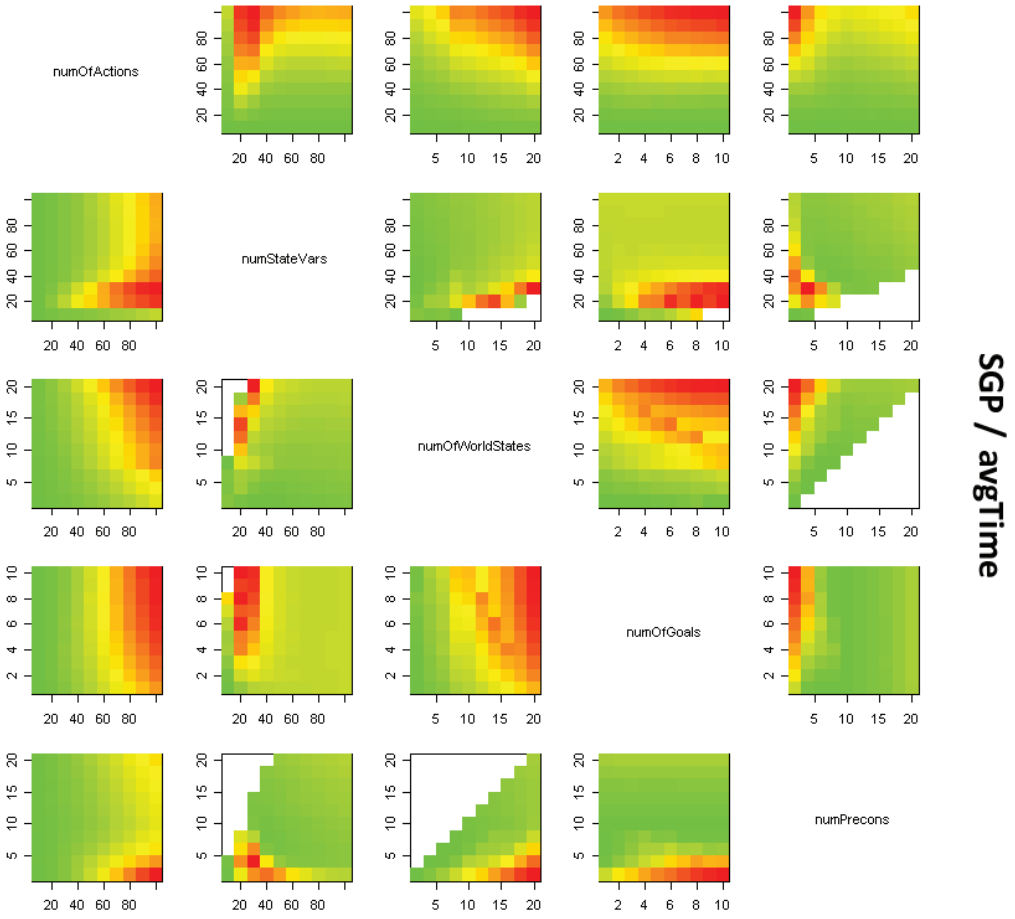

Abbildung D.9: Korrelationsmatrix der durchschnittlichen Planungsdauer; SGP

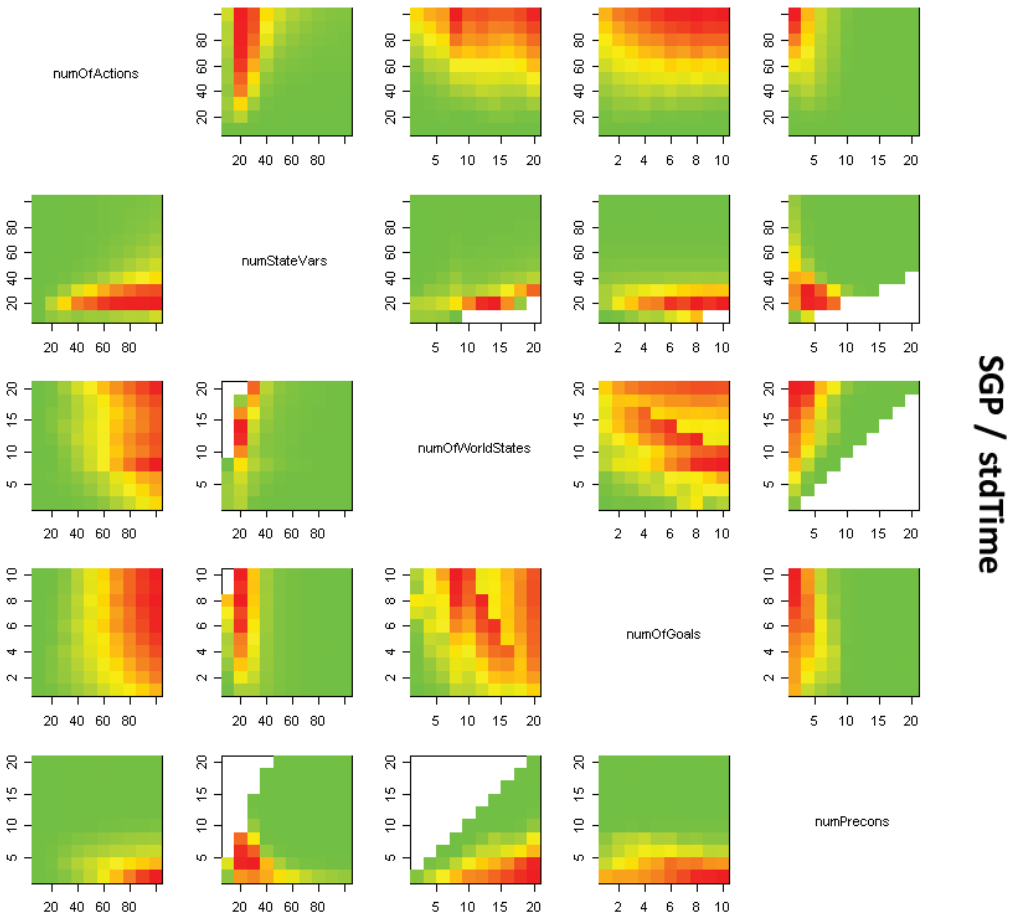

Abbildung D.10: Korrelationsmatrix der Standardabweichung der Planungsdauer; SGP

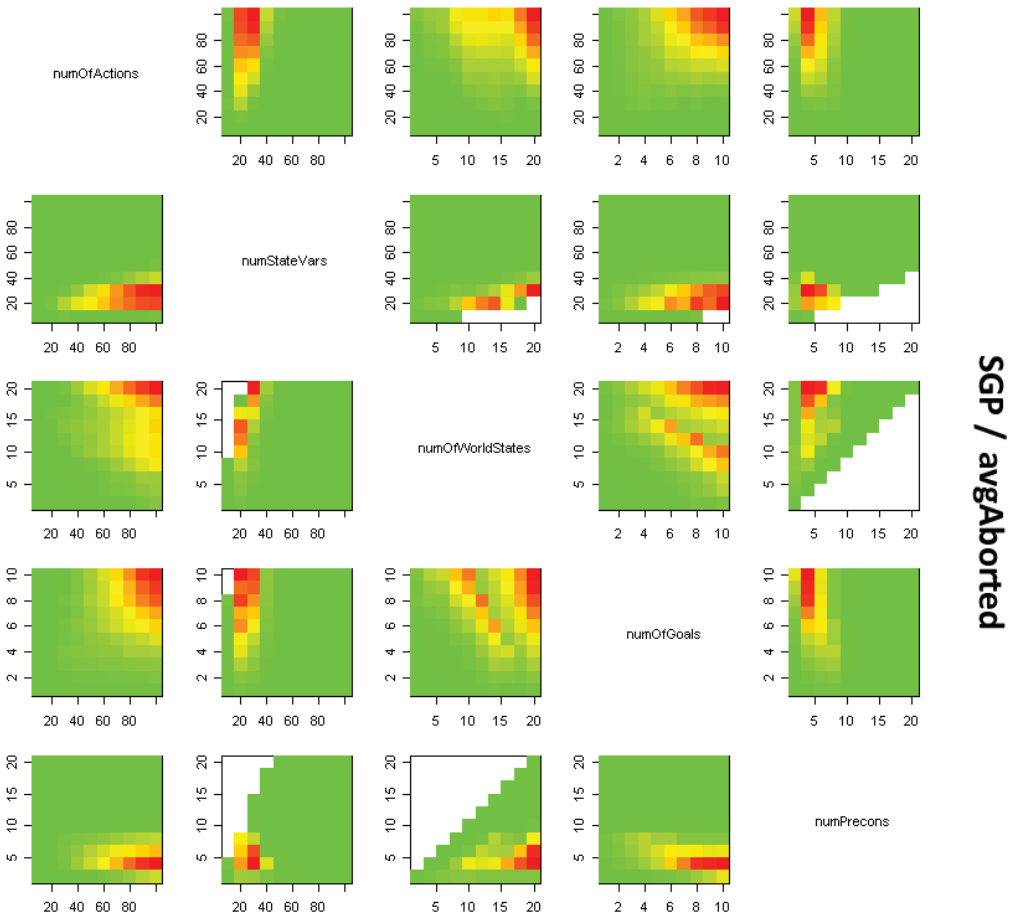

Abbildung D.11: Korrelationsmatrix der durchschnittlich abgebrochenen Läufe; SGP

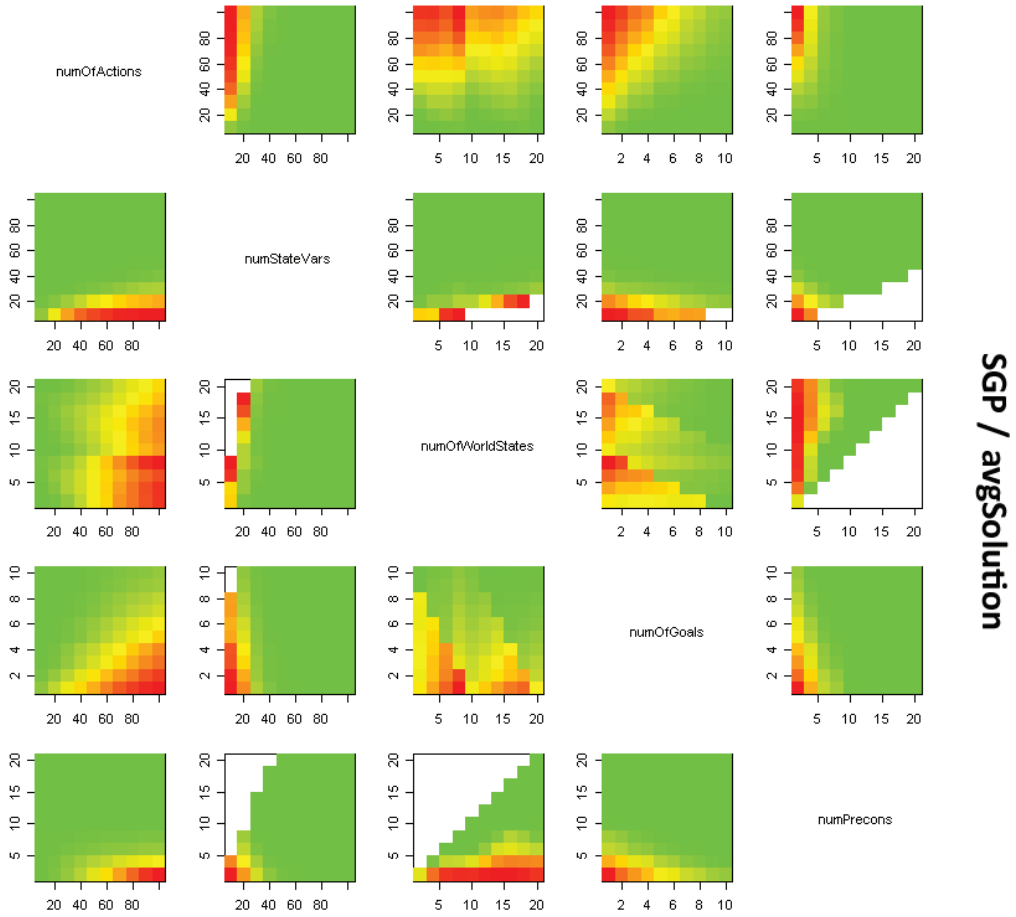

Abbildung D.12: Korrelationsmatrix der Anzahl der gefundenen Lösungen; SGP

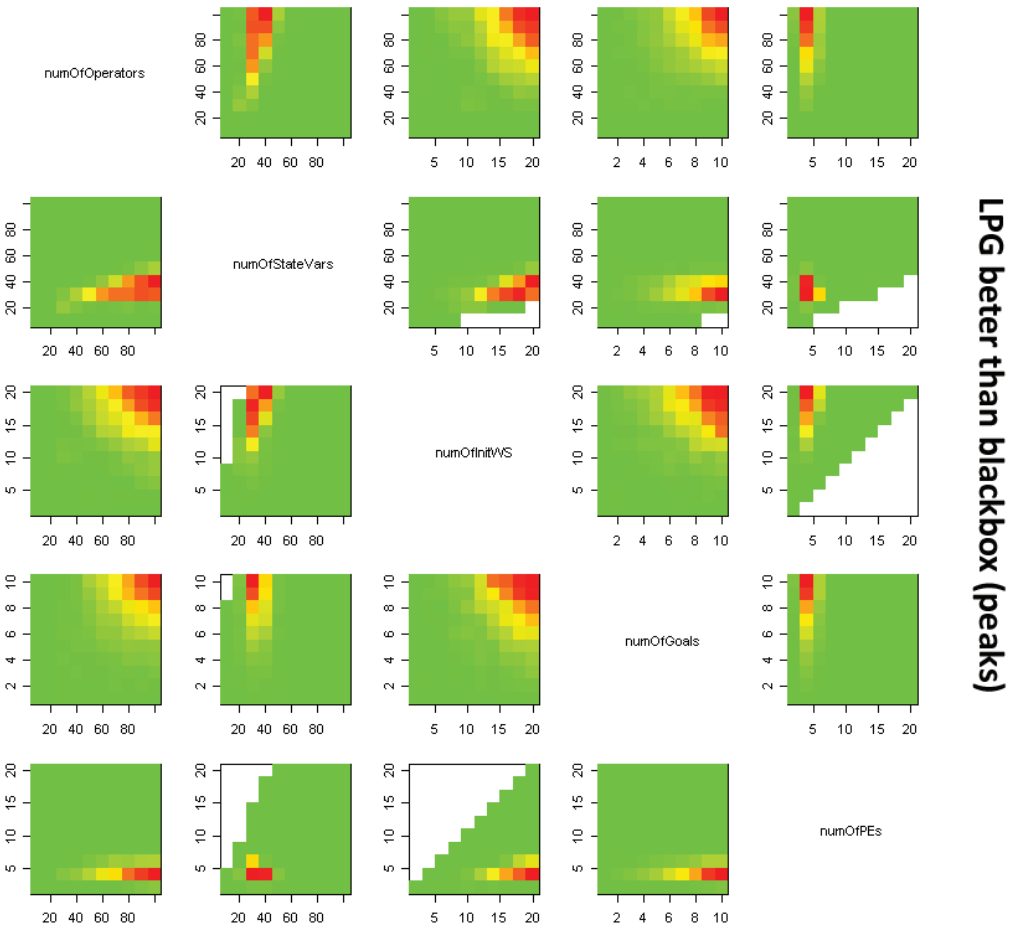

Abbildung D.13: Korrelationsmatrix der Dominanz von LPG gegenüber black-box

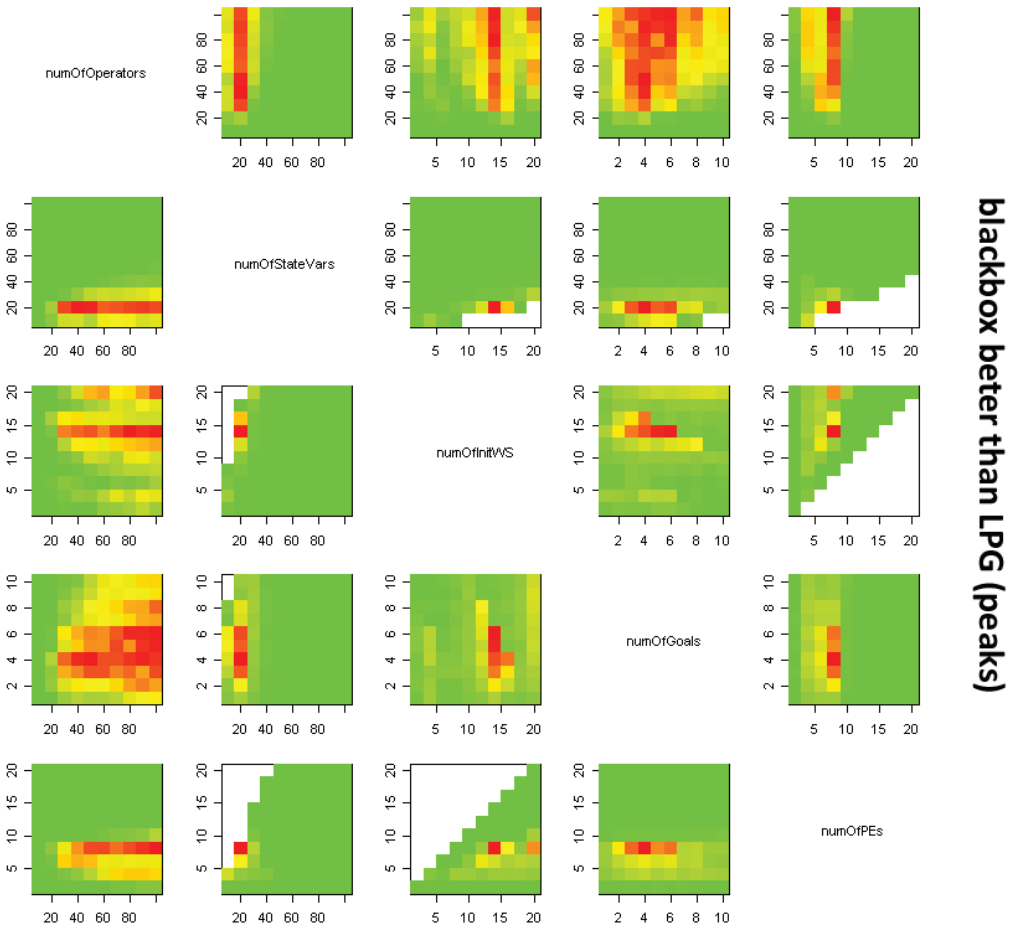

Abbildung D.14: Korrelationsmatrix der Dominanz von blackbox gegenüber LPG

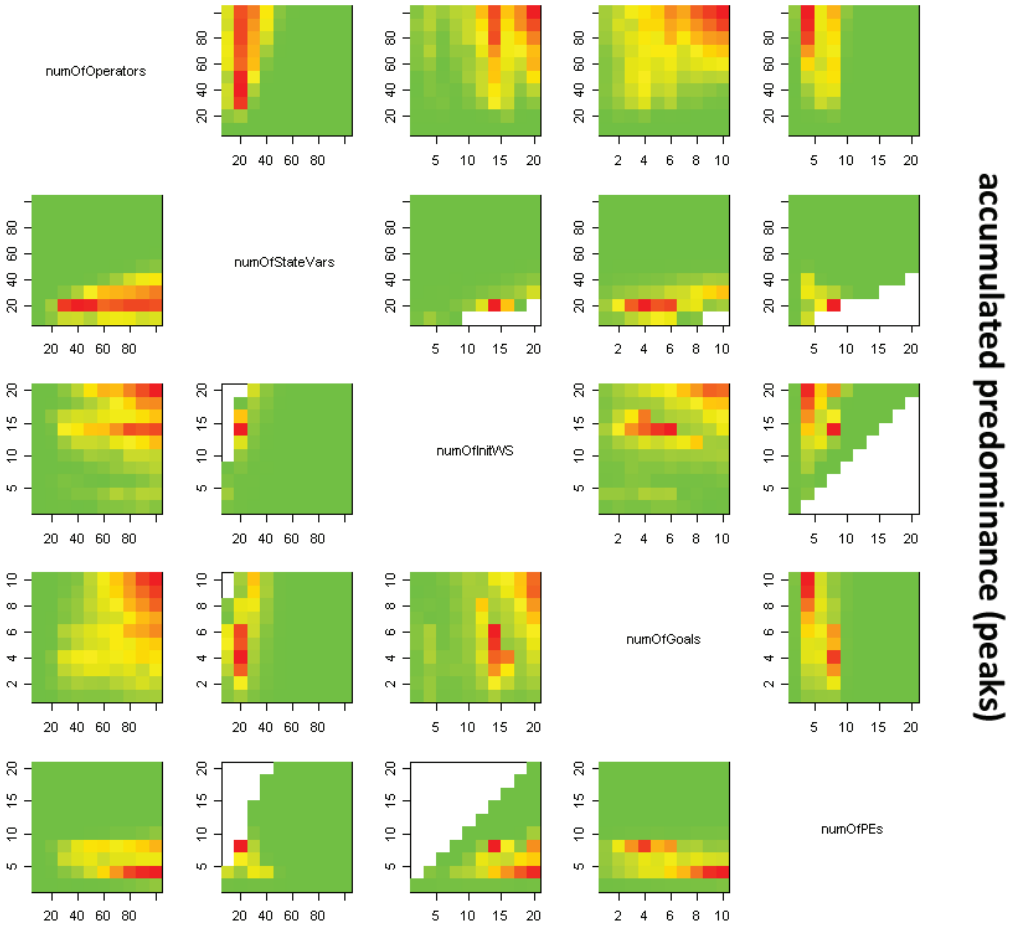

Abbildung D.15: Korrelationsmatrix der kumulierten Dominanz der Planer

Anhang E

Dichtefunktionen der Planungslaufzeiten der Beispielszenarien

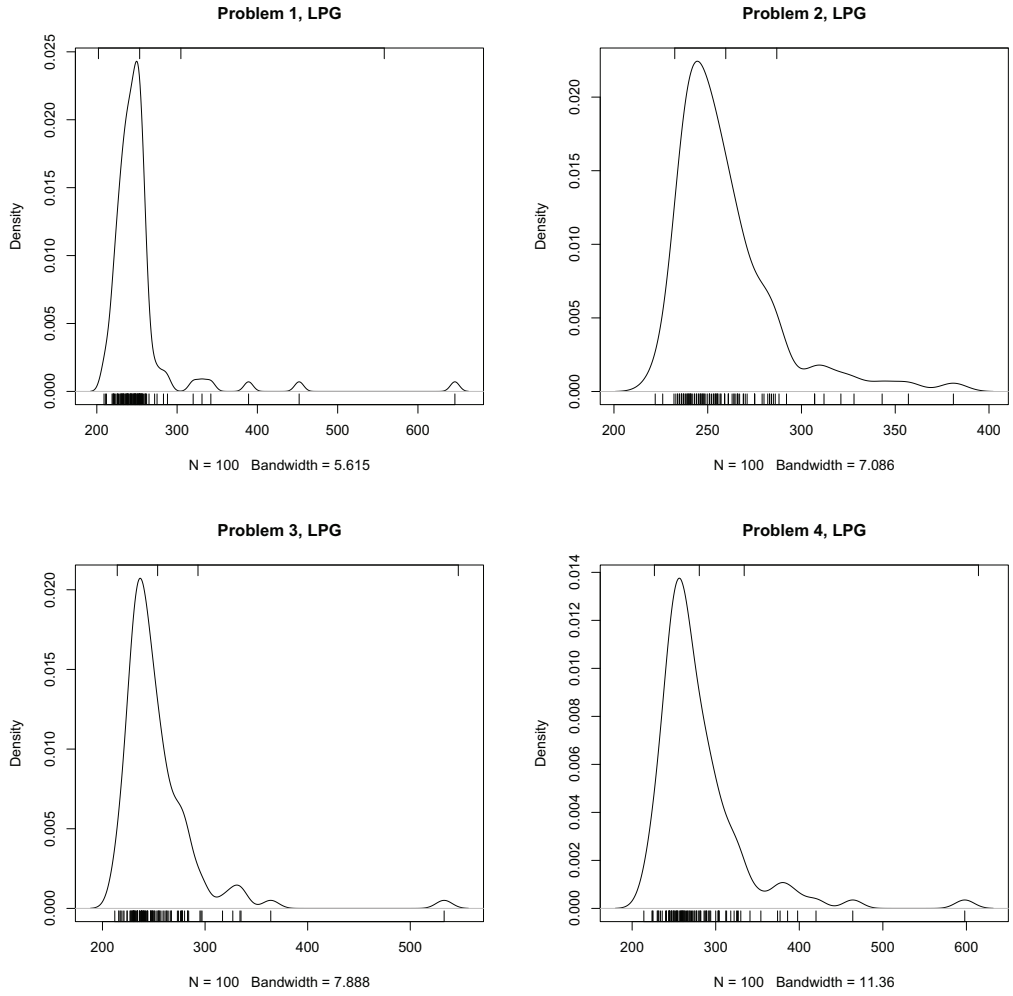

Abbildung E.1: Verteilung der Laufzeiten von LPG für die Probleme 1-4. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std abgetragen. Alle 100 Läufe pro Problem konnten dabei von LPG gelöst werden und ein Plan wurde erzeugt.

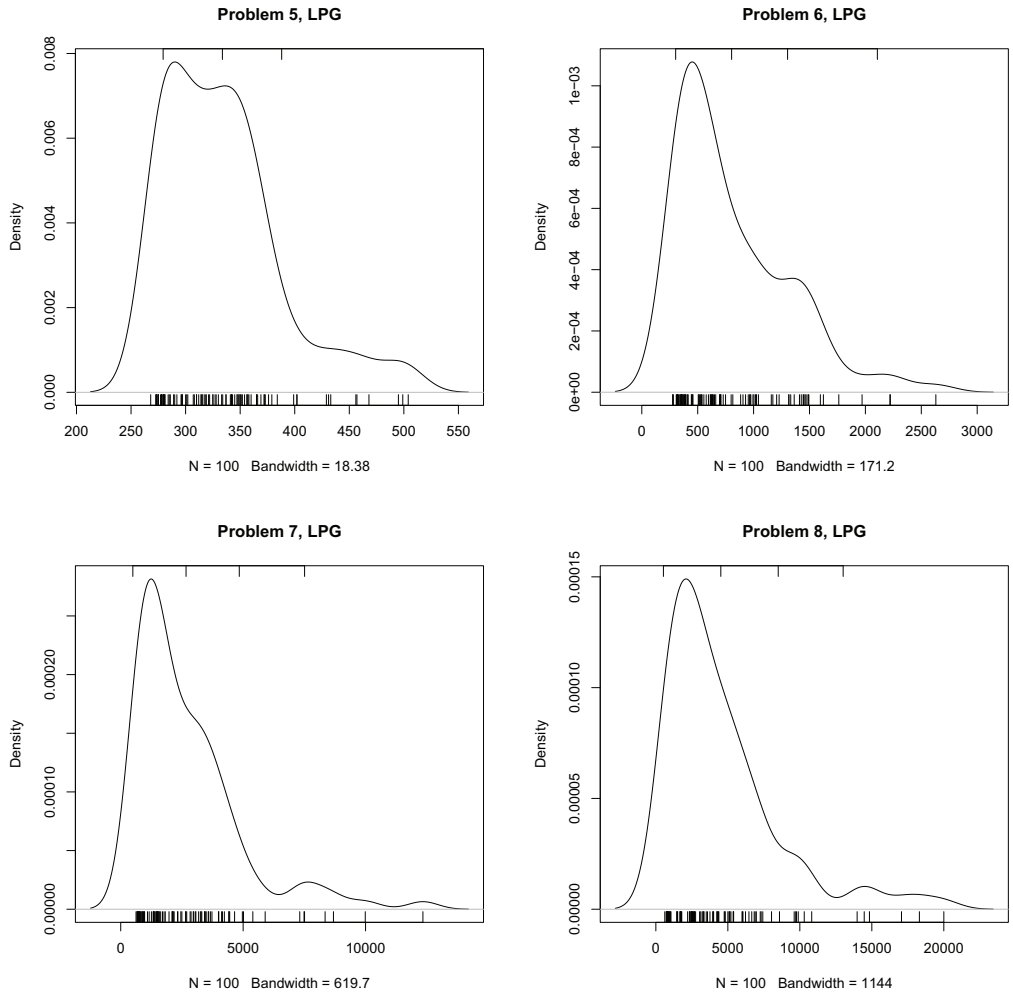

Abbildung E.2: Verteilung der Laufzeiten von LPG für die Probleme 5-8. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std (sofern sich dieser noch im Wertebereich der Laufzeiten befindet) abgetragen. Alle 100 Läufe pro Problem konnten dabei von LPG gelöst werden und ein Plan wurde erzeugt.

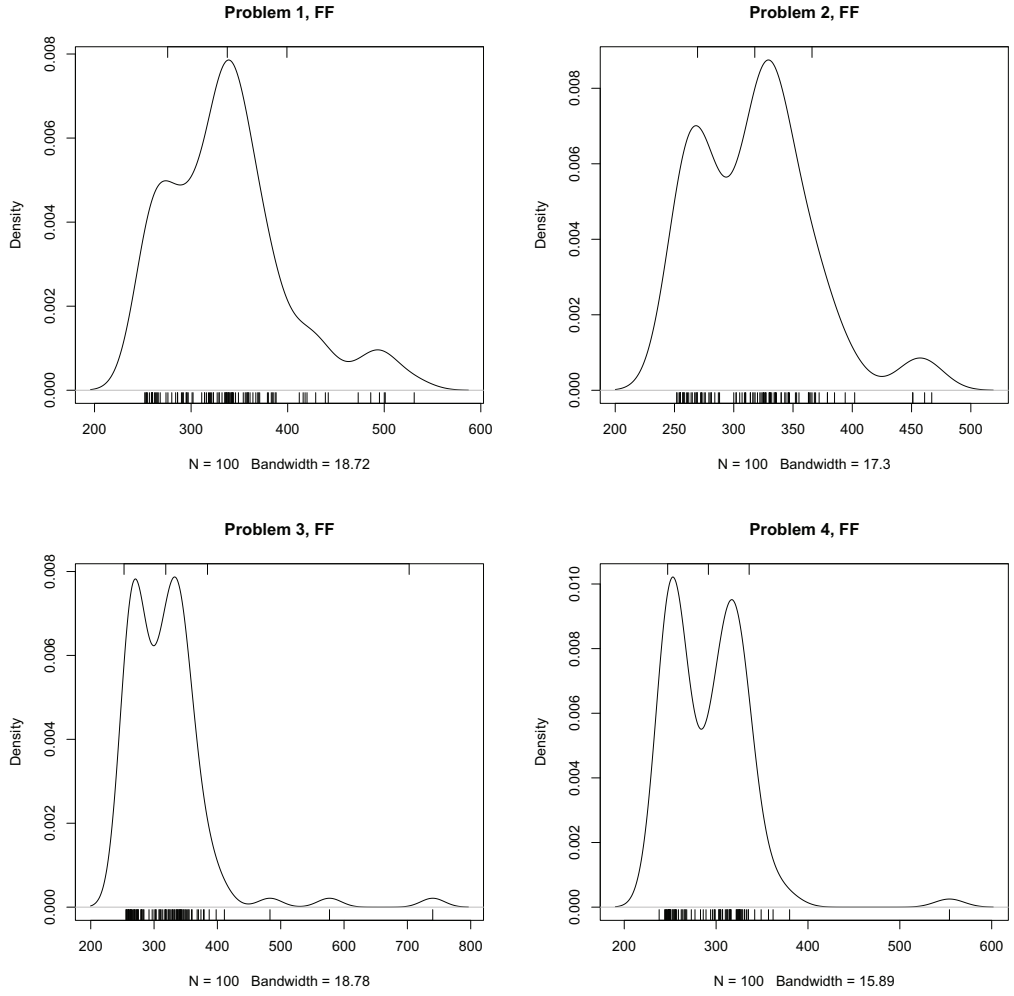

Abbildung E.3: Verteilung der Laufzeiten von FF für die Probleme 1-4. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std (sofern sich dieser noch im Wertebereich der Laufzeiten befindet) abgetragen. Alle 100 Läufe pro Problem konnten dabei von FF gelöst werden und ein Plan wurde erzeugt.

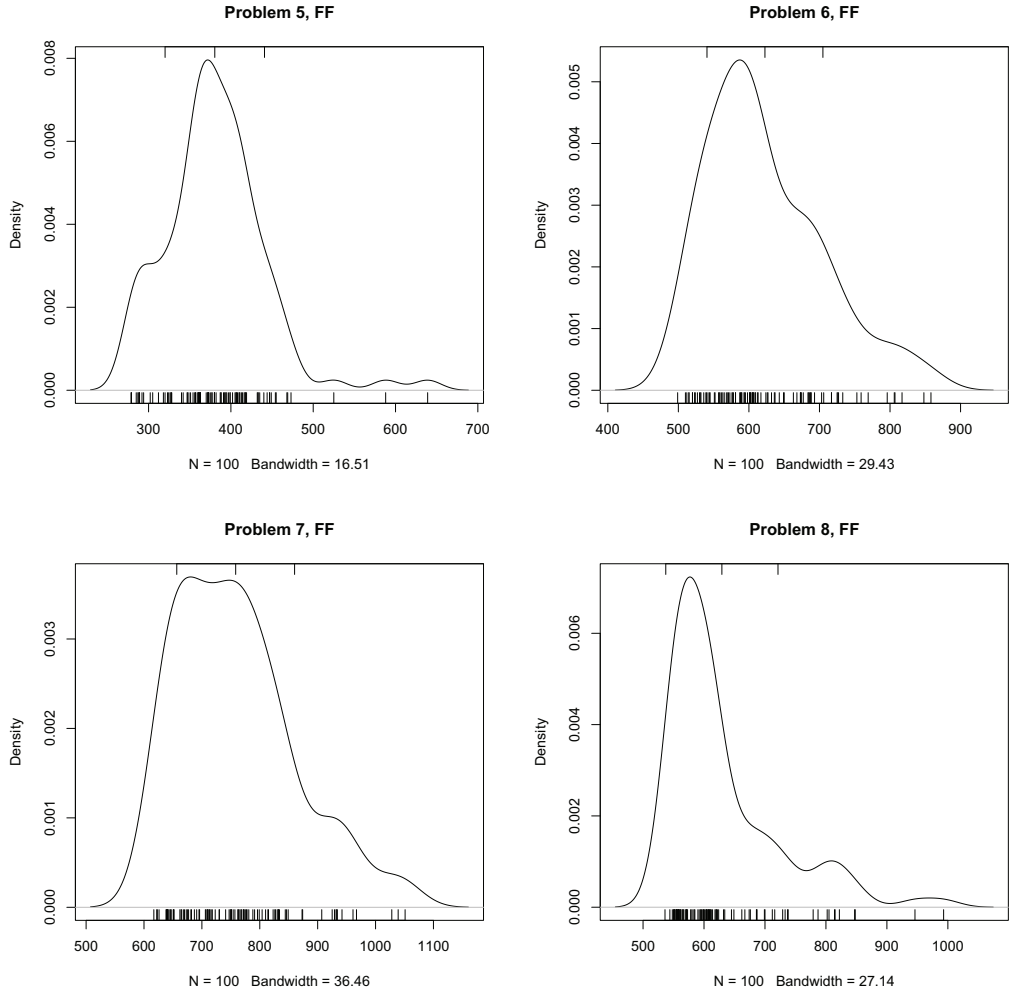

Abbildung E.4: Verteilung der Laufzeiten von FF für die Probleme 5-8. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std (sofern sich dieser noch im Wertebereich der Laufzeiten befindet) abgetragen. Alle 100 Läufe pro Problem konnten dabei von FF gelöst werden und ein Plan wurde erzeugt.

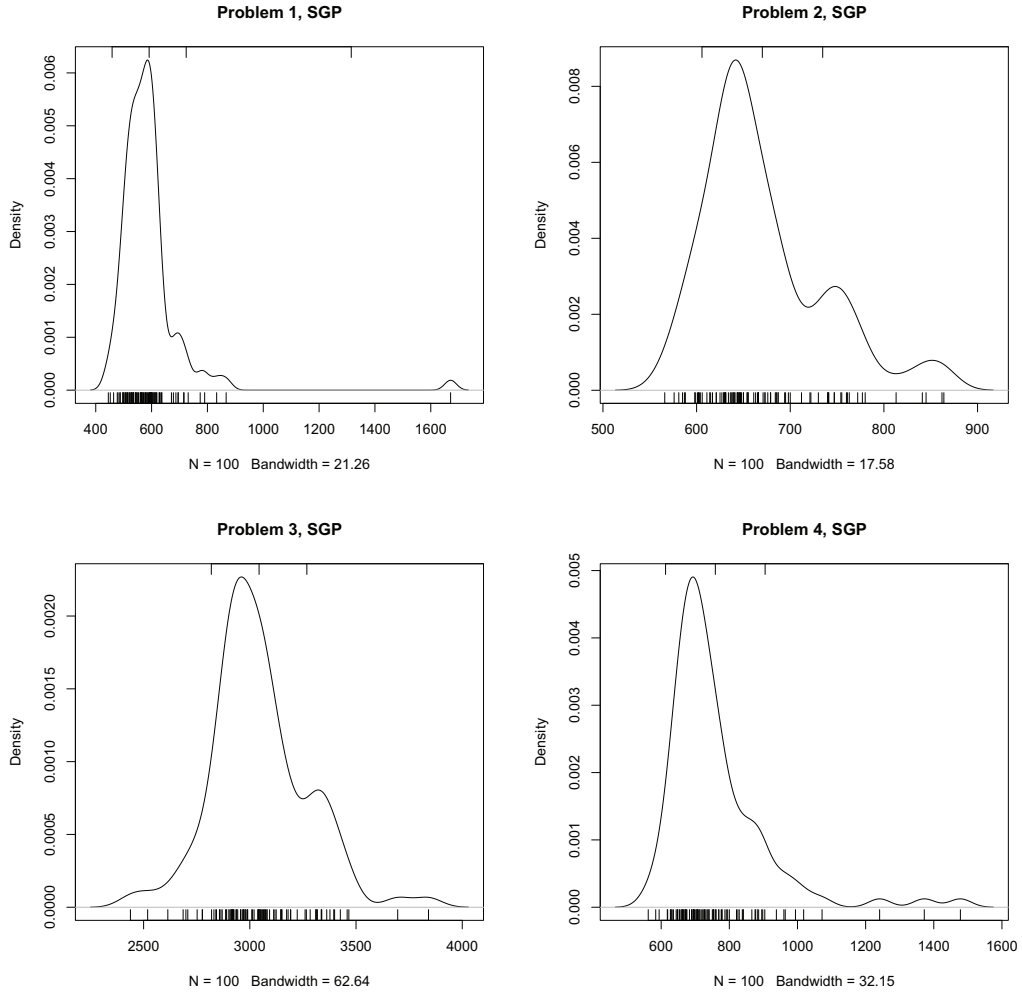

Abbildung E.5: Verteilung der Laufzeiten von SGP für die Probleme 1-4. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std (sofern sich dieser noch im Wertebereich der Laufzeiten befindet) abgetragen. Alle 100 Läufe pro Problem konnten dabei von SGP gelöst werden und ein Plan wurde erzeugt.

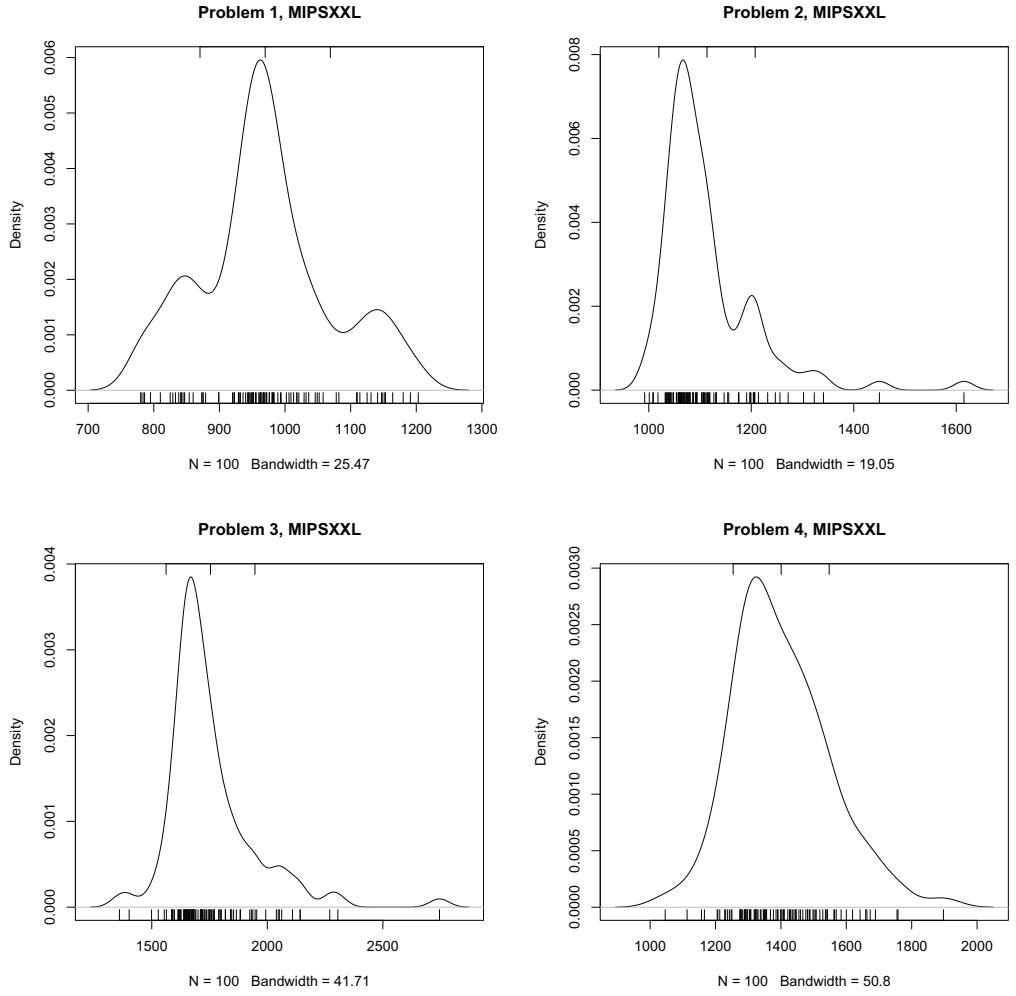

Abbildung E.6: Verteilung der Laufzeiten von MIPSXXL für die Probleme 1-4. Die Bandbreiten (Bandwidth) zur Glättung der Kennlinie sind für jedes Diagramm automatisch generiert. Am unteren Rand der Diagramme sind zur besseren Orientierung die diskreten Werte der Laufzeiten dargestellt. Am oberen Rand sind der Mittelwert (mean) zusätzlich und abzüglich der Standardvarianz (std) sowie der Wert 2mean+std (sofern sich dieser noch im Wertebereich der Laufzeiten befindet) abgetragen. Alle 100 Läufe pro Problem konnten dabei von MIPSXXL gelöst werden und ein Plan wurde erzeugt.

Literaturverzeichnis

[van der Aalst 1998] Aalst, Wil M. P. van der: The Application of Petri Nets to Workflow Management. In: *Journal of Circuits, Systems, and Computers* 8 (1998), Nr. 1, S. 21–66 23, 24

[van der Aalst u. a. 2003] Aalst, W.M.P. van der ; Hofstede, A.H.M. ter ; Kiepuszewski, B. ; Barros, A.P.: Workflow Patterns. In: *Distributed and Parallel Databases* 14 (2003), Nr. 1, S. 5–51 24

[Aarts und Encarnação 2006] Aarts, Emile ; Encarnação, José L.: Into Ambient Intelligence. In: Aarts, Emile (Hrsg.) ; Encarnação, José (Hrsg.): *True Visions - The Emergence of Ambient Intelligence*. Springer-Verlag, 2006, S. 321–337 5

[Agarwal u. a. 2008] Agarwal, Vikas ; Chafle, Girish ; Mittal, Sumit ; Srivastava, Biplav: Understanding approaches for web service composition and execution. In: *Compute '08: Proceedings of the 1st Bangalore annual Compute conference*. New York, NY, USA : ACM, 2008, S. 1–8. – ISBN 978-1-59593-950-0 14, 21, 25, 137, 138, 140

[Aggarwal u. a. 2004] Aggarwal, R. ; Verma, Kunal ; Miller, J. ; Milnor, W.: Constraint driven Web service composition in METEOR-S. In: *IEEE International Conference on Services Computing*, September 2004, S. 23–30 27, 32, 33

[Aiello u. a. 2002] Aiello, Marco ; Papazoglou, Mike P. ; Yang, Jian ; Carman, M. ; Pistore, Marco ; Serafini, Luciano ; Traverso, Paolo: A Request Language for Web-Services Based on Planning and Constraint Satisfaction. In: *Proceedings of the Third International Workshop on Technologies for E-Services: TES '02*. London, GB : Springer-Verlag, 2002, S. 76–85 56

[Alamri u. a. 2006] Alamri, Atif ; Eid, Mohamad ; Saddik, Abdulmotaleb E.: Classification of the state-of-the-art dynamic web services composition techniques. In: *International Journal of Web Grid Services* 2 (2006), Nr. 2, S. 148–166 14

[Alonso u. a. 2003] Alonso, Gustavo ; Casati, Fabio ; Kuno, Harumi ; Machiraju, Vijay: *Web Services - Concepts, Architectures and Applications*. Springer-Verlag Berlin Heidelberg, November 2003 13, 19, 20

[Amigoni u. a. 2005] Amigoni, F. ; Gatti, N. ; Pinciroli, C. ; Roveri, M.: What planner for ambient intelligence applications? In: *Systems, Man and Cybernetics, Part A, IEEE Transactions on* 35 (2005), Nr. 1, S. 7–21 23, 30, 98

[Andrews u. a. 2003] Andrews, Tony ; Curbera, Francisco ; Dholakia, Hitesh ; Goland, Yaron ; Klein, Johannes ; Leymann, Frank ; Liu, Kevin ; Roller, Dieter ; Smith, Dough ; Thatte, Satish ; Trickovic, Ivana ; Weerawarana, Sanjiva: *Business Process Execution Language of Web Services*. Online. May 2003. – URL `http://www.ibm.com/developerworks/library/specification/ws-bpel/` 24

[Armano u. a. 2003] Armano, Giuliano ; Cherchi, Giancarlo ; Vargiu, Eloisa: An Extension to PDDL for Hierarchical Planning. In: *Workshop on PDDL (ICAPS'03)*, Oktober 2003, S. 1–6 42

[Austin 1975] Austin, J. L.: *How to Do Things with Words: Second Edition (William James Lectures)*. 2. Harvard University Press, September 1975 85

[Baader u. a. 2003] Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The description logic handbook: theory, implementation, and applications*. New York, NY, USA : Cambridge University Press, 2003. – ISBN 0-521-78176-0 43

[Bacchus 2001] Bacchus, Fahiem: AIPS 2000 Planning Competition: The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems. In: *AI Magazine* 22 (2001), Nr. 3, S. 47–56 46

[Bacchus 2003] Bacchus, Fahiem: The power of modeling: a response to PDDL2.1. In: *Journal of Artificial Intelligence Research* 20 (2003), Nr. 1, S. 125–132 47

[Balzer u. a. 2004] Balzer, Steffen ; Liebig, Thorsten ; Wagner, Matthias: Pitfalls of OWL-S: a practical semantic web use case. In: *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA : ACM, 2004, S. 289–298 65

[Bartlett u. a. 2006] Bartlett, Marian S. ; Littlewort, Gwen ; Frank, Mark ; Lainscsek, Claudia ; Fasel, Ian ; Movellan, Javier: Fully Automatic Facial Action Recognition in Spontaneous Behavior. In: *International Conference on Automatic Face and Gesture Recognition*, 2006, S. 223–230 49

[Bedrax-Weiss u. a. 2003] Bedrax-Weiss, Tania ; McGann, Conor ; Ramakrishnan, Sailesh: Formalizing Resources for Planning. In: *In Proceedings of the ICAPS-03 Workshop on PDDL*, 2003 38

[ter Beek u. a. 2006] Beek, Maurice H. ter ; Bucciarone, Antonio ; Gnesi, Stefania: A survey on Service Composition Approaches: From Industrial Standards to Formal Methods / Istituto di Scienza e Tecnologie dell'Informazione, Consiglio Nazionale delle Ricerche. 2006 (2006-TR-15). – Forschungsbericht 14, 17, 18, 21

[Benatallah u. a. 2003] Benatallah, Boualem ; Dumas, Marlon ; Fauvet, Marie-Christine ; Rabhi, Fethi A.: Towards patterns of web services composition. In: *Patterns and skeletons for parallel and distributed computing*. London, GB : Springer-Verlag, 2003, S. 265–296 21

[Berners-Lee 2000] Berners-Lee, Tim: *Semantic Web on XML*. Talk at XML 2000. Dezember 2000 vii, 6

[Blankenburg u. a. 2008] Blankenburg, Bastian ; Botelho, Luis ; Calhau, Fábio ; Fernández, Alberto ; Klusch, Matthias ; Ossowski, Sascha: *CASCOM: Intelligent Service Coordination in the Semantic Web*. Kap. 11 Service Composition, S. 235–262, Birkhaeuser Verlag, 2008 27, 43, 51, 53, 58, 61, 85, 99

[Blum und Furst 1995] Blum, Avrim ; Furst, Merrick: Fast Planning Through Planning Graph Analysis. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1995, S. 1636–1642 41

[Blythe u. a. 2003] Blythe, Jim ; Deelman, Ewa ; Gil, Yolanda ; Kesselman, Carl ; Agarwal, Amit ; Mehta, Gaurang ; Vahi, Karan: The Role of Planning in Grid Computing. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'03)*, 2003, S. 9–13 34

[Bünnig und Cap 2007] Bünnig, Christian ; Cap, Clemens H.: Five Objectives to Diminish User Concerns About Privacy in Smart Environments. In: *Proceedings of MoMM2007 & iiWAS2007 Workshops*. Jakarta, Indonesia, Dezember 2007, S. 179–188 49

[Bohn u. a. 2006] Bohn, Hendrik ; Bobek, Andreas ; Golatowski, Frank: SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains. In: *Proceedings*

of 5th International Conference of Networking ICN'06. Morne, Mauritius, 2006, S. 43 63, 65

[Bonet und Geffner 2001] Bonet, Blai ; Geffner, Héctor: Planning as heuristic search. In: *Artificial Intelligence* 129 (2001), Nr. 1-2, S. 5–33 41

[Borgida und Brachman 2003] Borgida, Alex ; Brachman, Ronald J.: *The description logic handbook: theory, implementation, and applications*. Kap. Conceptual modeling with description logics, S. 349–372. New York, NY, USA : Cambridge University Press, 2003 81

[Box u. a. 2000] Box, Don ; Ehnebuske, David ; Kakivaya, Gopal ; Layman, Andrew ; Mendelsohn, Noah ; Frystyk Nielsen, Henrik ; Thatte, Satish ; Winer, Dave: *Simple Object Access Protocol (SOAP) 1.1*. World Wide Web Consortium Note. Mai 2000 13

[Bronsted u. a. 2007] Bronsted, Jeppe ; Hansen, Klaus M. ; Ingstrup, Mads: A Survey of Service Composition Mechanisms in Ubiquitous Computing. In: *UbiComp 2007 Workshop Proceedings* (2007), S. 87–92 14

[Bruijn und Heymans 2008] Bruijn, Jos d. ; Heymans, Stijn: On the Relationship between Description Logic-based and F-Logic-based Ontologies. In: *Fundamenta Informaticae* 82 (2008), Nr. 3, S. 213–236 62, 64

[Brumitt u. a. 2000] Brumitt, Barry ; Meyers, Brian ; Krumm, John ; Kern, Amanda ; Shafer, Steven: EasyLiving: Technologies for Intelligent Environments. In: Thomas, Peter J. (Hrsg.) ; Gellersen, Hans-Werner (Hrsg.): *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing* Bd. 1927. London, UK : Springer-Verlag, September 2000, S. 12–29 75

[Burghardt und Kirste 2007] Burghardt, Christoph ; Kirste, Thomas: Inferring intentions in generic context-aware systems. In: (Ojala, 2007), S. 50–54 57

[Burstein u. a. 2002] Burstein, Mark H. ; Hobbs, Jerry R. ; Lassila, Ora ; Martin, David ; McDermott, Drew V. ; McIlraith, Sheila A. ; Narayanan, Srini ; Paolucci, Massimo ; Payne, Terry R. ; Sycara, Katia P.: DAML-S: Web Service Description for the Semantic Web. In: *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*. London, GB : Springer-Verlag, 2002, S. 348–363 27

[Bylander 1992] Bylander, Tom: Complexity results for extended planning. In: *Proceedings of the first international conference on Artificial intelligence planning systems*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1992, S. 20–27 99

[Bylander 1996] Bylander, Tom: A probabilistic analysis of propositional STRIPS planning. In: *Artificial Intelligence* 81 (1996), S. 241–271 99, 100, 101, 102

[Carman u. a. 2003] Carman, Mark ; Serafini, Luciano ; Traverso, Paolo: Web Service Composition as Planning. In: *Workshop on Planning for Web Services*. Trento, Italy, June 2003 22

[Casati und Shan 2001] Casati, Fabio ; Shan, Ming-Chien: Dynamic and adaptive composition of e-services. In: *Information Systems* 26 (2001), Nr. 3, S. 143 – 163. – ISSN 0306-4379 17, 26, 32, 33

[Cass u. a. 2000] Cass, Aaron G. ; Lerner, Barbara S. ; Sutton, Stanley M. ; McCall, Eric K. ; Wise, Alexander ; Osterweil, Leon J.: Little-JIL/Juliette: a process definition language and interpreter. In: *ICSE '00: Proceedings of the 22nd international conference on Software engineering*. New York, NY, USA : ACM, 2000, S. 754–757 23, 24

[Chakraborty u. a. 2005] Chakraborty, Dipanjan ; Joshi, Anupam ; Finin, Tim ; Yesha, Yelena: Service Composition for Mobile Environments. In: *Mobile Networks and Applications* 10 (2005), August, Nr. 4, S. 435–451 29, 32, 33, 137, 140

[Chakraborty u. a. 2002] Chakraborty, Dipanjan ; Perich, Filip ; Joshi, Anupam ; Finin, Timothy W. ; Yesha, Yelena: A Reactive Service Composition Architecture for Pervasive Computing Environments. In: *Proceedings of the IFIP TC6/WG6.8 Working Conference on Personal Wireless Communications: PWC '02*. Deventer, Niederlande : Kluwer, B.V., 2002, S. 53–62 137

[Chan u. a. 2007] Chan, Hei ; Fern, Alan ; Ray, Soumya ; Wilson, Nick ; Ventura, Chris: Online Planning for Resource Production in Real-Time Strategy Games. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) 2007*, 2007 117

[Cheeseman u. a. 1991] Cheeseman, Peter ; Kanefsky, Bob ; Taylor, William M.: Where the really hard problems are. In: *In J. Mylopoulos and R. Reiter (Eds.), Proceedings of 12th International Joint Conference on AI (IJCAI-91), Volume 1*, Morgan Kauffman, 1991, S. 331–337 99

[Chen u. a. 2004] Chen, Harry ; Perich, Filip ; Finin, Tim ; Joshi, Anupam: SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In: *International Conference on Mobile and Ubiquitous Systems: Networking and Services* (2004), August, S. 258–267 4, 89

[Driscoll und Mensch 2009] Driscoll, Dan ; Mensch, Antoine: *Devices Profile for Web Services Version 1.1.* OASIS Standard. Juli 2009 63

[Dupuis u. a. 2007] Dupuis, Remi ; Pierson, Jerome ; Vallee, Mathieu: *Flexible & Dynamic Device Usage based on Software Agents and Semantic Web Technologies in an Ambient Home Environment.* Demonstration at Ubicomp2007. 2007 98

[Durfee 2001] Durfee, Edmund H.: Distributed problem solving and planning. In: *Mutli-agents systems and applications.* New York, NY, USA : Springer-Verlag New York, Inc., 2001, S. 118–149 141, 142, 143

[Dustdar und Schreiner 2005] Dustdar, Schahram ; Schreiner, Wolfgang: A survey on web services composition. In: *International Journal of Web Grid Services* 1 (2005), Nr. 1, S. 1–30 12, 14

[Edelkamp 2003] Edelkamp, Stefan: Taming numbers and durations in the model checking integrated planning system. In: *Journal of Artificial Intelligence Research* 20 (2003), Nr. 1, S. 195–238 40, 41

[Edelkamp und Hoffmann 2004] Edelkamp, Stefan ; Hoffmann, Jörg: *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition.* 2004 44, 88, 157

[Edelkamp u. a. 2006] Edelkamp, Stefan ; Jabbar, Shahid ; Nazih, Mohammed: Large-Scale Optimal PDDL3 Planning with MIPS-XXL. In: *5th International Planning Competition Booklet (IPC-2006)*, 2006 113

[Edwards u. a. 2005] Edwards, W. K. ; Newman, Mark W. ; Sedivy, Jana Z. ; Smith, Trevor F.: Bringing Network Effects to Pervasive Spaces. In: *IEEE Pervasive Computing* 4 (2005), Nr. 3, S. 15–17 29

[Eisenberg und Melton 1998] Eisenberg, Andrew ; Melton, Jim: Standards in practice. In: *SIGMOD Record* 27 (1998), Nr. 3, S. 53–58 47

[Erl 2006] Erl, Thomas: *Service-oriented architecture: concepts, technology, and design.* Prentice Hall PTR, 2006 13

[Erol u. a. 1994] Erol, Kutluhan ; Hendler, James ; Nau, Dana S.: HTN Planning: Complexity and Expressivity. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* Bd. 2. Seattle, Washington, USA : AAAI Press/MIT Press, 1994, S. 1123–1128 42

[Esler u. a. 1999] Esler, Mike ; Hightower, Jeffrey ; Anderson, Thomas E. ; Borriello, Gaetano: Next Century Challenges: Data-Centric Networking for Invisible Computing. In: *MOBICOM*, 1999, S. 256–262 75

[Fauvet u. a. 2001] Fauvet, Marie-Christine ; Dumas, Marlon ; Benatallah, Boualem ; Paik, Hye-Young: Peer-to-Peer Traced Execution of Composite Services. In: *TES '01: Proceedings of the Second International Workshop on Technologies for E-Services*. London, GB : Springer-Verlag, 2001, S. 103–117 144

[Fensel u. a. 2003] Fensel, Dieter (Hrsg.) ; Hendler, James A. (Hrsg.) ; Liebermann, Henry (Hrsg.): *Spinning the semantiv Web*. New Ed March 2005. The MIT Press, 2003 6

[Fielding u. a. 1999] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *RFC 2616 Hypertext Transfer Protocol - HTTP/1.1*. IETF RFC. Juni 1999 14

[Fikes und Nilsson 1971] Fikes, Richard E. ; Nilsson, Nils J.: Strips: A new approach to the application of theorem proving to problem solving. In: *Artificial Intelligence* 2 (1971), Nr. 3-4, S. 189–208 41

[Fink u. a. 1998] Fink, Eugene ; Polya, G. ; It, How To S.: How to Solve It Automatically: Selection Among Problem-Solving Methods. In: *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, AAAI Press, 1998, S. 128–136 147

[Fluegge u. a. 2006] Fluegge, Matthias ; Santos, Ivo J. G. ; Tizzo, Neil P. ; Madeira, Edmundo R. M.: Challenges and techniques on the road to dynamically compose web services. In: *ICWE '06: Proceedings of the 6th international conference on Web engineering*. New York, NY, USA : ACM, 2006, S. 40–47 14, 15, 16, 17, 18, 24

[Foster u. a. 2003] Foster, Howard ; Kramer, Jeff ; Magee, Jeff ; Uchitel, Sebastian: Model-based Verification of Web Service Compositions. In: *18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003 41

[Fox und Long 2003] Fox, Maria ; Long, Derek: PDDL2.1: An extension to PDDL for expressing temporal planning domains. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 2003 40, 85, 88

[Friedewald u. a. 2005] Friedewald, Michael ; Costa, Olivier D. ; Punie, Yves ; Alahuhta, Petteri ; Heinonen, Sirkka: Perspectives of ambient intelligence in the home environment. In: *Telematics and Informatics* 22 (2005), Nr. 3, S. 221–238 5

[Fujii und Suda 2004] Fujii, Keita ; Suda, Tatsuya: Dynamic service composition using semantic information. In: *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*. New York, NY, USA : ACM, 2004, S. 39–48 69

[Garlan u. a. 2002] Garlan, David ; Siewiorek, Daniel P. ; Steenkiste, Peter: Project Aura: Toward Distraction-Free Pervasive Computing. In: *IEEE Pervasive Computing* 1 (2002), S. 22–31 71

[Gerber u. a. 2005] Gerber, Andreas ; Klusch, Matthias ; Schmidt, Marcus: Semantic Web Service Composition Planning with OWLS-Xplan. In: *Proceedings 1st International AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA*, 2005 61

[Gerevini und Serina 2002] Gerevini, Alfonso ; Serina, Ivan: LPG: A Planner based on Local Search for Planning Graphs. In: *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02)*. Toulouse, France : AAAI Press, 2002 68, 102

[Gerevini u. a. 2009] Gerevini, Alfonso E. ; Haslum, Patrik ; Long, Derek ; Saetti, Alessandro ; Dimopoulos, Yannis: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. In: *Artificial Intelligence* 173 (2009), Nr. 5-6, S. 619–668 41, 46, 95, 96

[Ghallab u. a. 1998] Ghallab, Malik ; Howe, Adele ; Knoblock, Craig ; McDermott, Drew ; Ram, Ashwin ; Veloso, Manuela ; Weld, Daniel ; Wilkins, David: *PDDL - The Planning Domain Definition Language*, Juli 1998 22, 47

[Ghallab u. a. 2004] Ghallab, Malik ; Nau, Dana ; Traverso, Paolo: *Automated Planning, theory and practice*. Morgan Kaufmann, 2004 36, 37, 39, 41

[Giunchiglia und Traverso 2000] Giunchiglia, Fausto ; Traverso, Paolo: Planning as Model Checking. In: *ECP '99: Proceedings of the 5th European Conference on Planning*. London, GB : Springer-Verlag, 2000, S. 1–20 41

[Grimm 2004] Grimm, Robert: One.world: Experiences with a Pervasive Computing Architecture. In: *IEEE Pervasive Computing* 3 (2004), Nr. 3, S. 22–30 30

[Grimm u. a. 2004] Grimm, Robert ; Davis, Janet ; Lemar, Eric ; Macbeth, Adam ; Swanson, Steven ; Anderson, Thomas ; Bershad, Brian ; Borriello, Gaetano ; Gribble, Steven ; Wetherall, David: System support for pervasive applications. In: *ACM Transactions on Computer Systems* 22 (2004), Nr. 4, S. 421–486 30

[Hadley 2009] Hadley, Aerc J.: *Web Application Description Language (WADL)*. W3C Member Submission. August 2009. – URL `http://www.w3.org/Submission/wadl/` 60

[Hamadi und Benatallah 2003] Hamadi, Rachid ; Benatallah, Boualem: A Petri net-based model for web service composition. In: *ADC '03: Proceedings of the 14th Australasian database conference*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2003, S. 191–200 23

[Hatzi u. a. 2009] Hatzi, Ourania ; Meditskos, Georgios ; Vrakas, Dimitris ; Bassiliades, Nick ; Anagnostopoulos, Dimosthenis ; Vlahavas, Ioannis: PORSCE II: Using Planning for Semantic Web Service Composition. In: *Proceedings of International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS)*, 2009 61

[Heider 2009] Heider, Thomas: *A Unified Distributed System Architecture for Goal-based Interaction with Smart Environments*, Universität Rostock, Dissertation, April 2009 viii, 89, 90, 139, 140

[Heider und Kirste 2002] Heider, Thomas ; Kirste, Thomas: Supporting goal based interaction with dynamic intelligent environments. In: *ECAI*, 2002, S. 596–600 54, 151, 175, 176

[Heider und Kirste 2005] Heider, Thomas ; Kirste, Thomas: Smart Environments and Self-Organizing Appliance Ensembles. In: *Mobile Computing and Ambient Intelligence*, 2005 44

[Herfet und Kirste 2002] Herfet, Thorsten ; Kirste, Thomas: Embassi White Paper / Fraunhofer IGD Rostock. August 2002 (V1.1). – Forschungsbericht 139

[Hoffmann und Edelkamp 2005] Hoffmann, Jörg ; Edelkamp, Stefan: The Deterministic Part of IPC-4: An Overview. In: *Journal of Artificial Intelligence Research (JAIR)* 24 (2005), S. 519–579 46, 96

[Hoffmann und Nebel 2001] Hoffmann, Jörg ; Nebel, Bernhard: The FF planning system: fast plan generation through heuristic search. In: *Journal of Artificial Intelligence Research (JAIR)* 14 (2001), Nr. 1, S. 253–302. – ISSN 1076-9757 113

[Hogg u. a. 2008] Hogg, Chad ; Munoz-Avila, Héctor ; Kuter, Ugur: HTN-MAKER: learning HTNs with minimal additional knowledge engineering required. In: *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, AAAI Press, 2008, S. 950–956. – ISBN 978-1-57735-368-3 42

[Howe und Dahlman 2002] Howe, Adele E. ; Dahlman, Eric: A critical assessment of benchmark comparison in planning. In: *Journal of Artificial Intelligence Research (JAIR)* 17 (2002), S. 1–33 96, 103, 117, 118, 129, 133

[Howe u. a. 1999] Howe, Adele E. ; Dahlman, Eric ; Hansen, Christopher ; Scheetz, Michael ; Mayrhauser, Anneliese V.: Exploiting competitive planner performance. In: *Proceedings of the Fifth European Conference on Planning*, Springer-Verlag, 1999, S. 62–72 41, 125, 147, 174

[Hull und Su 2005] Hull, Richard ; Su, Jianwen: Tools for composite web services: a short overview. In: *SIGMOD Rec.* 34 (2005), Nr. 2, S. 86–95 14, 16, 17, 21

[IEEE 1990] IEEE: *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York: Institute of Electrical and Electronics Engineers (Veranst.), 1990 4

[Ilghami und Nau 2002] Ilghami, Okhtay ; Nau, Dana S.: CaMeL: Learning method preconditions for HTN planning. In: *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, AAAI Press, 2002, S. 131–141 42

[ISO 2005] ISO: *Unified Modeling Language (UML) Version 1.4.2*. ISO/IEC 19501:2005. 2005 24

[ISO 2008] ISO: *Information technology – UPnP Device Architecture – Part 1: UPnP Device Architecture Version 1.0*. ISO/IEC Standard 29341-1:2008. November 2008 62, 65

[Johanson u. a. 2003] Johanson, Brad ; Winograd, Terry ; Fox, Armando: Interactive Workspaces. In: *Computer* 36 (2003), Nr. 4, S. 99–101 29

[Kautz und Selman 1996] Kautz, Henry ; Selman, Bart: Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In: *Conference on Artificial Intelligence*, AAAI Press, 1996, S. 1194–1201 41

[Kautz und Selman 1998] Kautz, Henry ; Selman, Bart: BLACKBOX: A new approach to the application of theorem proving to problem solving. In: *Working notes of the Workshop on Planning as Combinatorial Search*, 1998, S. 58–60 41, 102

[Kifer und Lausen 1989] Kifer, Michael ; Lausen, Georg: F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In: *SIGMOD Rec.* 18 (1989), Nr. 2, S. 134–146 61

[Kirste 2006] Kirste, Thomas: Smart Environments. In: Aarts, Emile (Hrsg.) ; Encarnação, José (Hrsg.): *True Visions - The Emergence of Ambient Intelligence*. Berlin Heidelberg : Springer-Verlag, 2006, S. 321–337 3

[Kistler u. a. 2008] Kistler, R. ; Knauth, S. ; Klapproth, A.: UPnP in integrated home- and building networks. In: *IEEE International Workshop on Factory Communication Systems: WFCS 2008*, May 2008, S. 235–238 62, 63

[Klein 2004] Klein, Michael: *Handbuch zur DIANE Service Description*, Dezember 2004. – URL `http://hnsp.inf-bb.uni-jena.de/DIANE/docs/DSD-Cookbook.pdf` 21, 62, 65

[Klein u. a. 2005] Klein, Michael ; König-Ries, Birgitta ; Mussig, Michael: What is needed for semantic service descriptions? A proposal for suitable language constructs. In: *International Journal of Web Grid Services* 1 (2005), Nr. 3/4, S. 328–364 21, 22, 52, 62, 89

[Klusch 2008] Klusch, Matthias: *CASCOM: Intelligent Service Coordination in the Semantic Web*. Kap. 3 Semantic Web Service Description, S. 31–57, Birkhaeuser-Verlag, 2008 18, 20, 58, 64

[Klusch und Zhing 2008] Klusch, Matthias ; Zhing, Xiguo: Deployed Semantic Services for the Common User of the Web: A Reality Check. In: *International Conference on Semantic Computing* 0 (2008), S. 347–353 27, 32, 33, 34, 65

[Koehler 1998] Koehler, Jana: Planning under Resource Constraints. In: *Proceeding on European Conference on Artificial Intelligence: ECAI*, 1998, S. 489–493 38

[Kopecký u. a. 2006] Kopecký, Jacek ; Roman, Dumitru ; Moran, Matthew ; Fensel, Dieter: Semantic Web Services Grounding. In: *AICT-ICIW '06: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*. Washington, DC, USA : IEEE Computer Society, 2006, S. 127 77

[Kopecký u. a. 2007] Kopecký, Jacek ; Vitvar, Tomas ; Bournez, Carine ; Farrell, Joel: SAWSDL: Semantic Annotations for WSDL and XML Schema. In: *IEEE Internet Computing* 11 (2007), Nr. 6, S. 60–67. – ISSN 1089-7801 21, 60, 65

[Küster u. a. 2005] Küster, Ulrich ; Stern, Mirco ; König-Ries, Birgitta: A Classifcation of Issues and Approaches in Automatic Service Composition. In: Zirpins, C. (Hrsg.) ; Ortiz, G. (Hrsg.) ; Lamerdorf, W. (Hrsg.) ; Emmerich, W. (Hrsg.): *Engineering Service Compositions: First International Workshop* Bd. WESC05, Yorktown Heights: IBM Research Devision, December 2005 14

[Kuter u. a. 2004] Kuter, Ugur ; Sirin, Evren ; Nau, Dana ; Parsia, Bijan ; Hendler, James: Information Gathering During Planning for Web Service Composition. In: *Proceedings of the Third International Semantic Web Conference*. Hiroshima, Japan, 2004 83, 144, 145

[Lago u. a. 2002] Lago, Ugo D. ; Pistore, Marco ; Traverso, Paolo: Planning with a language for extended goals. In: *Eighteenth national conference on Artificial intelligence*. Menlo Park, CA, USA : American Association for Artificial Intelligence, 2002, S. 447–454 39

[Lara u. a. 2004] Lara, Ruben ; Roman, Dumitru ; Polleres, Axel ; Fensel, Dieter: A Conceptual Comparison of WSMO and OWL-S. In: *Proceedings of ECOWS 2004* Bd. 3250/2004. Erfurt : Springer-Verlag, September 2004, S. 254–269 62

[Lausen u. a. 2005] Lausen, H. ; Polleres, A. ; Roman, D.: Web Service Modeling Ontology (WSMO). In: *W3C Member Submission* (2005). – URL `http://www.w3.org/Submission/WSMO/` 21, 52, 61

[Lee u. a. 2007] Lee, Jin W. ; Kim, Su M. ; Lim, Hun ; Schuster, Mario ; Domene, Alexander: A Software Architecture for Virtual Device Composition and Its Applications. In: *Ubiquitous Computing Systems*. Berlin Heidelberg : Springer-Verlag, 2007 (Lecture Notes in Computer Science), S. 150–157 98

[Levesque u. a. 1997] Levesque, Hector J. ; Reiter, Raymond ; Lespérance, Yves ; Lin, Fangzhen ; Scherl, Richard B.: GOLOG: A logic programming language for dynamic domains. In: *The Journal of Logic Programming* 31 (1997), Nr. 1-3, S. 59 – 83. – Reasoning about Action and Change 27

[Liebowitz und Margolis 1994] Liebowitz, S. J. ; Margolis, Stephen E.: Network Externality: An Uncommon Tragedy. In: *The Journal of Economic Perspectives* 8 (1994), Nr. 2, S. 133–150. – ISSN 08953309 12

[Long und Fox 2003] Long, D. ; Fox, M.: The 3rd International Planning Competition: Results and Analysis. In: *Journal of Artificial Intelligence Research* 20 (2003), S. 1–59 42, 46, 84

[Manna und Waldinger 1992] Manna, Z. ; Waldinger, R.: Fundamentals of Deductive Program Synthesis. In: *IEEE Transactions on Software Engineering* 18 (1992), Nr. 8, S. 674–704 28

[Marquardt u. a. 2008] Marquardt, Florian ; Reisse, Christiane ; Uhrmacher, Adelinde M. ; Kirste, Thomas: A Two-way Approach to Service Composition in Smart Device Ensembles. In: *Proceedings of Second Baltic Conference on Advanced Topics in Telecommunication*. Tartu, Estland, August 2008, S. 49–60 xi, 15, 97, 98

[Marquardt und Uhrmacher 2008] Marquardt, Florian ; Uhrmacher, Adelinde: Evaluating AI Planning for Service Composition in Smart Environments. In: *Proceedings of the 7th International ACM Conference on Mobile and Ubiquitous Multimedia*. Umea, Schweden : ACM, Dezember 2008, S. 48–55 97, 101

[Marquardt und Uhrmacher 2009a] Marquardt, Florian ; Uhrmacher, Adelinde M.: An AI-Planning based Service Composition Architecture for Ambient Intelligence. In: *Proceedings of the 4th Workshop on Artificial Intelligence Techniques for Ambient Intelligence*, July 2009 147

[Marquardt und Uhrmacher 2009b] Marquardt, Florian ; Uhrmacher, Adelinde M.: Creating AI Planning Domains for Smart Environments using

PDDL. In: *Proceedings of International Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing*, November 2009 25

[Marquardt und Uhrmacher 2009c] Marquardt, Florian ; Uhrmacher, Adelinde M.: Toward Unobtrusive Service Composition in Smart Environments based on AI Planning. In: *Proceedings of the 23. Workshop on Planen, Scheduling und Konfigurieren, Entwerfen: PuK*, September 2009 101

[Martin u. a. 2004] Martin, David ; Burstein, Mark ; Hobbs, Jerry ; Lassila, Ora ; McDermott, Drew ; McIlraith, Sheila ; Narayanan, Srini ; Paolucci, Massimo ; Parsia, Bijan ; Payne, Terry ; Sirin, Evren ; Srinivasan, Naveen ; Sycara, Katia: *OWL-S: Semantic Markup for Web Services*. W3C Member Submission. November 2004 21, 58, 60, 65

[Martínez und Lespérance 2004] Martínez, Erick ; Lespérance, Yves: Web Service Composition as a Planning Task: Experiments using Knowledge-Based Planning. In: *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004 24

[McDermott 2000] McDermott, Drew: The 1998 AI Planning Systems Competition. In: *AI Magazine* 21 (2000), S. 35–55 46

[McDermott 2002] McDermott, Drew: Estimated-Regression Planning for Interactions with Web Services. In: *Proceedings of the AI Planning Systems Conference*, AAAI Press, 2002, S. 204–211 24, 34, 85

[McIlraith 2002] McIlraith, Sheila: Adapting Golog for composition of semantic web Services. In: *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning*, 2002, S. 482–493 23, 27, 32, 33, 34, 52

[Mecella u. a. 2002] Mecella, Massimo ; Presicce, Francesco ; Pernici, Barbara: Modeling E -service Orchestration through Petri Nets. In: *Technologies for E-Services*. Berlin Heidelberg : Springer-Verlag, 2002, S. 109–134 23

[Milanovic und Malek 2004] Milanovic, Nikola ; Malek, Miroslaw: Current Solutions for Web Service Composition. In: *IEEE* November (2004), S. 51–59 14, 20

[Nau u. a. 2003] Nau, Dana ; Au, Tsz-Chiu ; Ilghami, Okhtay ; Kuter, Ugur ; Murdock, J. W. ; Wu, Dan ; Yaman, Fusun: SHOP2: An HTN Planning System. In: *Journal of Artificial Intelligence Research* 20 (2003), Dezember, S. 379–404 42

[Nau 2007] Nau, Dana S.: Current Trends in Automated Planning. In: *AI Magazine* 28 (2007), Nr. 4, S. 43–58 viii, 36, 37, 136, 140, 141

[Nebel und Koehler 1995] Nebel, Bernhard ; Koehler, Jana: Plan reuse versus plan generation: a theoretical and empirical analysis. In: *Artificial Intelligence* 76 (1995), Nr. 1-2, S. 427 – 454. – Planning and Scheduling 145

[Nielsen 1994] Nielsen, Jakob: *Usability Engineering*. Morgan Kaufmann, San Francisco, 1994 94

[Nishikawa u. a. 2006] Nishikawa, Hiroshi ; Yamamoto, Shinya ; Tamai, Morihiko ; Nishigaki, Kouji ; Kitani, Tomoya ; Shibata, Naoki ; Yasumoto, Keiichi ; Ito, Minoru: UbiREAL: Realistic Smartspace Simulator for Systematic Testing. In: *UbiComp 2006: Ubiquitous Computing* (2006), S. 459–476 68, 75

[Ojala 2007] Ojala, Timo (Hrsg.): *Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia, MUM 2007, Oulu, Finland, December 12-14, 2007*. Bd. *284*. ACM, 2007. (ACM International Conference Proceeding Series) 222

[O'Keefe 2006] O'Keefe, Greg: Improving the Definition of UML. In: *Model Driven Engineering Languages and Systems* (2006), S. 42–56 24

[Panning u. a. 2008] Panning, A. ; Al-Hamadi, A. ; Niese, R. ; Michaelis, B.: Facial expression recognition based on Haar-like feature detection. In: *Pattern Recognition and Image Analysis* 18 (2008), Nr. 3, S. 447–452 49

[Papazoglou u. a. 2006] Papazoglou, Michael P. ; Traverso, Paolo ; Dustdar, Schahram ; Leymann, Frank ; Krämer, Bernd J.: Service-Oriented Computing: A Research Roadmap. In: Cubera, Francisco (Hrsg.) ; Krämer, Bernd J. (Hrsg.) ; Papazoglou, Michael P. (Hrsg.): *Service Oriented Computing (SOC)*, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, 2006 (Dagstuhl Seminar Proceedings 05462) 15, 98

[Patil u. a. 2004] Patil, Abhijit A. ; Oundhakar, Swapna A. ; Sheth, Amit P. ; Verma, Kunal: Meteor-s web service annotation framework. In: *WWW '04: Proceedings of the 13th international conference on World Wide Web*. New York, NY, USA : ACM, 2004, S. 553–562 27

[Peer 2005] Peer, Joachim: Web Service Composition as AI Planning - a Survey / University of St. Gallen, Schweiz. 2005. – Forschungsbericht 22, 23

[Penberthy und Weld 1992] Penberthy, Scott J. ; Weld, Daniel S.: UCPOP: A Sound, Complete, Partial Order Planner for ADL. In: Nebel, Bernhard (Hrsg.) ; Rich, Charles (Hrsg.) ; Swartout, William (Hrsg.): *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. San Mateo, USA : Morgan Kaufmann, 1992, S. 103–114 47, 102

[Peot und Smith 1992] Peot, Mark A. ; Smith, David E.: Conditional nonlinear planning. In: *Proceedings of the first international conference on Artificial intelligence planning systems*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1992, S. 189–197 83

[Plociennik 2010] Plociennik, Christiane: *Device Cooperation in Ad-hoc Multimedia Ensembles*, Fakultät für Informatik und Elektrotechnik, Universität Rostock, Dissertation, 2010. – unpublished 144

[Plociennik u. a. 2009] Plociennik, Christiane ; Burghardt, Christoph ; Marquardt, Florian ; Kirste, Thomas ; Uhrmacher, Adelinde: Modelling Device Actions in Smart Environments. In: *Proceedings of International Conference on Intelligent Interactive Assistance and Mobile Multimedia Computing*. Rostock, November 9-11 2009 86, 87

[Ponnekanti u. a. 2001] Ponnekanti, Shankar ; Lee, Brian ; Fox, Armando ; Hanrahan, Pat ; Winograd, Terry: ICrafter: A Service Framework for Ubiquitous Computing Environments. In: *Ubicomp 2001: Ubiquitous Computing*. Berlin Heidelberg : Springer-Verlag, 2001 (LNCS), S. 56–75 29

[Ponnekanti und Fox 2002] Ponnekanti, Shankar R. ; Fox, Armando: SWORD: A developer toolkit for web service composition. In: *Proceedings of the 11th International WWW Conference (WWW2002)*. Honolulu, HI, USA, 2002 14, 22, 27, 32, 33, 34

[Rao und Su 2004] Rao, Jinghai ; Su, Xiaomeng: A Survey of Automated Web Service Composition Methods. In: *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition, SWSWPC2004, LNCS*. San Diego, USA, 2004 14, 16, 28, 32, 33, 34, 52, 146

[Reisig und Rozenberg 1998] Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*. Bd. *1491*. Springer-Verlag, 1998. (Lecture Notes in Computer Science) 23

[Reisse u. a. 2008a] Reisse, Christiane ; Burghardt, Christoph ; Marquardt, Florian ; Kirste, Thomas: Intelligente Umgebungen und das Semantic Web. In: *Information Managment & Consulting* 23. Jahrgang, Heft 2, Mai 2008 (2008), Mai, S. 28–33 58

[Reisse u. a. 2008b] Reisse, Christiane ; Burghardt, Christoph ; Marquardt, Florian ; Kirste, Thomas ; Uhrmacher, Adelinde M.: Smart Environments Meet the Semantic Web. In: Zaslavsky, Arkady (Hrsg.) ; Wiberg, Mikael (Hrsg.): *Proceedings of the 7th International ACM Conference on Mobile and Ubiquitous Multimedia*. Umea, Schweden : ACM, Dezember 2008, S. 88–91 58, 98

[Reisse und Kirste 2008] Reisse, Christiane ; Kirste, Thomas: A distributed mechanism for device cooperation in Smart Environments. In: *Advances in Pervasive Computing. Adjunct proceedings of the 6th International Conference on Pervasive Computing*. Sydney, Australia, 2008, S. 53–56 144

[Richter und Westphal 2008] Richter, Silvia ; Westphal, Matthias: The LAMA Planner - Using Landmark Counting in Heuristic Search. In: *Short paper for the International Planning Competition 2008*, 2008 124, 130

[Rintanen 2004] Rintanen, Jussi: Phase transitions in classical planning: an experimental study. In: *In Principles of Knowledge Representation and Reasoning*, AAAI Press, 2004, S. 710–719 96, 101, 102

[Rintanen und Hoffmann 2001] Rintanen, Jussi ; Hoffmann, Jörg: An overview of recent algorithms for AI planning. In: *Künstliche Intelligenz* 2 (2001), May, S. 5–11 41

[Roberts und Howe 2009] Roberts, Mark ; Howe, Adele E.: Learning from planner performance. In: *Artificial Intelligence* 173 (2009), Nr. 5-6, S. 536–561 41

[Roman u. a. 2005] Roman, Dumitru ; Lausen, Holger ; Keller, Uwe: *WSMO Final Draft*. April 2005 61, 65

[Roman u. a. 2003] Roman, M. ; Ziebart, B. ; Campbell, R.H.: Dynamic application composition: customizing the behavior of an active space. In: *1st IEEE Conference on Pervasive Computing and Communications*. Dallas : IEEE CS Press, March 2003, S. 169–176 75

[Rouvoy u. a. 2008] Rouvoy, Romain ; Eliassen, Frank ; Floch, Jacqueline ; Hallsteinsen, Svein O. ; Stav, Erlend: Composing Components and Services Using a Planning-Based Adaptation Middleware. In: Pautasso,

Cesare (Hrsg.) ; Tanter, Éric (Hrsg.): *Software Composition, 7th International Symposium, SC 2008* Bd. 4954. Budapest, Ungarn : Springer-Verlag, März 2008, S. 52–67 30

[Russell und Norvig 2003] Russell, Stuart ; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. 2nd edition. Prentice-Hall, Englewood Cliffs, NJ, 2003 37, 42

[Satyanarayanan 2001] Satyanarayanan, M.: Pervasive Computing: Vision and Challenges. In: *IEEE Personal Communications* 8 (2001), S. 10–17 4

[Sheshagiri u. a. 2003] Sheshagiri, Mithun ; desJardins, Marie ; Finin, Timothy: A Planner for Composing Services Described in DAML-S. In: *Proceedings of The Second International Joint Conference on Autonomous Agents Multiagent Systems, AAMAS 2003*. Melbourne, Victoria, Australien, Juli 2003 23, 27, 32, 33, 34

[Simpson u. a. 2006] Simpson, Richard ; Schreckenghost, Debra ; LoPresti, Edmund ; Kirsch, Ned: Plans and Planning in Smart Homes. In: *Designing Smart Homes* (2006), S. 71–84 72

[Sirin u. a. 2004] Sirin, E. ; Parsia, B. ; Wu, D. ; Hendler, J. ; Nau, D.: HTN planning for web service composition using SHOP. In: *Journal of Web Semantics* 1 (4) (2004), S. 377–396 23, 42

[Sirin u. a. 2003] Sirin, Evren ; Hendler, James ; Parsia, Bijan: Semi-automatic Composition of Web Services using Semantic Descriptions. In: *Web Services: Modeling, Architecture and Infrastructure Workshop in Conjunction with ICEIS*, 2003 21

[Srivastava 2003] Srivastava, B.: A Limited Extension of PDDL for Planning with Non-primitive actions. In: *ICAPS 2003 Workshop on Planning Competition*, 2003 42

[Srivastava und Koehler 2003] Srivastava, Biplav ; Koehler, Jana: Web Service Composition - Current Solutions and Open Problems. In: *Proceedings of ICAPS 2003* (2003) 21, 29, 40, 56

[Srivastava und Koehler 2004] Srivastava, Biplav ; Koehler, Jana: Planning with Workflows - An Emerging Paradigm for Web Service Composition. In: *Proceedings of Workshop on Planning and Scheduling for Web and Grid Services at ICAPS 2004*, June 2004 16, 24, 40, 84

[Stelios u. a. 2008] Stelios, Mitilineos A. ; Nick, Argyreas D. ; Effie, Makri T. ; Dimitris, Kyriazanos M. ; Thomopoulos, Stelios C. A.: An indoor localization platform for ambient assisted living using UWB. In: *MoMM '08: Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*. New York, NY, USA : ACM, 2008, S. 178–182 49

[Svensson u. a. 2007] Svensson, D. ; Hedin, G. ; Magnusson, B.: Pervasive applications through scripted assemblies of services, July 2007, S. 301–307 30, 32, 33, 75

[Tabatabaei u. a. 2008] Tabatabaei, Sayed Gholam H. ; Kadir, Wan Mohd Nasir W. ; Ibrahim, Suhaimi: Semantic Web Service Discovery and Composition Based on AI Planning and Web Service Modeling Ontology. In: *Asia-Pacific Conference on Services Computing. 2006 IEEE* 0 (2008), S. 397–403 62

[Tanenbaum 2003] Tanenbaum, Andrew S.: *Computer Networks*. 4. Prentice Hall, 2003 12

[Thiébaux u. a. 2005] Thiébaux, Sylvie ; Hoffmann, Jörg ; Nebel, Bernhard: In defense of PDDL axioms. In: *Artificial Intelligence* 168 (2005), Nr. 1-2, S. 38 – 69 47, 88

[Thoene u. a. 2003] Thoene, Sebastian ; Depke, Ralph ; Engels, Gregor: Process-Oriented, Flexible Composition of Web Services with UML. In: *Advanced Conceptual Modeling Techniques* (2003), S. 390–401 23

[Traverso und Pistore 2004] Traverso, Paolo ; Pistore, Marco: Automated Composition of Semantic Web Services into Executable Processes. In: *The Semantic Web ISWC 2004* (2004), S. 380–394 28, 32, 33, 34

[United Nations Conference on Trade and Development (UNCTAD) 2007] United Nations Conference on Trade and Development (UNCTAD): *Information Economy Report 2007-2008 Science and technology for development: the new paradigm of ICT*. 2007 4

[Vallée u. a. 2005] Vallée, M. ; Ramparany, F. ; Vercouter, L.: Flexible composition of smart device services. In: *The 2005 International Conference on Pervasive Systems and Computing (PSC-05)*. Las Vegas, USA, Juni 2005 29, 32, 33

[W3C 2001] W3C: *Team Comment on the "Web Services Description Language (WSDL) 1.1SSubmission*. W3C Staff Comment. 2001. – URL `http://www.w3.org/Submission/2001/07/Comment` 59

[W3C 2007a] W3C ; Mitra, Nilo (Hrsg.) ; Lafon, Yves (Hrsg.): *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation. April 2007. – URL `http://www.w3.org/TR/soap12-part0/`. – W3C Recommendation 13

[W3C 2007b] W3C: *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation. Juni 2007. – URL `http://www.w3.org/TR/wsdl20/` 21, 58, 65

[Weihe 2001] Weihe, Karsten: On the Differences between "Practical" and "Applied". In: *Proceedings of 4th International Workshop on Algorithm Engineering, WAE 2000*. Lecture Notes in Computer Science. Saarbrücken : Springer-Verlag, 2001, S. 1–10 100

[Weiser 1991] Weiser, Mark: The computer for the 21st century. In: *Scientific American* 265 (1991), September, Nr. 3, S. 66–75. ISBN 1558602461 3

[Weld 1994] Weld, Daniel S.: An Introduction to Least Commitment Planning. In: *AI Magazine* 15 (1994), Nr. 4, S. 27–61 72, 85

[Weld u. a. 1998] Weld, Daniel S. ; Anderson, Corrin R. ; Smith, David E.: Extending Graphplan to Handle Uncertainty & Sensing Actions. In: *Proceedings of AAAI '98*, The AAAI Press, 1998 (15), S. 897 – 904 102

[Wodtke und Weikum 1997] Wodtke, Dirk ; Weikum, Gerhard: A Formal Foundation for Distributed Workflow Execution Based on State Charts. In: *ICDT '97: Proceedings of the 6th International Conference on Database Theory*. London, UK : Springer-Verlag, 1997, S. 230–246. – ISBN 3-540-62222-5 23

[Wolf 2009] Wolf, Karsten: A Theory of Service Behavior. In: Kopp, Oliver (Hrsg.) ; Lohmann, Niels (Hrsg.): *Proceedings of the 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2-3, 2009* Bd. 438, CEUR-WS.org, März 2009, S. 1–7 143

[Wu u. a. 2003] Wu, Dan ; Parsia, Bijan ; Sirin, Evren ; Hendler, James ; Nau, Dana: Automating DAML-S Web Services Composition Using SHOP2. In: *The SemanticWeb - ISWC*, 2003 23, 42

[Yang u. a. 2006] Yang, Yuping ; Mahon, Fiona ; Williams, M. ; Pfeifer, Tom: Context-Aware Dynamic Personalised Service Re-composition in a Pervasive Service Environment. In: *Ubiquitous Intelligence and Computing* (2006), S. 724–735 72

[Yu und Reiff-Marganiec 2006] Yu, Hong Q. ; Reiff-Marganiec, Stephan: Semantic Web Services Composition via Planning as Model Checking / University of Leicester. 2006. – Forschungsbericht 28

[Zambonelli und Parunak 2002] Zambonelli, Franco ; Parunak, H. Van D.: Signs of a Revolution in Computer Science and Software Engineering. In: Petta, Paolo (Hrsg.) ; Tolksdorf, Robert (Hrsg.) ; Zambonelli, Franco (Hrsg.) ; Ossowski, Sascha (Hrsg.): *Workshop Notes of the Third International Workshop Engineering Societies in the Agents World (ESAW)*, 2002 54

[Zhang u. a. 2004] Zhang, Jia ; Chung, Jen-Yao ; Chang, Carl K. ; Kim, Seong W.: WS-Net: A Petri-net Based Specification Model for Web Services. In: *Web Services, IEEE International Conference on* 0 (2004), S. 420. ISBN 0-7695-2167-3 23

[Zheng und Yan 2008] Zheng, Xianrong ; Yan, Yuhong: An Efficient Syntactic Web Service Composition Algorithm Based on the Planning Graph Model. In: *ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2008, S. 691–699 28

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Dissertation selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Barleben, den 16. Oktober 2010

Florian Marquardt

Lebenslauf

Persönliche Daten

Dipl.-Ing. Florian Marquardt

geboren am 01.03.1980 in Magdeburg

ledig

Promotion

12/2009 - 02/2010 Promotionsstipendium der Universität Rostock

12/2006 - 11/2009 Promotionsstipendium der DFG im Graduiertenkolleg MuSAMA

Studium

10/1999 - 03/2006 Universitätsstudium der Computervisualistik an der Otto-von-Guericke-Universität Magdeburg

Abschluss Diplom-Ingenieur für Computervisualistik (1,2)

Schulausbildung

09/1991 - 06/1998 Humboldt-Gymnasium Magdeburg

Abschluss Abitur (2,2)

Barleben, 16. Oktober 2010

Dienstekomposition in intelligenten Umgebungen basierend auf KI-Planung

Zusammenfassung:

Das Ziel der Forschung auf dem Gebiet der intelligenten Umgebungen ist es, Techniken zu entwickeln, die es erlauben, dynamische Umgebungen zu kreieren, mit denen Nutzer in natürlicher und unaufdringlicher Art interagieren können. Diese Umgebungen setzten sich aus verschiedensten heterogenen Geräten und deren Diensten zusammen. Die Herausforderung besteht darin, basierend auf den Intentionen des Nutzers und den Fähigkeiten der Geräte, höherwertige Dienste anzubieten. Dazu wird oft das Zusammenspiel mehrerer Dienste benötigt, welches durch eine Dienstekomposition erzielt werden kann. KI-Planung ist eine Methode, Dienstekomposition umzusetzen. Um eine unaufdringliche Dienstekomposition in intelligenten Umgebungen gewährleisten zu können, sind schnelle und verlässliche Implementierungen von Planungsalgorithmen erforderlich.

Im Rahmen der vorliegenden Arbeit wurde experimentell das Laufzeitverhalten von verschiedenen Planern untersucht. Dabei stellte sich heraus, dass kein überlegener Planer existiert. Verschiedene Planer haben verschiedene Stärken, sowohl aus effizienzbasierter als auch aus funktioneller Sicht. Die Auswertung der Experimente führte zu verschiedenen Strategien. Von diesen war die Austauschstrategie am besten geeignet, die Laufzeit von Planern zu verkürzen und dabei die Anzahl der gefundenen Lösungen zu erhöhen.

Auch die Möglichkeiten der Modellierung von Problemen der Dienstekomposition wurden anhand von mehreren Beispielszenarien aus der Literatur evaluiert, was zu einer Richtline für die verteilte Modellierung von Dienstbeschreibungen führte.

Basierend auf den Erfahrungen aus den Laufzeitexperimenten und der Modellierung wurde ein Composer entworfen und umgesetzt, der verschiedene Planer für die Dienstekomposition in intelligenten Umgebungen nutzen kann. Der Composer stellt einen erweiterten Metaplaner dar, der die Möglichkeiten serviceorientierter Architekturen nutzt.

Schlagwörter: Dienstekomposition, Intelligente Umgebungen, KI-Planung

Service Composition in intelligent environments based on AI planning

Abstract:

The aim of research in the field of smart environments is to explore techniques which allow to build dynamic ensembles of devices in a fashion that users can interact with them in an unobtrusive and natural way. These devices are likely to be heterogeneous and offer different services. Based on user intentions and the capabilities of the devices, the challenge is to provide more advanced services. Therefore often cooperation is required which is based on the composition of services. AI planning is one possibility to realize service composition. For unobtrusive service composition in smart environments fast and reliable implementations of planning algorithms are required.

Experimentally the behavior of several planners were investigated. It concludes that no silver bullet exists. Different planners show different strengths in terms of efficiency as well as functionality. The analysis of the experiments led to different strategies. Among them the exchange strategy was best suited to shorten runtime of the planners and to increase the number of found solution as well.

In addition the possibilities of modelling service composition problems was evaluated based on several szenarios from literature, what led to a guideline for distributed modelling of service descriptions.

Based on the experiences from the runtime experiments and the modeling a composer is introduced which uses different AI planners to compose services in smart environments. The composer can be seen as an enhanced meta planner that leverages SOA technologies.

Keywords: Service composition, Smart Environments, AI planning