Universität Ulm
Fakultät für Informatik
Institut für Künstliche Intelligenz

ulm university universität
uulm

Ein Framework zur nutzerorientierten Erklärung hybrider Pläne

Diplomarbeit von:
Christian Bauer

Eingereicht am:
30.06.2011

Gutachter:
Prof. Dr. S. Biundo-Stephan
Dr. B. Schattenberg

Betreuer:
Dr. B. Schattenberg
F. Müller

Das Planungssystem PANDA entwickelt unter Kombination von Ansätzen des HTN- und des POCL-Planens Handlungspläne für verschiedene Domänen. Eine Möglichkeit, solche Pläne einzusetzen, liegt in der Entwicklung von planbasierten Assistenzsystemen, die einen Nutzer in der Umsetzung diverser Aufgaben unterstützen. Um das Vertrauen des Nutzers in solche Assistenzsysteme zu stärken und sein Verständnis der ihm gegebenen Handlungsanweisungen zu verbessern kann ein System eingesetzt werden, welches diese Pläne *erklärt*.

Wir stellen in dieser Arbeit ein generisches Framework vor, welches zur Umsetzung von Systemen zur *nutzerorientierten Planerklärung* verwendet werden kann. Den grundlegenden Ansatz zur Erzeugung von Planerklärungen bildet darin ein dreischrittiger Prozess. Dabei werden zunächst alle aus dem Plan ableitbaren Erklärungen generiert. In einem zweiten Schritt werden diese Erklärungen auf *Defizite* untersucht, das heißt auf Eigenschaften, die sie für die konkrete Anwendungssituation unbrauchbar machen. Wir entfernen diese Defizite mit Hilfe von zu diesem Zweck entwickelten Erklärungsmodifikationen und erhalten damit Erklärungen, die zur Präsentation geeignet sind. Zum Abschluss des Erklärungsprozesses wird die beste dieser Erklärungen ausgewählt, indem eine Bewertung der Erklärungen anhand ihrer positiven Eigenschaften, ihrer *Werte*, durchgeführt wird.

Die Anpassung dieses Ablaufs an den Nutzer wird dadurch realisiert, dass Anforderungen an eine auszugebende Erklärung, also die zu vermeidenden Defizite sowie die erwünschten Werte, nicht allgemein definiert, sondern erst von der konkreten Anwendung und ihrem Nutzer festgelegt werden. Wir identifizieren in dieser Arbeit mit den Eigenschaften des Nutzers, der Domäne des zu erklärenden Plans sowie der eingesetzten Benutzerschnittstelle drei Quellen von Informationen, die bei der Definition dieser Werte und Defizite eine Rolle spielen können, und bestimmen welchen Einfluss sie jeweils ausüben. Die Repräsentation der Anforderungen und ihre Auswertung durch das Framework erfolgt durch eine beschreibungslogische Wissensbasis. Zum Abschluss der Arbeit stellen wir eine prototypische Implementierung des Frameworks sowie einer Beispielanwendung auf Basis der Smartphone-Domäne vor.

Inhaltsverzeichnis

1 Einleitung **9**
1.1 Maschinell erstellte Pläne und ihre Erklärung 9
1.2 Aufgabenstellung und Aufbau der Arbeit 10

2 Grundlagen **12**
2.1 Planen 12
2.1.1 Partial Order Causal Link Planning 13
2.1.2 Hierarchical Task Network Planning 16
2.1.3 PANDA 18
2.2 Wissensrepräsentation 22
2.2.1 Beschreibungslogik 23
2.2.2 Web Ontology Language 26

3 Die Smartphone-Domäne **28**

4 Allgemeine Aspekte der Planerklärung **32**
4.1 Mögliche Fragestellungen 32
4.2 Aufbau einer Planerklärung 36
4.3 Die Suche nach einer guten Erklärung 37
4.3.1 Bewertung von Erklärungen 38
4.3.2 Maßstäbe für die Bewertung 39
4.3.3 Informierte Planerklärung 41

5 Ein Ansatz zur nutzerorientierten Planerklärung **46**
5.1 Erzeugung von Roherklärungen 47
5.1.1 Axiome zum Beweis von Beziehungen im Plan 48
5.1.2 Axiome zum Beweis von erfragten Eigenschaften 49
5.2 Behandlung von Defiziten 50
5.2.1 Behandlung von Planschritt-Defiziten 52
5.2.2 Behandlung von Argument-Defiziten 54
5.2.3 Behandlung von Verbindungs-Defiziten 55
5.2.4 Behandlung von Überlange-Erklärung-Defiziten 56
5.2.5 Anmerkungen zum Ansatz 56

6 Wissensbasierte Detektion von Defiziten und Bewertung von Erklärungen **62**
6.1 Repräsentationsform 63

6.2 Grundlegende Bestandteile der Wissensbasis . . . 64
6.2.1 Domäne . . . 64
6.2.2 Erklärung . . . 65
6.3 Defizite und Werte – Schnittstelle zwischen Framework und Anwendung 66
6.3.1 Definition und Detektion von Defiziten . . . 66
6.3.2 Bewertung von Erklärungen . . . 68
6.4 Wissen über den Nutzer . . . 69
6.4.1 Verwertbarkeit von Information . . . 70
6.4.2 Verfügbarkeit von Information . . . 71
6.4.3 Repräsentation des Nutzermodells . . . 73
6.5 Wissen über die Schnittstelle . . . 74
6.5.1 Repräsentation der Schnittstelle . . . 75
6.6 Wissen über die Domäne . . . 75
6.6.1 Technisch bedingte Domänenelemente . . . 76
6.6.2 Selbsterklärende Domänenelemente . . . 76
6.6.3 Repräsentation von Wissen über die Domäne . . . 77

7 Prototypische Implementierung des Frameworks 82
7.1 Überblick über die Implementierung des Erklärungsprozesses . . . 83
7.2 Verwendete Bibliotheken . . . 83
7.3 Erzeugung von OWL-Repräsentationen . . . 85
7.3.1 Repräsentation von Domänen . . . 85
7.3.2 Repräsentation von Erklärungen . . . 85
7.4 Definition von Werten und Defiziten in der Implementierung . . . 86
7.5 Die Umsetzung des Erklärungsprozesses im Detail . . . 86
7.5.1 Die Explainer-Klasse . . . 86
7.5.2 Annahme einer Anfrage . . . 87
7.5.3 Defizitbehandlung . . . 88
7.5.4 Bewertung von Erklärungen . . . 90
7.6 Anwendungsbeispiel ShellClient: Textbasierte Planerklärung in der Smartphone-Domäne . . . 90
7.6.1 Der ShellClient . . . 91
7.6.2 Anpassung der Kommunikationsstruktur . . . 92
7.6.3 Wissensmodelle . . . 93

8 Verwandte Arbeiten 96
8.1 Mixed Initiative Planning . . . 96
8.2 Erklärung . . . 97

9 Zusammenfassung und Ausblick 99
9.1 Zusammenfassung . . . 99
9.2 Ausblick . . . 99

Literaturverzeichnis **102**

Abbildungsverzeichnis **106**

1 Einleitung

1.1 Maschinell erstellte Pläne und ihre Erklärung

Neben vielen anderen Kerngebieten behandelt die Forschung im Bereich der künstlichen Intelligenz auch die maschinelle Erzeugung von Handlungsplänen. Das dabei angestrebte Ziel, Handlungspläne automatisiert durch Computersysteme entwickeln zu lassen, ist von zahlreichen möglichen Einsatzbereichen geprägt. Eine Motivation besteht in der fortschreitenden Entwicklung autonomer Systeme – man denke beispielsweise an die Marsmissionen mit unbemannten Fahrzeugen –, die Aufgaben von zunehmend höherer Schwierigkeit meistern sollen, und zu diesem Zweck nicht mehr auf einfache reaktive Systeme zurückgreifen können, sondern ihre Vorgehensweise zum Erreichen von Missionszielen im Voraus planen müssen. Ein weiterer Anreiz ist es, menschliche Nutzer in der Lösung von Problemen zu unterstützen, die zu umfangreich sind, um von ihnen allein in einer akzeptablen Zeit gelöst zu werden. Diese Art von Unterstützung ist auch deshalb interessant, weil ein automatisch generierter Plan für ein solches komplexes Problem im Nachhinein schwer zu validieren ist, und von einem Nutzer besser nachvollzogen werden kann, wenn dieser am Planungsvorgang selbst beteiligt war.

In eine ähnliche Richtung geht die Erforschung von planbasierten Assistenzsystemen (siehe z.B. [BBG+11]), die einen Nutzer bei der Ausübung bestimmter Tätigkeiten unterstützen sollen. Eine Anwendung solcher Software-Assistenten kann es beispielsweise sein, ihrem Nutzer Handlungsanweisungen zur Lösung verschiedener Aufgaben bereitzustellen. Ein entscheidender Aspekt ist hier, dass eine Maschine die erhaltenen Handlungsanweisungen nicht in ihrer Bedeutung verstehen, sondern lediglich zu ihrer operationellen Umsetzung in der Lage sein muss. Ein Mensch wird hingegen Anweisungen in Frage stellen, deren Sinn sich ihm nicht erschließt, und die er daher nicht ausführen kann oder will. Dies ist ein intrinsisches Problem von Assistenzsystemen, die ja gerade bei unverständlichen Aufgabenstellungen helfend eingreifen sollen. Würde man dem Nutzer einen menschlichen, als Experten „ausgewiesenen“ Berater zur Seite stellen, würde er dessen Kompetenz in der Regel als gegeben annehmen und seinen Anweisungen Folge leisten. Bei einem Software-Assistenzsystem ist eher das Gegenteil der Fall, nämlich dass der Nutzer ihm, und damit auch von ihm stammenden Handlungsplänen, prinzipiell mit Misstrauen gegenübersteht – dem Softwaresystem wird nicht das selbe *Vertrauen* entgegengebracht wie einem menschlichen Experten.

An dieser Stelle kann der Assistent durch ein weiteres System unterstützt werden, welches dem Nutzer ermöglicht, sich einen Plan und das Systemverhalten als ganzes *erklären* zu lassen. Ist es ihm möglich, Fragen zu Eigenschaften des Plans zu stellen und erhält er auf diese verständliche Antworten, kann einerseits das Vertrauen des Nutzers in das System gestärkt werden, indem begründet wird, warum der Plan „so aussieht wie er aussieht“ – oder etwas formaler gesagt, warum ein Aspekt der Problemstellung auf diese Weise gelöst, warum dafür eine bestimmte Ressource verwendet wurde und so weiter. Andererseits kann ein solches System auch zum Erreichen des eigentlichen Ziels des Assistenten beitragen, nämlich den Nutzer in der Ausführung seiner Aufgaben zu unterstützen, da mit jeder Klärung einer seiner Fragen die Kompetenz des Nutzers bezüglich der Domäne erweitert wird.

Entscheidend für den Einfluss, den ein solches Erklärungssystem ausüben kann, ist die Qualität der gelieferten Erklärungen von Plänen, das heißt wie verständlich diese die nachgefragten Phänomene erklären. So ist klar, dass ein Planungsexperte, der das Erklärungssystem dazu nutzen will, die Arbeitsweise eines Planungssystems zu überprüfen, einen ganz anderen Blickwinkel auf einen Plan hat als ein Endnutzer, der sich in die Funktionen eines neuen Handys einarbeitet und sich dabei von einem Assistenten unterstützen lässt. Entsprechend sollten diesen auch unterschiedliche Erklärungen geliefert werden, da der Endnutzer beispielsweise ein stark eingeschränktes Vokabular bezüglich der Bedienung seines Handys hat, und eine Erklärung von Details gar nicht verstehen könnte. Darüber hinaus könnte der Experte zur Validierung eher ein normales PC-System mit den üblichen Eingabe- und Ausgabegeräten nutzen und sich die Erklärung auf einem großflächigen Display anzeigen lassen, während die Funktionen eines Handys eher über eine Audioausgabe erklärt werden, um seine gleichzeitige Bedienung mit Hilfe des Touchscreens nicht zu beeinträchtigen. Insgesamt ergeben sich aus unterschiedlichen Gründen eine Anzahl von Anforderungen, die aufgegriffen werden können um eine nutzeradaptive Planerklärung zu realisieren, die auch in so unterschiedlichen Fällen wie den genannten gute Erklärungen liefert kann.

1.2 Aufgabenstellung und Aufbau der Arbeit

Ziel dieser Arbeit ist es, ein generisches Framework zu entwerfen, welches zur Umsetzung nutzerorientierter Erklärungssysteme instanziiert werden kann. Ein Nutzer soll Anfragen an das System zu von ihm betrachteten Handlungsplänen stellen und darauf Erklärungen erhalten können. Diese Erklärungen sollen sich, um eine größtmögliche Verständlichkeit und Akzeptanz zu erreichen, am Nutzer orientieren, das heißt an seinen persönlichen Präferenzen sowie seinem Bezug zur Domäne des zu erklärenden Plans, und der Schnittstelle der Interaktion zwischen Nutzer und Erklärungssystem. Daraus ergeben sich mehrere Teilziele für diese Arbeit.
Zunächst ist zu untersuchen, wie eine Erklärung generell aufgebaut sein kann, und

woran sich prinzipiell ihre Qualität für den Nutzer bemessen lässt. Weiterhin muss ein generischer Prozess gefunden werden, durch den Erklärungen für von Nutzern gestellte Fragen ermittelt werden können. Dabei soll bestimmt werden, wie dieser Prozess durch Informationen über den Nutzer verbessert werden kann. Dazu ist wiederum festzustellen, welches Wissen über den Nutzer konkret herangezogen werden kann, und wie dieses Wissens zu modellieren ist, um seine Verarbeitung durch das System zu ermöglichen. Schließlich ist eine Software-Schnittstelle zu entwickeln, an der konkret umzusetzende Erklärungssysteme ansetzen und in den vom Framework vorgegebenen generischen Erklärungsprozess steuernd eingreifen können, um eine Adaption an ihre Nutzer zu erreichen. Die Arbeit beschäftigt sich dagegen nicht mit Problemstellungen auf der Anwendungsseite, die zum Beispiel eine Verbalisierung von Erklärungen, insbesondere für Nutzer, die keine Planungsspezialisten sind, umsetzen muss. Ebenfalls wurden keine didaktischen Fragestellungen wie beispielsweise eine dialogbasierten Erklärung von Planphänomenen behandelt.

Wir bauen zur Umsetzung dieser Aufgaben diese Arbeit wie folgt auf. In Kapitel 2 beschreiben wir die grundlegenden Mechanismen, die zur Erzeugung der Pläne, die wir erklären wollen, verwendet werden, sowie Ansätze, mit denen Wissen maschinenlesbar aufbereitet werden kann. Wir stellen in Kapitel 3 eine Planungsdomäne vor, an der sich die Beispiele dieser Arbeit orientieren werden. In Kapitel 4 betrachten wir einige allgemeine Aspekte der Planerklärung, nämlich mögliche vom Nutzer gestellte Fragen, welche Form eine Antwort auf solche Fragen haben muss, und woran die Qualität dieser Antworten gemessen werden kann. Kapitel 5 beschreibt den Ansatz, den wir zur Umsetzung einer nutzerorientierten Planerklärung verfolgen. In Kapitel 6 legen wir dar, wie dieser Prozess durch Wissen über den Nutzer gesteuert, und wie dieses Wissen zu diesem Zweck zu modellieren ist. Schließlich zeigen wir in Kapitel 7 Aspekte einer prototypischen Implementierung unseres Frameworks und einer damit realisierten Anwendung für die Erklärung von Plänen aus der in Kapitel 3 vorgestellten Domäne. Im Anschluss daran werden wir in Kapitel 8 unseren Ansatz in den Kontext thematisch verwandter Arbeiten einordnen, und schließlich in Kapitel 9 die Ergebnisse unserer Arbeit noch einmal zusammenfassen und einige mögliche Erweiterungen präsentieren.

2 Grundlagen

In den folgenden Kapiteln sollen die in dieser Arbeit verwendeten theoretischen wie praktischen Grundlagen vermittelt werden. Dazu gehören einerseits eine Einführung in die Handlungsplanung anhand zweier grundlegender Herangehensweisen sowie der Vorstellung eines konkreten maschinellen Planungssystems, andererseits zwei Ansätze zur maschinenverarbeitbaren Modellierung von Wissen.

2.1 Planen

Menschen betreiben ständig Handlungsplanung, sind sich dessen aber nicht immer bewusst. Nur wenn es sich bei den durchzuführenden Handlungen um komplexe Prozesse und nicht etwa einfache, verinnerlichte Abläufe handelt, spricht man tatsächlich von *Planen*, und drückt damit explizit die Notwendigkeit bewusster Planungsaktivität aus.
Unabhängig davon wie kompliziert die Aufgabenstellung ist gibt es jedoch in jedem Fall eine Zielvorgabe, zu deren Erreichen von einer ebenfalls gegebenen Ausgangssituation aus der Planende eine geeignete Kombination möglicher Handlungen sucht. Die Ausführung jeder dieser Handlungen bewirkt eine Veränderung der Welt, wobei ihre Ausführung nicht zu jeder Zeit möglich ist, sondern jeweils an ihr eigene Anforderungen an die Situation geknüpft ist. Um diese Anforderungen zu erfüllen werden andere Aktionen herangezogen, die wiederum eigene Bedingungen mit sich bringen. Auf diese Weise entsteht eine Kette von Handlungen, die letztlich von der Ausgangssituation zum Ziel führt – ein Plan.

Maschinelles Planen stellt den Versuch dar, die bei dieser Suche nach einer Problemlösung beim Menschen ablaufenden Prozesse nachzubilden, und übernimmt dabei meist eine Betrachtungsweise der Welt als System von Zuständen und Zustandsübergängen. Ein Zustand definiert sich über die Eigenschaften der Welt, die in ihm gelten, während Aktionen diese Eigenschaften verändern und so zu einem neuen Zustand führen. Der Mensch ist in der Lage, die Bedingungen, an die diese Aktionen wie beschrieben geknüpft sind, in die Planung einzubeziehen, da sie je nach Art der Aktion intuitiv klar sind oder durch Lernen erkannt wurden. Ein Planungssystem hingegen hat grundsätzlich keine Kenntnis über die *Domäne*, in der ein Plan entwickelt werden soll, also die Umgebung mit den in ihr gültigen Regeln, verfügbaren Aktionen und beteiligten Entitäten. All dieses Wissen, welches der Mensch selbst

erworben hat, muss dem System stattdessen in einem Domänenmodell explizit zur Verfügung gestellt werden.

Die im Folgenden vorgestellten Ansätze maschinellen Planens sind sehr von der menschlichen Planungsweise inspiriert. Hier sind vor allem zwei Charakteristika ausschlaggebend, die die Basis für unterschiedlich arbeitende Planungssysteme bilden. Zum einen lässt sich ein Mensch bei der Handlungsplanung stark von kausalen Zusammenhängen leiten, die die Aktionen verbinden. Ausgehend vom zu erreichenden Ziel können beispielweise alle zur Verfügung stehenden Aktionen daraufhin untersucht werden, ob sie zur Herstellung der Eigenschaften des Zielzustands geeignet sind. Sind Kandidaten gefunden, die die gesuchten Veränderungen der Welt mit sich bringen, können wiederum deren Voraussetzungen betrachtet und Aktionen gesucht werden, die einen Zustand mit den gewünschten Eigenschaften versehen usw. Auf diese Weise hangelt sich ein Mensch vom Ziel aus rückwärts zum momentanen Zustand, indem er eine Kette von Handlungen findet, die den jeweils herrschenden Zustand in einen überführen, der der Ausführung der nächsten Aktion genügt.
Zum anderen ist es eine Eigenschaft menschlichen Planens, dass Handlungen auf unterschiedlichen Abstraktionsebenen betrachtet werden können. Möchte man mit seinem Mobiltelefon einen Anruf tätigen, so wird dies als einzelne Aktion behandelt. Von den Schritten wie „Telefonbuch öffnen“, „Kontakt auswählen“ und „Abheben drücken“, die nötig sind, um diese Aktion tatsächlich durchzuführen, wird zunächst abstrahiert. Das Resultat sind teilweise abstrakte Pläne, deren Komplexität auf Grund der kleineren Anzahl enthaltener Schritte geringer ist als bei entsprechenden, bis auf das primitive Level durchgeplanten Varianten.

2.1.1 Partial Order Causal Link Planning

Unter *Partial Order Causal Link* (kurz POCL) Planern versteht man eine Klasse von Planungsformalismen, deren Vertreter zwei Eigenschaften gemein haben. Erstens orientiert sich ihre Vorgehensweise an der Feststellung und Sicherung von kausalen Zusammenhängen zwischen den Schritten eines Plans, sogenannter *kausaler Links*. Zweitens führen diese Formalismen zu Plänen, deren Planschritte nicht vollständig, sondern lediglich *partiell angeordnet* sind. Wir stellen die grundsätzliche Funktionsweise des POCL-Planens am Beispiel des „Systematic Non-Linear Planning“-Formalismus (kurz SNLP) vor, dessen detaillierte Beschreibung man in [MR91] findet.

Grundlage für SNLP ist eine Definition von *Operatoren*, mit denen der Weltzustand verändert werden kann. Ein solcher Operator besitzt eine Liste von *Vorbedingungen*, die für seine Ausführbarkeit erfüllt sein müssen, sowie eine *Add-* und eine *Delete Liste*, die seine Auswirkungen auf den zu seiner Ausführungszeit herrschenden Zustand repräsentieren. Die Erzeugung von Planschritten, aus denen Pläne konstruiert werden können, erfolgt durch Instanziierung dieser Operatoren. Um Problemstellungen

formulieren zu können, werden in SNLP zwei spezielle Operatoren START und FINISH definiert, die jeweils als erster bzw. letzter Schritt im Plan verwendet werden. START besitzt weder Vorbedingungen noch eine Delete Liste, sondern nur eine Add Liste, und erzeugt damit einen Initialzustand. FINISH hingegen hat ausschließlich Vorbedingungen und legt damit Eigenschaften eines zu erreichenden Weltzustandes fest.

In einem Plan können zwei Planschritte t_j mit einer Vorbedingung ϕ, sowie t_i mit ϕ als Element seiner Add Liste, vorhanden sein. Der kausale Zusammenhang, dass diese Bedingung ϕ durch Ausführung von t_i vor t_j erfüllt werden kann, kann durch einen kausalen Link $\langle t_i, \phi, t_j \rangle$ explizit ausgedrückt werden (eine weitere gebräuchliche Darstellung für kausale Links ist $t_i \xrightarrow{\phi} t_j$).

Für einen solchen kausalen Link existiert eine *kausale Bedrohung*, wenn sich ein Schritt t_{threat} im Plan befindet, der ϕ in seiner Add- oder Delete Liste hat. Die Bedrohung durch ein Vorkommen von ϕ in der Delete Liste von t_{threat} besteht darin, dass dessen Ausführung zwischen t_i und t_j die bereits herstellbare Bedingung zunichte machen würde. Dass auch ein Auftreten von ϕ in der Add Liste als Bedrohung aufgefasst wird liegt darin begründet, dass ϕ dann mehrmals wahr gemacht würde, und zwar möglicherweise als letztes nicht von t_i sondern eben von t_{threat}, was der Bedeutung eines kausalen Links widerspricht. Die Auflösung einer solchen Bedrohung geschieht durch die Einführung einer *Ordnungsbedingung* an die Anordnung der Planschritte, dergestalt, dass t_{threat} vor t_i oder nach t_j ausgeführt werden muss. Die entsprechende Schreibweise für eine solche Ordnungsbedingung lautet $t_{threat} \prec t_i$ bzw. $t_j \prec t_{threat}$.

Ein Plan besteht in diesem Formalismus aus einer Menge von Planschritten sowie den zwischen diesen definierten kausalen Links und Ordnungsbedingungen, und beschreibt damit wie eingangs erläutert zunächst nur eine partielle Anordnung. Als Kriterium dafür, ob ein solcher Plan noch durch eine totale Anordnung seiner Schritte realisiert werden kann, wird der Begriff der *topologischen Sortierung* eingeführt, und definiert, dass ein Plan genau dann konsistent ist, wenn es eine solche topologische Sortierung für ihn gibt. Diese bezeichnet eine Linearisierung aller Schritte eines nicht-linearen Plans, die folgende Bedingungen erfüllt:

- Der erste Schritt der Linearisierung ist START.
- Der letzte Schritt der Linearisierung ist FINISH.
- Für jeden kausalen Link $t_i \xrightarrow{\phi} t_j$ im Plan gilt, dass t_i vor t_j angeordnet ist.
- Für jede Ordnungsbedingung $t_i \prec t_j$ des Plans gilt, dass t_i vor t_j angeordnet ist.

Eine Problemstellung wird im POCL-Planen durch die verfügbaren Operationen zur Manipulation der Welt sowie einen Plan definiert, der nur die oben beschriebenen Planschritte START und FINISH enthält. Die Aufgabe, ein solches Problem zu

lösen, besteht dann darin, einen konsistenten Plan zu finden, der zwei Kriterien genügt:

- Für jede Vorbedingung ϕ eines Planschritts t_j besitzt der Plan einen kausalen Link $\langle t_i, \phi, t_j \rangle$
- Für jede kausale Bedrohung durch einen Schritt t_{hreat} eines kausalen Links $\langle t_i, \phi, t_j \rangle$ spezifiziert der Plan entweder eine Ordnungsbedingung $t_{hreat} \prec t_i$ oder $t_j \prec t_{hreat}$

Ein Plan, der diese Kriterien erfüllt, wird als „vollständiger“ Plan bezeichnet, und jede seiner möglichen topologischen Sortierungen stellt eine Lösung dar. Abbildung 2.1 zeigt einen partiell geordneten Plan zur Einrichtung eines neuen Kontakts im Adressbuch eines Mobiltelefons mit einem Namen und einer Telefonnummer. Aus diesem Plan lassen sich beispielsweise die Lösungen
$(START \prec set\_Contact.Name \prec set\_Contact.Connection \prec Configure\_Contact \prec FINISH)$ und
$(START \prec set\_Contact.Connection \prec set\_Contact.Name \prec Configure\_Contact \prec FINISH)$ ableiten.

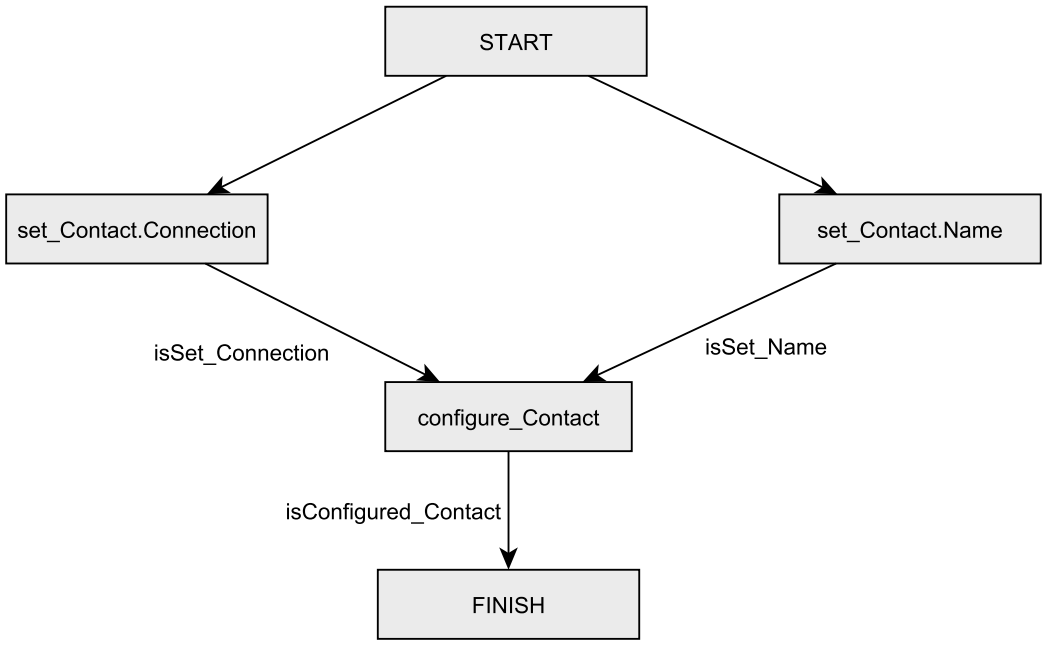


Abbildung 2.1: Ein einfacher POCL-Plan aus fünf Tasks und den kausalen Links $set\_Contact.Name \xrightarrow{isSet\_Name} Configure\_Contact$, $set\_Contact.Connection \xrightarrow{isSet\_Connection} Configure\_Contact$ und $configure\_Contact \xrightarrow{isConfigured\_Contact} FINISH$

Um eine solche Lösung zu finden, wiederholt SLNP nun, ausgehend vom Plan der nur START und FINISH beinhaltet, folgende Schritte so lange, bis ein vollständiger Plan vorliegt:

1. Ist der Plan inkonsistent, so kehre zum vorhergehenden Plan zurück
2. Führe für eine kausale Bedrohung eine entsprechende Ordnungsbedingung ein und untersuche den resultierenden Plan
3. Erzeuge für eine Vorbedingung, für die noch kein kausaler Link definiert wurde, einen solchen kausalen Link entweder zu einem geeigneten, bereits im Plan vorhandenen, oder aber zu einem neu einzufügenden Planschritt, und untersuche den resultierenden Plan

2.1.2 Hierarchical Task Network Planning

Wie POCL steht auch *Hierarchical Task Network* oder kurz HTN-Planen als Ansatz maschinellen Planens stellvertretend für Systeme, die sich durch eine ähnliche Grund-Vorgehensweise auszeichnen. Bei HTN-Planern besteht diese darin, Handlungsplanung durch hierarchische Zerlegung komplexer Aufgaben in immer kleinere und letztlich primitive Bestandteile zu realisieren. Wir geben im Folgenden eine Einführung in HTN-Planen am Beispiel des Universal Method-Composition Planners, kurz UMCP. [EHN94] beschreibt den Formalismus im Detail.

Die Ausführung von Aktionen wird auch in UMCP durch die Instanziierung definierter Operationen, hier *Tasks* genannt, realisiert. UMCP verwendet drei unterschiedliche Ausprägungen dieser Tasks:

- *Primitive Tasks* stehen für einzelne, direkt ausführbare Aktionen in der Welt, und verfügen dazu über eine Liste von Variablen für die beeinflussten Domänenelemente
- *Zieltasks* stellen durch ein enthaltenes Literal jeweils ein Ziel dar, welches der Planer erreichen und damit eine Bedingung, die er aktiv wahr machen muss
- *Komplexe Tasks* modellieren Handlungen, die aus mehreren Schritten, oder Ziele, die aus mehreren Bedingungen bestehen

Ziel- und komplexe Tasks werden entsprechend dieser Definitionen auch als *abstrakte* Tasks bezeichnet. Zur Umsetzung solcher abstrakten Tasks dienen sogenannte *Tasknetze*, die eine Kombination von Tasks beschreiben, die den abstrakten Task realisieren kann. Ein solches Tasknetz besitzt darüber hinaus *Wahrheitsbedingungen*, deren Erfüllung der Planer sicherstellen muss, *Ordnungsbedingungen* an die Anordnung der enthaltenen Tasks zueinander, und *Variablenbedingungen*, die die Bindung von Variablen aneinander oder an Konstanten festlegen. Abstrakte Tasks werden durch *Methoden* mit den zu ihrer Umsetzung geeigneten Tasknetzen verknüpft, wobei für jeden abstrakten Task mindestens eine solche Methode definiert

sein muss. Abbildung 2.2 zeigt ein Beispiel für das Zusammenspiel von abstraktem Task, Methode und dem hier nur aus zwei primitiven Tasks bestehenden Tasknetz. Ähnlich wie für partiell geordnete Pläne im POCL-Planen wird auch für Tasknetze im HTN-Planen ein Konsistenzkriterium definiert. Ein Tasknetz ist genau dann konsistent, wenn es eine *Vervollständigung* genannte totale Ordnung seiner Schritte unter Beachtung aller seiner Wahrheits-, Ordnungs- und Variablenbedingungen gibt, die also ausführbar ist.

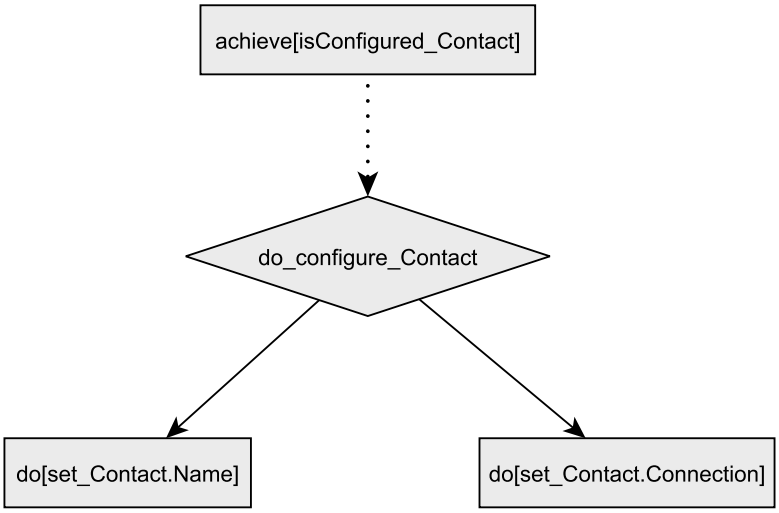


Abbildung 2.2: Die Methode do_configure_Contact zerlegt den Zieltask achieve[isConfigured_Contact] in zwei primitive Tasks

Ein Problem für UMCP wird durch eine Domäne – das heißt die verfügbaren Tasks, Methoden und Tasknetze –, einen initialen Weltzustand sowie ein initiales Tasknetz, welches die zu erfüllenden Aufgaben angibt, spezifiziert. Lösungen für ein solches Problem sind alle Vervollständigungen eines konsistenten Tasknetzes, welches nur aus primitiven Tasks besteht und aus dem initialen Tasknetz abgeleitet wurde. UMCP sucht nach solchen Lösungen, indem das aktuelle Tasknetz TN auf Konflikte, wie beispielsweise das Vorhandensein abstrakter Tasks, untersucht wird, und diese Konflikte nach und nach gelöst werden:

- Besteht TN nur aus primitiven Tasks und gibt es eine Vervollständigung von TN, so ist diese eine Lösung. Gibt es keine Vervollständigung, kehre zum vorherigen Tasknetz zurück
- Bestimme die im Tasknetz herrschenden Konflikte
- Löse einen oder mehrere Konflikte durch Entfernen eines abstrakten Task und Hinzufügen des Tasknetzes einer geeigneten Methode der Domäne, und untersuche das resultierende Tasknetz

2.1.3 PANDA

Die im Rahmen dieser Arbeit entwickelten Ideen sowie deren prototypische Implementierung basieren auf Plänen, die das am Institut für Künstliche Intelligenz der Universität Ulm entwickelte Planungssystem PANDA ([BBS10]), ein Vertreter der domänenunabhängigen, hybriden Planungssysteme, erzeugt hat. Unter Verwendung eines Fragments der Prädikatenlogik erster Stufe als formaler Semantik für die Planungssprache kombiniert es die Ansätze des HTN- und POCL-Planens und entwickelt dabei eine Problemlösung mittels Planverfeinerung, modifiziert also eine Grundversion eines Plans so lange bis eine Lösung entstanden ist. Die Beschreibung des Systems selbst sowie der in dieser Arbeit angestellten Überlegungen wird sich an Beispielen orientieren, die aus der Smartphone-Domäne stammen, welche zu diesem Zweck im Rahmen dieser und weiterer Arbeiten für PANDA entworfen wurde. In dieser wird die Ausführung unterschiedlich komplexer Aufgaben mit einem Smartphone modelliert, wie beispielsweise das Absetzen von Anrufen oder Nachrichten, aber auch die Organisation von Meetings. Eine genaue Beschreibung der Domäne findet sich in Kapitel 3.

Im nächsten Abschnitt sollen zuerst die Konzepte vorgestellt werden, die zur Realisierung des Planungssystems PANDA verwendet werden. Es folgen die darauf aufbauenden Beschreibungen von Problemstellungen, Plänen sowie Lösungen, und schließlich eine Darlegung des verwendeten Planungsalgorithmus'.

Grundlegende Konzepte

Wie beim maschinellen Planen üblich wird die Welt auch in PANDA als System von Zuständen und Zustandsübergängen modelliert, welche die Informationen über bestehende und sich ändernde Attribute der betrachteten Domäne beinhalten. Ein Zustand ist definiert über die in ihm geltenden Eigenschaften der modellierten Welt, wird also durch eine Menge von positiven prädikatenlogischen Atomen charakterisiert.

Die *Sorten*, auf denen Parameter und Konstanten als Bestandteile dieser Atome fußen, beschreiben Typen von Objekten, wie beispielsweise `EMail`. Neben solchen realen sind jedoch auch abstrakte Sorten wie `Message` möglich. Damit können Objektklassen beschrieben werden, die nicht real existieren, jedoch gewisse gemeinsame Eigenschaften besitzen. Um solche Gemeinsamkeiten in den Planungsprozess einfließen lassen zu können bietet PANDA Sortenhierarchien an. `EMail` kann dabei nicht nur als Unterklasse von `Message` deklariert werden, sondern auch Ober- oder Unterklasse weiterer Typen sein, so dass multiple Vererbung und damit die Modellierung komplexer Sortenhierarchien möglich ist.

Um die zur Beschreibung von Zuständen nötigen Atome zu bilden werden auf diesen Sorten aufbauende *Relationen* verwendet. Diese repräsentieren Aussagen über

Objekte und ermöglichen die Wiedergabe von Beziehungen, in denen diese zueinander stehen. Ein Beispiel hierfür ist die Relation `associated_contact:Contact × Person`, die angibt, dass ein Eintrag im Telefonbuch eines Mobiltelefons mit einer echten Person assoziiert wird. Relationen können jedoch auch nullstellig sein, also keine Aussagen über konkrete Objekte machen, sondern allgemeine Eigenschaften eines Zustands ausdrücken. Beinhaltet eine Zustandsbeschreibung beispielsweise `inMode_People()`, so sagt dies aus, dass sich das Mobiltelefon in einem bestimmten Modus befindet, von dem aus spezielle Operationen wie beispielsweise das Anrufen von besonders häufig kontaktierten Personen möglich sind.

PANDA betrachtet seine Zustände stets unter der *Closed World Assumption*, nimmt also an, dass in einem Zustand σ von jedem Literal λ, welches nicht in seinen Merkmalen enthalten ist, die Negation $\neg\lambda$ gilt. Die Möglichkeit von Zustandsübergängen realisiert PANDA mit dem Konzept sogenannter *Tasks*. Ein solcher Task ist definiert durch ein Tupel $t(v_1, ..., v_n) = \langle prec, post \rangle$. Über seinen Parametern v sind zwei Formeln *prec* und *post* definiert, seine *Vorbedingung* und *Effekte*. Die Vorbedingung verkörpert die Eigenschaften die ein Zustand besitzen muss, damit der Task ausgeführt werden kann. Die Effekte hingegen geben die Auswirkungen auf die Welt an, die der Task bei seiner Anwendung hat, und die, zusammen mit den vor der Taskanwendung geltenden Attributen, die Merkmale des neu erstellten Zustands bestimmen.

Auf Basis dieser Tasks kann nun ein *Plan* definiert werden als ein Tupel $P = \langle S, \prec, VC, CL \rangle$, mit einer Menge von Planschritten S aus den verfügbaren Tasks, die durch eine Menge von *Ordnungsbedingungen* $\prec$ in eine Halbordnung gebracht werden. Eine Menge von *kausalen Links* CL beschreibt explizit die kausalen Zusammenhänge, die zwischen den Planschritten bestehen. Schließlich enthält ein Plan *(Non-)Codesignationen* VC über die Parameter der beinhalteten Tasks, gibt also Gleichheits- und Ungleichheitsbeziehungen zwischen den Parametern an.

Als hybrides Planungssystem, dessen Ursprünge auch im HTN-Planen liegen, kennt PANDA Tasks nicht nur in primitiver, sondern auch in abstrakter Form. Der Aufbau *abstrakter Tasks* ist hierbei der gleiche wie der von primitiven Tasks, die Domäne muss jedoch für jeden abstrakten Task mindestens eine *Methode* spezifizieren. Methoden bilden prozedurales Wissen über die Domäne in der Form ab, dass sie eine gültige Ersetzung eines abstrakten Tasks durch ein Planfragment definiert. In der Smartphone-Domäne gibt es beispielsweise den abstrakten Task `call(Contactable)`, für dessen Implementierung mehrere Möglichkeiten bekannt sind: die Methode `do_call_EnterNumber` gibt eine manuelle Eingabe der Nummer als Lösungsweg vor, `do_call_SelectContact` hingegen fügt Tasks ein, die den entsprechenden Kontakt aus dem Telefonbuch auswählen und den Anruf auf diesem Weg starten. Methoden definieren neben einem Tasknetz auch Codesignationen zwischen den Parametern des implementierten Tasks und denen der Subtasks und Ordnungsbedingungen über letzteren. Schließlich können in einer Methode auf

Basis des prozeduralen Wissens über die jeweilige Implementierung die aus dem HTN-Planen bekannten kausalen Links verankert werden.

Im Gegensatz zu anderen Planungssystemen sind bei PANDA abstrakte Tasks und ihre Implementierungen semantisch stark verbunden. Als hierarchisch übergeordnetes Element ist der abstrakte Task ein Stellvertreter im Plan für alle seine möglichen Konkretisierungen, und sollte daher auch Pendants zu deren Vor- und Nachbedingungen besitzen, um die Ausführbarkeit des Plans auf jedem Abstraktionslevel sicherzustellen. Zu diesem Zweck unterstützt das System sogenannte *Dekompositions-Axiome*, die Relationen aus den Vorbedingungen abstrakter Tasks in Beziehung zu entsprechenden Zusammenfassungen von Zustandsattributen setzt. Die den Methoden zugehörigen Planfragmente stellen auf Basis dieser Axiome nur dann gültige Implementierungen dar, wenn die Konjunktion der Vorbedingungen der enthaltenen Tasks eines der Disjunktionsglieder der rechten oder aber das Atom der linken Seite beinhaltet. Für ein Beispiel zur Anwendung eines Dekompositionsaxioms sei ein Zustand mit folgenden Merkmalen gegeben:

```
associated_Contact('Hans_Büro', 'Hans Klein')
isSet_Contact.Number('Hans_Büro')
```

Der Task `do_call('Hans Klein')` mit der Vorbedingung `isCallable('Hans Klein')` ist in diesem Zustand ausführbar, obwohl diese Bedingung nicht explizit erfüllt ist. Möglicht macht das ein Dekompositionsaxiom für die Relation `isCallable: Contactable`, welches mit den beschriebenen Methoden eine sinnvolle Zerlegung des abstrakten Tasks ergibt:

$$
\begin{aligned}
isCallable\,(?Contactable) \Leftrightarrow\ & known\_Contactable.Mobile\,(?Contactable) \vee \\
& known\_Contactable.Office\,(?Contactable) \vee \\
& \exists\, Contact\ ?Contact : \\
& \quad associated\_Contact\,(?Contact, ?Contactable) \wedge \\
& \quad isSet\_Contact.Number\,(?Contact)
\end{aligned}
$$

Das Dekompositionsaxiom besagt also, dass um eine Person anrufen zu können entweder eine ihrer Telefonnummern bekannt, oder aber ein ihr zugeordneter Eintrag im Telefonbuch mit einer solchen Nummer existieren muss – was im gegebenen Zustand erfüllt ist. Daher kann die Methode `do_Call_SelectContact` angewendet werden, die den Anruf durch die Auswahl des entsprechenden Eintrags aus dem Telefonbuch umsetzt und deren Subtasks diese Zerlegung als Vorbedingungen besitzen.

Aufbauend auf der Beschreibung von Domänenbestandteilen und Plänen kann nun die Definition von *Problemen* und *Lösungen* für PANDA erfolgen. Ein solches Planungsproblem $\pi = \langle D, S_{init}, S_{goal}, P_{init} \rangle$ bezieht sich zunächst auf eine Domäne D, in der es verankert ist. Weiterhin besteht es aus einem *initialen Zustand* S_{init}, einer Formel, die die zu Beginn des Planungsprozesses geltenden Attribute angibt,

und einer *Zielzustandsbeschreibung* S_{goal}, welche die Anforderungen an eine Lösung festlegt. Schließlich gibt es noch ein *initiales Tasknetz* P_{init}, welches die von einer Lösung zu implementierenden Tasks beinhaltet und damit den Ausgangspunkt für den Planungsvorgang bildet.

Eine Lösung für ein Planungsproblem $\pi = \langle D, S_{init}, S_{goal}, P_{init} \rangle$ ist in PANDA ein Plan $P_{sol} = \langle S_{sol}, \prec_{sol}, VC_{sol}, CL_{sol} \rangle$, der folgende Eigenschaften erfüllt:

- S_{sol} ist eine Verfeinerung des initialen Tasknetzes P_{init}, ist also durch eine Reihe von Veränderungen am Plan aus P_{init} hervorgegangen
- S_{sol} enthält nur primitive Tasks, deren enthaltene Parameter alle durch VC_{sol} an Konstanten gebunden sind
- Die Ordnungsbedingungen in $\prec_{sol}$ sind zyklenfrei
- Die (Non-)Codesignationen in VC bilden eine konsistente Menge von Bedingungen
- Die kausalen Links CL_{sol} sind konsistent mit den Ordnungsbedingungen $\prec_{sol}$, d.h. für jeden kausalen Link $\langle t_i, \phi, t_j \rangle \in CL_{sol}$ gibt es eine (implizite oder explizite) Ordnungsbedingung $t_i \prec t_j \in \prec_{sol}$
- Für jede Vorbedingung ϕ eines Tasks $t_j \in S_{sol}$ gibt es einen Task $t_i \in S_{sol}$ und einen kausalen Link $\langle t_i, \phi, t_j \rangle \in CL_{sol}$
- Der Plan ist frei von kausalen Bedrohungen, d.h. die Ordnungsbedingungen $\prec_{sol}$ des Plans erlauben keine Anordnung zweier Planschritte $t_{threat} \prec t_j$ mit $\langle t_i, \phi, t_j \rangle \in CL_{sol}$ und $\neg\phi \in post_{t_{threat}}$

Planungsverlauf

Der Algorithmus den PANDA zur Entwicklung einer Lösung verwendet besteht aus vier Schritten:

1. Selektion eines Plans zur Weiterentwicklung
2. Detektion der *Defizite* des Plans
3. Bestimmung aller passenden *Verfeinerungsmöglichkeiten*, die zur Behebung dieser *Defizite* angewendet werden können
4. Anwendung einer dieser *Verfeinerungsmöglichkeiten* je *Defizit* und Einordnung der auf diesem Weg verfeinerten Pläne in die Fringe

Der erste Schritt besteht also in der Auswahl eines Plans aus der Menge von Plänen, die momentan als Kandidaten zur Weiterentwicklung zur Auswahl stehen. Diese Menge wird als *Fringe* bezeichnet und enhält zu Beginn des Planungsprozesses nur den Plan P_{init} der initialen Problemstellung. Entsprechend der oben beschriebenen Anforderungen an eine Lösung für ein gegebenes Problem besitzen Pläne, die keine solche Lösung darstellen, sogenannte *Defizite*. Ein Defizit ist eine explizite Repräsentation einer Eigenschaft eines der Schritte t_i eines Plans, an der das Planungssystem noch eine Verfeinerung vornehmen muss, und kann eine der vier folgenden Ausprägungen haben:

- *Abstrakter Task*: t_i ist nicht primitiv
- *Offene Vorbedingung*: t_i hat eine Vorbedingung, für die noch kein sichernder kausaler Link im Plan existiert
- *Fehlende Parameterbindung*: t_i hat einen Parameter, der noch nicht an eine Konstante gebunden wurde
- *Kausale Bedrohung*: es existiert ein kausaler Link $\langle t_j, \phi, t_i \rangle$ und ein Task t_{threat} mit $\neg\phi \in post_{t_{threat}}$, und der Plan erlaubt die Anordnung $t_j \prec t_{threat} \prec t_i$ dieser Tasks

Um diese Defizite zu beheben und damit den betrachteten Plan weiterzuentwickeln kennt das System für jede Art von Defizit eine Anzahl von *Verfeinerungsmöglichkeiten*. Die Anwendung einer solchen Verfeinerung bringt jeweils einen Nachfolgeplan ohne das behandelte, potentiell jedoch mit neuen Defiziten hervor, der anschließend in die Fringe einfügt wird. Abbildung 2.3 zeigt die Durchführung eines solchen Planungsschritts. Das System wiederholt nun die angegebenen Schritte so lange, bis die Fringe entweder eine Lösung enthält, oder aber leer ist, was bedeutet dass keine Planentwicklung zu einer Lösung geführt hat.

2.2 Wissensrepräsentation

„The W3C OWL 2 Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things.“[1]

Die maschinenlesbare Repräsentation von Wissen ist ebenso wie Handlungsplanung ein Bereich der künstlichen Intelligenz, der schon seit geraumer Zeit intensiv erforscht wird. Eine solche Repräsentation soll nicht nur dazu dienen, Computersystemen grundsätzlich Wissen zur Verfügung zu stellen. Zum einen ermöglicht sie diesen darüber hinaus den Austausch von Wissen untereinander, zum anderen können

[1] Aus der Einleitung von [KPSR+09]

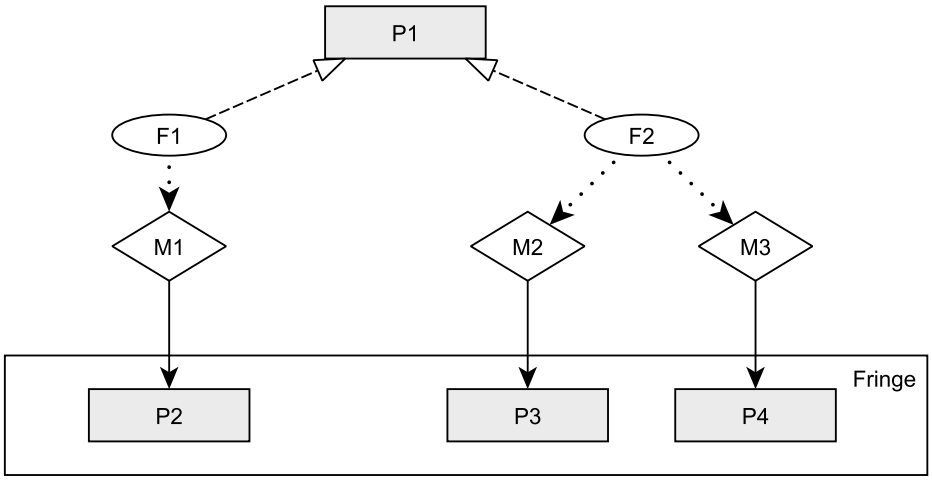


Abbildung 2.3: Ein Planungsschritt des PANDA-Systems.
Aus einem Plan $P1$ mit zwei Defiziten $F1$ und $F2$, für die die Verfeinerungen $M1$ bzw. $M2$ und $M3$ gefunden wurden, werden durch Anwendung dieser Verfeinerungen drei Nachfolgepläne $P2$, $P3$ und $P4$ entwickelt, die nun den Inhalt der *Fringe* bilden

auf diese Weise Inferenzalgorithmen dazu genutzt werden, weitere im bestehenden Wissen implizierte Fakten abzuleiten und explizit zu machen.

2.2.1 Beschreibungslogik

Ein Formalismus zur Wissensrepräsentation, der große Verbreitung gefunden hat, ist die Beschreibungslogik. Wie bei jeder Wissensrepräsentation ist auch bei der Beschreibungslogik abzuwägen, wo man sich auf der Skala der Expressivität der Sprache einerseits und ihrer Entscheidbarkeit andererseits einordnen soll. Je mehr Ausdrucksmöglichkeiten die Sprache zur Verfügung stellt, desto komplexere Sachverhalte können durch sie modelliert werden. Gleichzeitig verringern sich jedoch mit einer Erweiterung der Ausdrucksfähigkeit die Möglichkeiten, auf dem so erzeugten Modell Schlussfolgerungsmechanismen anzuwenden.
Beschreibungslogik ist, was diese Aspekte betrifft, wie folgt einzuordnen. Ihre Mächtigkeit ergibt sich aus ihrer Basis, die in $\mathcal{ALC}$ – der *Attributive Language* $\mathcal{AL}$, ergänzt um $\mathcal{C}$ für das Komplement – liegt, die die Definition von Konzepten nach folgender Regel erlaubt, wobei A, C und D Konzepte und R eine Rolle darstellen:

$$
\begin{array}{lll}
C, D & \rightarrow & A \mid \qquad\qquad (atomares\ Konzept) \\
 & & \top \mid \qquad\qquad (top) \\
 & & \bot \mid \qquad\qquad (bottom) \\
 & & \neg A \mid \qquad\qquad (atomare\ Negation) \\
 & & C \sqcap D \mid \qquad\qquad (Schnittmenge) \\
 & & \forall R.D \mid \qquad\qquad (Wertrestriktion) \\
 & & \exists R.D \qquad\qquad (Existenzquantifikation)
\end{array}
$$

Die Bedeutung der Definition eines Konzeptes C durch $\forall R.D$ entspricht der des prädikatenlogischen Axioms $C(x) \Leftrightarrow (\forall y(R(x,y) \Rightarrow D(y)))$, und analog dazu ist $\exists R.D$ äquivalent zu $C(x) \Leftrightarrow (\exists y(R(x,y) \wedge D(y)))$. *top* beschreibt hier das Konzept, das alle im Modell beinhalteten Konzepte beinhaltet, *bottom* hingegen das leere Konzept. Diese Konzepte haben vor allem für das *Reasoning* Bedeutung, also die Schlussfolgerungsmechanismen, die später vorgestellt werden. Die Familie der beschreibungslogischen Sprachen bietet eine Vielzahl von Erweiterungen der hier gezeigten Grundlage, die je nach Bedarf der Ausdrucksmöglichkeiten eingesetzt werden können. Wir beschränken uns für die Vorstellung von Beschreibungslogik an dieser Stelle jedoch auf den Basisumfang, und beschreiben im Folgenden zunächst den Aufbau einer Wissensrepräsentation mittels Beschreibungslogik, und anschließend die Möglichkeiten zur Schlussfolgerung, die darauf angewendet werden können.

Zweigeteilte Modellierung

Die beschriebenen Sprachelemente dienen der Definition der *Terminological Box*, kurz *T-Box*, in welcher das Vokabular der repräsentierten Domäne mit Hilfe von *Konzepten* und *Rollen* definiert wird. *Konzepte* bilden eine Hierarchie von Klassen, auf deren Basis die Objekte der Domäne beschrieben und eingeordnet werden. Beispielsweise könnte eine Domäne das Konzept `Message` mit einem zugehörigen Unterkonzept `EMail` beinhalten. *Rollen* stützen sich in ihrer Definition auf die beschriebenen Konzepte und bezeichnen binäre Relationen zwischen diesen, das heißt Verhältnisse, in denen sie zueinander stehen. Eine Rolle `attached(Message, Information)` könnte zum Beispiel ausdrücken, dass eine `Information` an eine `Message` angehängt ist.

In der *Assertional Box* oder kurz *A-Box* befinden sich Aussagen in Form von Konzept- oder Rollenzusicherungen über den Weltausschnitt, den die Domäne repräsentiert, genauer über die *Individuen* in diesem, unter Verwendung des Vokabulars der T-Box. Damit lässt sich beispielsweise ausdrücken, dass Hans Klein eine Person ist und eine Information über den Zeitpunkt seines wöchentlichen Meetings besitzt:

```
Person(Hans Klein)
Time(WeeklyMeeting_0815am)
haveInformation(Hans Klein, WeeklyMeeting_0815am)
```

Diese Trennung von Terminologie und Individuen ist aus zwei Gründen sinnvoll. Erstens ist es hilfreich für die Modellierung von Domänen, eine solche Trennung vorzunehmen. Ein Entwickler kann auf diese Weise zunächst eine sinnvolle Terminologie aufbauen, um sie anschließend mit Daten von Individuen zu füllen. Der zweite Punkt betrifft die anzuwendenden Schlussfolgerungsmechanismen, die durch eine solche Aufspaltung jeweils auf den einen oder anderen Teil des Modells spezialisiert und damit in ihrer Effizienz verbessert werden können.

Schlussfolgerungsmechanismen

Wir beschreiben nun die Schlussfolgerungsmechanismen, die auf beschreibungslogischen Sprachen anwendbar sind, und die wir entsprechend der vorangegangenen Erläuterungen an der ihnen zu Grunde liegenden T- bzw. A-Box unterscheiden.
Schlussfolgerungsmechanismen die die T-Box betreffen können grundsätzlich einem von zwei Zwecken dienen. Eine Überprüfung der *Erfüllbarkeit* aller Konzepte, ob es also überhaupt ein Individuum geben kann, welches ihren jeweiligen Restriktionen genügt, wird zum Nachweis der Konsistenz der T-Box durchgeführt. Daneben kann die Terminologie um Informationen erweitert werden, indem geprüft wird, ob zwei Konzepte *äquivalent* sind oder ein Konzept ein anderes *subsumiert*.
Auch auf der A-Box kann Reasoning zur Konsistenzüberprüfung betrieben werden, beispielsweise um Widersprüche in darin vorgenommenen Zusicherungen aufzudecken. Dies geschieht, indem bei der *Realisierung* das speziellste Konzept bezüglich der Subsumtionshierarchie für ein gegebenes Individuum bestimmt wird. Ist dieses das *bottom*-Konzept, d.h. kann das Individuum keinem Konzept der T-Box zugeordnet werden, ist die A-Box inkonsistent. Daneben ist jedoch auch die Behandlung von *Anfragen* relevant, die beispielsweise alle diejenigen Individuen geliefert bekommen wollen, die einem bestimmten Konzept angehören. Beide genannten Tests können auch durch *Instanztests* realisiert werden, die ganz allgemein Auskunft darüber geben, ob ein gegebenes Individuum eine Instanz eines ebenfalls gegebenen Konzepts darstellt.

Bei der Ableitung von Wissen mit Hilfe dieser Tests ist von großer Bedeutung, dass bei Beschreibungslogiken eine andere Position in der Betrachtung ihrer Welt eingenommen wird, als es beispielsweise bei Datenbanken der Fall ist. Eine Datenbank beschreibt einen geschlossenen Abschnitt dieser Welt, weshalb Systeme, die Anfragen an solche Datenbanken realisieren, annehmen, dass von allen Fakten, die nicht darin enthalten sind, das Gegenteil gilt.

Betrachten wir als Beispiel eine Domäne mit einem Konzept Person und nur einer Aussage $Person(HansKlein)$, so kann ein solches System beispielsweise eine Anfra-

ge, die wissen will, ob auch das Individuum $Hans\,Klein$ dem Konzept Person angehört, mit „Nein“ beantworten, denn diese Eigenschaft ist nicht explizit zugesichert, was bedeutet dass das Gegenteil der Fall sein muss. Ein Reasoner für eine Beschreibungslogik ist zu diesem Schluss nicht in der Lage, da er das Wissen im betrachteten Modell zu keinem Zeitpunkt als vollständig betrachtet. Von beliebiger Quelle, sei es eigene Schlussfolgerung oder eine Zusicherung von außen, könnte eine Aussage $Person(Hans\,Klein)$ hinzukommen, so dass eine vorher gemachte Ableitung sich als nicht korrekt herausstellen würde. Diese „Open World Assumption“ genannte Sichtweise hat weitreichende Folgen für die Verwendung von beschreibungslogischen Systemen, die sich ebenso wie weitere Details zu Beschreibungslogiken allgemein beispielsweise aus [BCM+07] entnehmen lassen.

2.2.2 Web Ontology Language

Die *Web Ontology Language*, kurz OWL, wurde vom World Wide Web Consortium im Zuge ihrer Bemühungen vorgestellt, die Grundlagen für ein „Semantic Web“ zu schaffen. Dafür wurde ein allgemeines „Resource Description Framework“ (kurz RDF) entwickelt, welches eine Formalisierung für die Beschreibung von Daten bietet, mit dem Ziel, diese zwischen Anwendungen, Web-Seiten etc. beliebig austauschen zu können, und dafür insbesondere die Daten zusätzlich zu ihrer bereits gegebenen Syntax mit einer für Maschinen „verständlichen“ Semantik zu versehen. OWL ist nun eine Anwendung dieses Frameworks und dient dazu, in Form einer sogenannten „Ontologie“ einen Weltausschnitt in einer Form zu beschreiben, die durch eine RDF-basierte Semantik maschinenlesbar bleibt.

Neben der RDF-basierten gibt es auch eine *direke Semantik*, die die Interpretation von OWL-Axiomen als beschreibungslogische Aussagen ermöglicht, so dass die im vorhergehenden Kapitel beschriebenen Schlussfolgerungsmechanismen zur Verarbeitung des in der Ontologie modellierten und zur Inferenz neuen Wissens verwendet werden können. Die Verwendung dieser direkten Semantik für OWL, die in [PSMG09] genau beschrieben wird, wird durch einen Aufbau der OWL-Elemente möglich, der von der Beschreibungslogik abgeleitet ist, und deren Elementen daher sehr nahe kommt – Konzepte werden in OWL als *Klassen*, Rollen als *Eigenschaften* bezeichnet, während der Begriff von *Individuen* auch in OWL verwendet wird. Diese Semantik ist jedoch nur gültig, wenn man die Ausdrucksmächtigkeit von OWL einschränkt. Es wurden dazu eine Anzahl von Profilen von OWL definiert, die diese Einschränkung vorschreiben, wobei OWL in seinem vollen Umfang als *OWL Full* bezeichnet wird. Die direkte Semantik lässt sich auf das Profil *OWL DL* anwenden, welches einer beschreibungslogischen Sprache $\mathcal{SROIQ}$ entspricht. Gegenüber der im vorangegangenen Abschnitt vorgestellten Sprache $\mathcal{ALC}$ bietet OWL DL damit einige Erweiterungen bezüglich Eigenschaften, die hier als transitiv oder invers zu anderen Eigenschaften deklariert und in Eigenschafts-Hierarchien verankert werden können, sowie bezüglich Werteinschränkungen, die durch klassenbeschränkte

Quantifizierung und namentliche Einschränkungen erweitert wurden. Neben OWL DL gibt es noch drei weitere Profile *OWL EL*, *QL* und *RL*, die weitere Einschränkungen der Mächtigkeit der Sprache zugunsten einer besseren Berechenbarkeit vornehmen, und daher ebenfalls unter der direkten Semantik betrachtet werden können. Ihr jeweiliger Umfang lässt sich beispielsweise [KPSR$^+$09] entnehmen. Neben unterschiedlichen Semantiken kann OWL auch mit Hilfe unterschiedlicher Syntaxen ausgedrückt werden, die jeweils unterschiedlichen Zwecken dienen, so zielt die *Manchester-Syntax* beispielsweise darauf ab, Aussagen möglichst lesbar darzustellen – zwingend unterstützt werden muss von Systemen die mit OWL arbeiten jedoch nur die XML-basierte *RDF/XML*-Syntax.

Der Aufbau einer OWL Ontologie ist auf Grund der Nähe zu Beschreibungslogiken sehr ähnlich zu einer damit formulierten Ontologie. Man definiert Konzepte und Eigenschaften, die anschließend verwendet werden können um komplexere Konzepte zu beschreiben. OWL bietet zu diesem Zweck drei Arten von Eigenschaften an. *Objekt-Eigenschaften* entsprechen den von $\mathcal{ALC}$ bekannten Rollen, die Aussagen über die Beziehung zwischen zwei Konzepten ermöglichen. *Datentyp-Eigenschaften* haben als Wertebereich keine Objekte sondern primitive Datentypen wie beliebige Zeichenketten, Fließkommazahlen etc. *Annotations-Eigenschaften* schließlich sind nicht für die Schlussfolgerung relevant, sondern erlauben es, Bestandteile von Ontologien mit Kommentaren zu versehen. Konzepte können nicht nur mit Objekt-Eigenschaften, sondern auch mit weiteren in der Logik verbreiteten Attributen wie Äquivalenz und Disjunktheit zu anderen Konzepten sowie ihrer Einordnung in eine Konzept-Hierarchie beschrieben werden.

3 Die Smartphone-Domäne

In dieser Arbeit wird häufig auf Elemente der Smartphone-Domäne für das PANDA-System Bezug genommen, die uns als Grundlage für konkrete Beispiele sowie als Anwendungsfall für die prototypische Implementierung dient. Die Idee hinter der Modellierung dieser Domäne ist, dass ein Smartphone dazu verwendet werden soll, verschiedene organisatorische Aufgaben durchzuführen. Zu diesen gehören einerseits Aktivitäten wie das Anlegen von Terminen, Erinnerungsereignissen und ähnlichem für den Nutzer des Geräts selbst, andererseits die Organisation beispielsweise von Meetings, zu denen weitere Teilnehmer einzuladen sind. Diese müssen im Vorfeld einer solchen Veranstaltung mit gewissen Informationen versorgt werden, wozu so grundlegende Dinge wie Ort und Zeit zählen, aber auch Tagesordnungspläne, Handouts etc.

Ein planbasiertes Assistenzsystem kann den Nutzer eines Smartphones dabei auf zwei Arten unterstützen. Zum einen können grundsätzliche Handlungspläne gegeben werden, die ihm bei der Verwendung von bisher unbekannten Funktionen des Geräts unterstützen. Zum anderen kann ein solches System aber auch sicherstellen, dass alle Schritte, die zum Beispiel für die Einladung mehrerer Personen zu einem Meeting, deren Versorgung mit den vorab benötigten Informationen, und dem Erstellen eines Eintrags im Kalender des Smartphones, mit dem der Nutzer selbst an das Meeting erinnert wird, auch wirklich durchgeführt werden.

Um diese Aktivitäten für das PANDA-System planbar zu machen, werden in der Domäne daher sowohl Elemente modelliert, die zur Abbildung der Verwendung von Informationen durch den Nutzer selbst nötig sind, um eben Kalendereinträge, Aufgaben, aber zum Beispiel auch Telefonbucheinträge für zu kontaktierende Personen zu erstellen, als auch solche Elemente, die zur Übermittlung von Informationen an andere Personen mit Hilfe des Smartphones dienen. Das Abstraktionsniveau der Modellierung reicht dabei von sehr abstrakten Tasks zur Übertragung verschiedener Arten von Nachrichten bis hinunter auf die Ebene der Betätigung von Bedienelementen des Smartphones. Wir geben in den nächsten Abschnitten einen Überblick über die Bestandteile der Domäne, wobei wir meist eine bestimmte Variante prototypisch für die Umsetzung eines ihrer Aspekte vorstellen, und beschreiben kurz, was für Planungsprobleme in der Domäne umgesetzt werden können.

Grundbestandteile der Domäne

Eine zentrale Rolle in der Smartphone-Domäne spielen Informationen verschiedener Art, die sich auf zwei Weisen unterteilen lassen. Eine Aufteilung der Sorte `Information` basiert auf den Objekten, die die Informationen betreffen. Daraus ergibt sich zum einen die Subsorte `ContactableInformation`, die Informationen über Personen oder Firmen, die in der Domäne unter einer Sorte `Contactable` vereint sind, betrifft. Hierzu gehören Sorten, die EMail-Adressen sowie verschiedene Telefonnummern zur Kontaktaufnahme repräsentieren. Ihnen stehen „allgemeine“ Sorten gegenüber, die als Subsorten von `RegularInformation` modelliert sind, wie Uhrzeiten, Orte oder Projektpläne, aber auch Medien wie Bilder, Tonaufnahmen und ähnliches.
Eine andere Unterteilungsmöglichkeit ergibt sich daraus, mit Hilfe welcher Nachrichten die Informationen übertragen werden können. Diese Nachrichten werden in der Domäne durch die Sorte `Message` mit den Subsorten `EMail`, `SMS` und `Call` (ein Anruf wird hier also nicht anders betrachtet als unidirektionale Nachrichten) repräsentiert, die jeweils zur Vermittlung unterschiedlicher Informationen dienen können, wobei eine `EMail` prinzipiell zur Übermittlung jeder Art von Information verwendet werden kann.

Einen zweiten grundlegenden Aspekt der Domäne stellen die verschiedenen Modi dar, in denen sich das Smartphone befinden kann, und die als Ausgangspunkte für verschiedene Aktivitäten dienen. In welchem Modus das Gerät sich befindet, wird mit Relationen modelliert, die ein Präfix „`inMode_`“ tragen, beispielsweise `inMode_Home`, womit anzeigt wird, dass sich das Gerät im Standardmodus befindet, von dem aus in die verschiedenen anderen Modi gewechselt werden kann. Dieser Wechsel erfolgt durch Tasks, die sich ein Präfix „`enterMode_`“ teilen. Die verfügbaren Modi reichen dabei vom bereits genannten Standardmodus über einen durch `inMode_Telephone` spezifizierten Modus zur direkten Eingabe einer Telefonnummer bis hin zu Modi zum Anlegen von Terminen und Erinnerungsereignissen, bei deren Aktivität `inMode_Calendar` bzw. `inMode_Alarm` gesetzt wird.

Verwendung von Informationen

Informationen können in der Smartphone-Domäne verwendet werden, um verschiedene organisatorische Tätigkeiten durchzuführen. Dazu gehört zum Beispiel das Anlegen von Kontakten im Adressbuch des Geräts, wozu ein abstrakter Task `create_Contact` modelliert wurde. Dessen einzige implementierende Methode `do_create_Contact` erfordert zunächst einen Wechsel in das Adressbuch mittels `enterMode_Contacts`, wo anschließend mit einem weiteren abstrakten Task `create_newContact` der Kontakt angelegt wird. Dessen Implementierung erfordert unter anderem einen Druck auf die Schaltfläche zum Anlegen eines Kontakts,

was das niedrigste von den Tasks der Domäne repräsentierte Abstraktionsniveau darstellt, und durch `press_Contacts.New` modelliert ist. Der nächste Schritt ist dann die Konfiguration des neuen Kontakts, wofür eine Telefonnummer oder EMail-Adresse sowie ein Name benötigt wird. Optional kann dem Kontakt zusätzlich ein Bild zugeordnet werden. Der ganze Vorgang wird durch einen Druck auf die „OK"-Taste abgeschlossen, verkörpert durch den Task `press_Contacts.NewContact.OK`. Zur Speicherung eines Kontakts wird dabei noch ein `Contact` benötigt, der als Subsorte von `Creatable` einen entsprechenden Speicherplatz im Smartphone darstellt, und der darüber hinaus nicht belegt sein darf. Diese Belegung von Speicherplätzen wird durch eine Relation `inUse` modelliert.

Das Anlegen weiterer `Creatables` wie Terminen, Aufgaben oder Erinnerungen folgt ebenfalls dem Schema, dass ein Wechsel in den entsprechenden Modus, die Konfiguration des Objekts mit einem Minimum an Information und schließlich seine Speicherung erfolgen müssen. Eine erfolgreiche Konfiguration eines `Creatable` wird in der Domäne mit einer Relation `isConfigured` repräsentiert, wobei Dekompositionsaxiome zur Übertragung auf die Ebene der verschiedenen konkret anlegbaren Objekte existieren.

Übermittlung von Information

Neben den beschriebenen Mitteln zur Verwendung von Informationen durch den Smartphone-Benutzer selbst wurde in der Domäne auch der Versand von Informationen an andere Personen modelliert. Auf höchster Abstraktionsebene wird dieser Versand im Modell durch einen Task `transferMessage` realisiert. Dessen Implementierung besteht dann aus drei Schritten: Zunächst werden die zu vermittelnden Informationen an eine Nachricht angehängt, wobei diese Informationen natürlich auf Senderseite vorhanden sein müssen, was mit Hilfe der Relation `haveInformation` ausgedrückt wird. Zum Anhängen von Informationen an eine Nachricht dient ein Task `attachMultipleInformation`, der wie der Name schon sagt auch zum Anhängen von mehreren Informationsobjekten verwendet werden kann. Entsprechend gibt es eine Implementierung `do_attachMultipleInformation_Single`, mit der eine einzelne, und `do_attachMultipleInformation_Multiple`, mit der mehrere Informationen mit einer Nachricht verknüpft werden können; Die letztgenannte Methode setzt dies so um, dass in ihrem Tasknetz wieder der Task `attachMultipleInformation` enthalten ist. Um anzugeben, dass eine Information an eine Nachricht angehängt ist, dient die Relation `attached`.

Der zweite Schritt der Informationsübermittlung ist die Übermittlung an sich, zu der ein Kontakt zum erwünschten Empfänger hergestellt werden muss. Dies wird durch einen Task `contact` modelliert, der durch Tätigen eines Anrufs mittels eines Tasks `call`, oder das Versenden einer EMail (`send_EMail`) oder einer SMS (`send_SMS`) implementiert werden kann, wobei wie oben beschrieben nicht jede

Nachricht zur Übertragung jeder Information geeignet ist. Zudem muss eine entsprechende Kontaktmöglichkeit gegeben sein, das heißt um zum Beispiel jemandem eine Information per EMail zukommen zu lassen, muss für diesen Kontakt die Relation `mailable` gelten. Für die verschiedenen, das Vorhandensein einer Kontaktmöglichkeit repräsentierenden Relationen, gibt es jeweils mehrere Möglichkeiten, die durch entsprechende Dekompositionsaxiome zusammengefasst sind. `mailable` lässt sich beispielsweise daraus ableiten, dass eine EMailadresse des gewünschten Empfängers bekannt ist, was mit einer Relation `haveInformation_Contactable` formuliert wird, oder aber dass im Adressbuch ein Kontakt für den Empfänger existiert, dem eine EMailadresse zugeordnet ist.

Ist für einen Empfänger nun beispielsweise eine EMailadresse gegeben, kann eine EMail auf mehrere Arten auf den Weg gebracht werden. Eine Möglichkeit stellt der Wechsel in den EMail-Modus dar, wo eine Nachricht verfasst und mit der Adresse des Empfängers versehen werden kann. Für Personen, die häufiger zu kontaktieren sind, bietet es sich alternativ an, einen Eintrag im Favoriten-Menü zu erzeugen, von dem aus ein direkter Zugriff auf die verfügbaren Kontaktmöglichkeiten möglich ist; Von dort aus sind also auch Anrufe und der Versand von SMS möglich, vorausgesetzt die jeweilige Kontaktinformation ist gegeben.

Die Entgegennahme von Information aus einer Nachricht auf der Empfängerseite schließlich wird durch einen Task `extractMultipleInformation` modelliert, dessen Implementierungen genau wie bei seinem Gegenstück `attachMultipleInformation` aufgebaut sind. Ist diese Informationsextraktion abgeschlossen, ist der Empfänger im Besitz der Informationen, was durch die Relation `hasInformation` angezeigt wird.

Problemstellungen in der Domäne

Mit den in der Domäne modellierten Konzepten sind eine Anzahl von Planungsproblemen formulierbar, deren Komplexität sich gut skalieren lässt. Einfache Probleme können darin bestehen, beispielsweise einen Termin, einen Telefonbucheintrag oder Kontakt anzulegen, oder eine Nachricht mit einer Information zu versehen und zu versenden. Pläne die solche Probleme lösen können dann zum Beispiel im Sinn einer Anleitung für den Nutzer verwendet werden, der mit der Bedienung des Geräts nicht vertraut ist.
Größere Probleme können dann durch Kombination der Erstellung von Terminen mit dem Versand zugehöriger Informationen an die Teilnehmer erstellt werden, wobei die Komplexität durch eine Erhöhung der Teilnehmerzahl, den Versand unterschiedlicher Informationen an die unterschiedlichen Empfänger etc. beliebig erhöht werden kann, was sowohl für den Nutzer als auch für das Planungssystem zunehmend schwieriger zu lösen ist. Gerade mit der steigenden Komplexität wird aber auch ein Assistenzsystem immer wertvoller, welches den Nutzer an noch auszuführende Teilschritte erinnern kann.

4 Allgemeine Aspekte der Planerklärung

Wir wollen in diesem Kapitel einige grundlegende Aspekte der Planerklärung behandeln. In Abschnitt 4.1 wird untersucht, wie die Fragen, die ein Nutzer zu einem Plan stellen kann, entstehen, und welche diese sind. Abschnitt 4.2 skizziert den grundlegenden Aufbau einer Erklärung, mit der eine solche Frage beantwortet werden kann. Abschnitt 4.3 beschäftigt sich schließlich damit, welche Eigenschaften die Qualität einer Erklärung ausmachen, und führt zu einem Ansatz, wie diese Qualität sichergestellt werden kann.

4.1 Mögliche Fragestellungen

In diesem Abschnitt soll untersucht werden, mit welchen Fragen ein Erklärungssystem konfrontiert werden kann. Zu diesem Zweck sollen zuerst mögliche Gründe identifiziert werden, aus denen ein Nutzer Teile eines Plans nicht nachvollziehen kann, da davon auszugehen ist, dass genau diese Tatsache beim Nutzer Fragen aufwirft.

Eine naheliegende Ursache für die Unverständlichkeit von Plänen für einen menschlichen Rezipienten ist dessen Unkenntnis von Teilen oder sogar der gesamten Domäne, für die der Plan entwickelt wurde. Auch wenn die Domäne in Grundzügen bekannt ist werden fehlende Informationen über die Bedingungen zur Ausführbarkeit beliebiger oder die Umsetzung abstrakter Aktionen, eine gewisse Komplexität in Domäne und Problemstellung vorausgesetzt, zwangsweise Fragen aufwerfen. Man könnte an diesem Punkt noch weiter gehen und sogar in Betracht ziehen, dass der Nutzer die Problemstellung nicht kennt, für die der Plan entwickelt wurde. Eine entsprechende Frage ist jedoch kaum mit mehr zu beantworten als mit eben dieser Problemstellung, was weniger ein Problem der Erklärbarkeit als der Verbalisierung ist, und damit nicht im Fokus dieser Arbeit liegt.

Eine weitere wichtige Quelle für Planmerkmale, die einem Nutzer nicht nachvollziehbar erscheinen können, sind die Diskrepanzen zwischen den Vorgehensweisen, die ein Mensch und ein Computersystem beim Planen an den Tag legen. Eine zentrale Eigenschaft der menschlichen Vorgehensweise dabei ist es, dass nicht alle Planschritte auf die Ebene primitiver Aktionen heruntergebrochen werden müssen. Stattdessen kann ein Plan unter Umständen auch als fertig betrachtet werden, wenn er noch

Schritte beinhaltet, die eine Abstraktion mehrerer in Wirklichkeit durchzuführender Schritte darstellen. Dies kann für den Menschen aus zweierlei Gründen bereits zufriedenstellend sein.

Erstens plant ein Mensch für gewöhnlich in Domänen, die starker Unsicherheit unterliegen. Hier können unvorhersehbare äußere Einflüsse dafür sorgen, dass Voraussetzungen, von denen zur Planungszeit ausgegangen wurde, zur Ausführungszeit nicht mehr erfüllt sind. Das Wissen um diese Unsicherheit macht die Erzeugung eines vollständig expliziten Plans wenig erstrebenswert, da je nach Problemstellung nur geringe Aussichten darauf bestehen, diesen später tatsächlich umsetzen zu können. Gerade bei der Benutzung von Mobiltelefonen ergibt sich ein typisches Beispiel, wenn ein Meeting organisiert und dafür die gewünschten Teilnehmer eingeladen werden sollen. Befindet sich einer dieser Teilnehmer zur Zeit eines Anrufversuchs in einem Funkloch, so schlägt ein maschinell generierter Plan in diesem Moment fehl, während der Mensch auf die veränderte Situation reagieren und die Person stattdessen mit einer SMS über den anstehenden Termin informieren kann. Ansätze, diese Möglichkeiten auch maschinellen Planungssystemen zur Verfügung zu stellen, realisieren eine solche Anpassung an veränderte Bedingungen beispielsweise mit Hilfe von *Planreparatur*, wofür unter anderem auch für `PANDA` ein Ansatz existiert (vgl. [BBS08]).

Der zweite Grund dafür, dass nicht vollständig konkretisierte Pläne für Menschen akzeptabel sein können, liegt in der Art, wie die Ausführbarkeit von Plänen nachgewiesen wird. Wie eingangs in Kapitel 2.1 beschrieben ist das Resultat eines maschinellen Planungsvorgangs stets ein vollständig expliziter Plan, sei es, weil von Beginn an nur mit primitiven Tasks geplant wird, oder weil das Vorhandensein abstrakter Tasks als noch zu behandelndes Problem betrachtet wird, wie es beispielsweise auch bei `PANDA` der Fall ist. Durch diese vollständige Instanziierung aller variablen Bestandteile ist gerade auch die Planausführbarkeit bewiesen. Demgegenüber muss ein Mensch nicht alle primitiven Bestandteile von Plänen betrachten, denn er kann sich auf seine Erfahrung und Intuition bezüglich der Ausführbarkeit abstrakter Aktionen verlassen. Er verwendet sein Wissen über deren verschiedene Implementierungen, um sogenannte *Landmarken* zu identifizieren, also die kritischen Punkte, von denen die Ausführbarkeit eines Plans oder Planfragments abhängt. Diese Charakteristik der menschlichen Planungsweise ermöglicht es, schon durch eine oberflächliche Prüfung die Durchführbarkeit eines abstrakten Tasks durch Einsatz einer bestimmten Implementierung mit hoher Wahrscheinlichkeit korrekt festzustellen. In Abbildung 4.1 sind beispielsweise die drei in der Smartphone-Domäne bekannten Methoden angegeben, mit deren Hilfe der abstrakte Task `call (Contactable, Call)` zum Anrufen einer Person realisiert werden kann. Die Namensgebung der Methoden weist bereits auf die entscheidenden Eigenschaften der jeweiligen darunterliegenden Implementierung hin, so setzt `do_Call_Favourite` den Anruf beispielsweise durch die Verwendung des Favoriten-Menüs um. Der kritische Punkt des skizzierten Tasknetzes dieser Methode kann leicht in der Vorbedingung `isFavourite(Contact)` des Planschritts `select_Favourite(Contact)` erkannt werden, da dieser die

Besonderheit dieser Implementierung gegenüber den anderen Möglichkeiten ausmacht, und die anderen Bestandteile des Planfragments eher triviale Aufgaben sind.

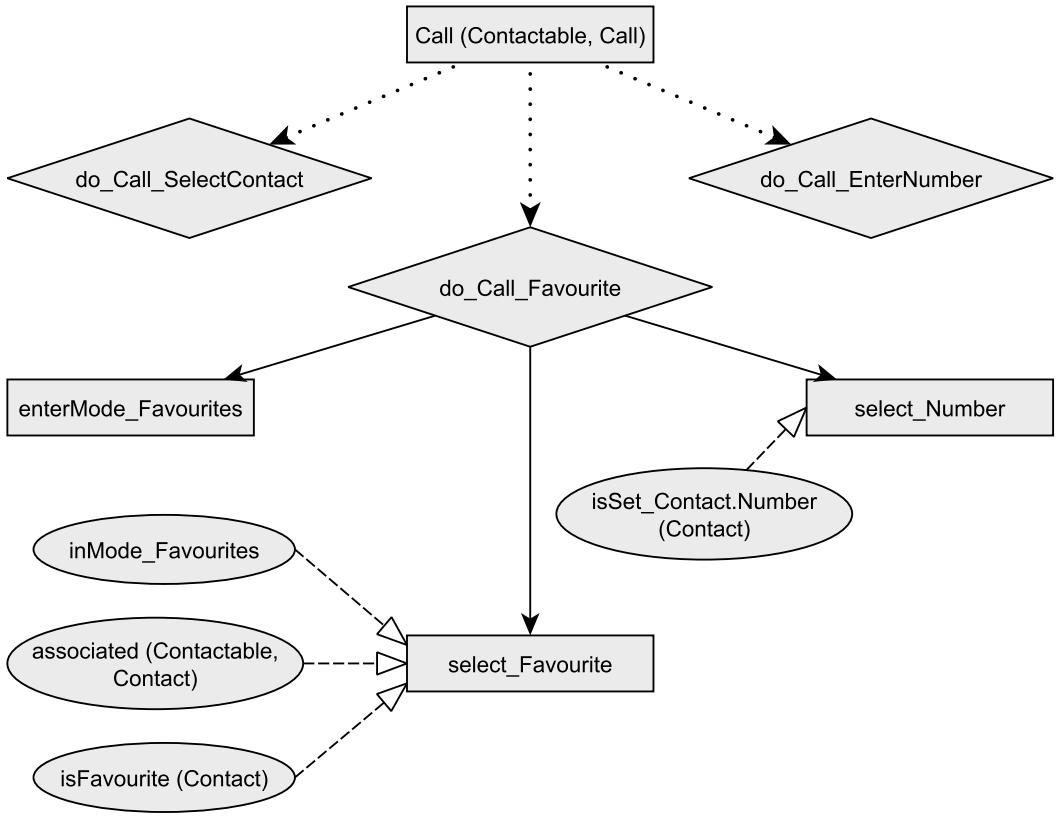


Abbildung 4.1: Eine Implementierung des Tasks Call(Contactable, Call) durch die Methode do_Call_Favourite und das zugehörige Dekompositionsaxiom. Ellipsen stellen Vor- bzw. Nachbedingungen von Tasks dar

Eine weitere Folge der maschinellen Planerzeugung ist, dass die Anordnung der Elemente des resultierenden Plans nicht derjenigen entsprechen muss, die sich bei einem von Menschen entwickelten Plan ergeben würde. Bedingt wird dies einerseits durch die Modellierung von Domäne und Problem, die den Schritten eine gewisse Ordnung aufzwingen kann, vor allem aber durch die eingesetzten Planungsstrategien, die den Prozess der Planerzeugung steuern und insbesondere die Reihenfolge beeinflussen, in der der Plan bis hin zu seiner Ausführbarkeit modifiziert wird. Diese Strategien werden bei maschinellen Planungssystemen auf Grund ihrer Laufzeiteffizienz und systemspezifischen Eigenheiten ausgewählt, nicht weil ihre Pläne den von Menschen erzeugten ähneln. Dagegen wird die Reihenfolge der Planschritte in einem von einem Mensch erarbeiteten Plan beispielsweise durch die Bedeutung, die er diesen zumisst, beeinflusst. So ordnet ein Mensch die Erledigung wichtiger Schritte eher am Anfang

eines Plans ein, was auch im Hinblick auf eine eventuell notwendige Planreparatur Sinn macht, da dann zur Ausführung der essenziellen Teile des Plans noch mehr Zeit zur Verfügung steht.

Dass ein Mensch stets den Faktor Zeit, oder allgemein Ressourcen, einbezieht, ist auch eine weitere Ursache dafür, dass maschinell erstellte Pläne für ihn unerwartete Formen annehmen können. Insbesondere Planungssysteme, die kein *Scheduling*, also keine Zeit- und allgemeine Ressourcenverwaltung betreiben, sondern nur an einer gültigen Lösung interessiert sind, können Pläne mit Schritten hervorbringen, die durch eine geschicktere Anordnung vermeidbar wären. So ist ein Plan, in dem für jeden Anruf die Nummer von Hand eingegeben wird, obwohl der Kontakt auch im Telefonbuch verzeichnet ist, für die obige Problemstellung der Organisation eines Meetings technisch zwar korrekt – effizient, und daher für einen menschlichen Betrachter des Plans naheliegend, ist er jedoch nicht. An dieser Stelle sei angemerkt, dass die zum Zeitpunkt des Verfassens dieser Arbeit aktuelle Implementierung von `PANDA` noch kein Scheduling unterstützt, eine Integration wird jedoch in [SB07] bereits theoretisch beschrieben.

Schließlich gibt es noch eine dritte Ursache dafür, dass maschinell erstellte Pläne für Menschen nicht nachvollziehbar sein könnem, nämlich dass diese Pläne ein Produkt einer zu komplexen Problemstellung und damit zwangsweise zu kompliziert sind, um ihren Aufbau ohne Kenntnis aller Einflussfaktoren zu verstehen. Die Entwicklung von Plänen für solche schwierigen Problemstellungen ist zwar gerade einer der Gründe dafür, maschinelles Planen zu betreiben, stellt aber beispielsweise bei der Überprüfung der Korrektheit eines erstellten Plans durch einen Menschen ein Problem dar. Besonders hervorzuheben sind hierbei Einschränkungen auf Parameterbelegungen, die sich im Planungsprozess ergeben, deren Menge ein für den menschlichen Betrachter schwer zu durchschauendes Ausmaß entwickeln kann. Diese Einschränkungen sind darüber hinaus in der Präsentation eines Plans normalerweise nicht enthalten, einerseits eben auf Grund ihrer Anzahl, andererseits weil zur Ausführung des Plans nur die letztlich resultierenden Parameterbelegungen von Bedeutung sind. Dadurch sind für einen Betrachter des fertigen Plans keine Informationen verfügbar, die beim Nachvollziehen des Zustandekommens der Parameterbelegungen hilfreich wären.

Aus den hier herausgearbeiteten Unterschieden zwischen der menschlichen und maschinellen Planungsweise sowie der Problematik der potentiell hohen Komplexität von Plänen lassen sich eine Vielzahl von Fragen ableiten, deren Erklärung das Verständnis eines Plan grundlegend verbessern kann. Wir beschränken uns in dieser Arbeit jedoch auf drei Fragestellungen, die sich aus der Betrachtung eines Plans ergeben, da sie sich auf bestimmte erkennbare Merkmale dieses Plan beziehen, und die keinerlei weiteres Wissen auf Nutzerseite voraussetzen:

- Warum ist ein bestimmter Planschritt Teil des Plans?

- Warum sind zwei Planschritte auf eine bestimmte Weise zueinander angeordnet?
- Warum besitzt ein Parameter den ihm zugeordneten Wert?

Entsprechende negative Varianten, wie „Warum ist ein bestimmter Task *nicht* Teil des Plans?“, aber auch die Beantwortung der Frage „Wie wurde ein abstrakter Task der Problemstellung implementiert?“ würden zu der Verständlichkeit des Plans ebenfalls beitragen, ihre Beantwortung erfordert aber eine gänzlich andere Herangehensweise. Zudem resultieren sie eben nicht direkt aus der Betrachtung des Plans, sondern es sind zusätzliche Informationen über die Domäne oder die Problemstellung nötig, da sie sich auf im fertigen Plan nicht enthaltene Elemente beziehen. Ein in der Problemstellung enthaltener abstrakter Task beispielsweise ist in einem solchen fertigen Plan gerade nicht mehr zu sehen, da er zur Entwicklung der Lösung ja konkretisiert werden musste.

4.2 Aufbau einer Planerklärung

Nachdem wir nun einige Fragen herausgearbeitet haben, die ein Nutzer zu einem Plan stellen kann, um diesen besser nachvollziehen zu können, müssen wir untersuchen, wie eine Antwort auf diese Fragen aussehen kann. Angenommen es liegt der in Abbildung 4.2 (S. 43) gezeigte Plan vor, der die zum Tätigen eines Anrufs nötigen Schritte darlegt, und unter anderem den Task `press_home.People` zum Öffnen der Kontaktliste beinhaltet. Um eine Frage, warum dieser Task für den Plan notwendig ist, zu beantworten, lässt sich intuitiv als gute Erklärung bestimmen, seine Bedeutung für den nachfolgenden Task als Begründung anzuführen: „Man muss `press_home.People` ausführen, um die Bedingung `inMode_People` wahr zu machen, die zur Ausführung des Tasks `press_People.More` benötigt wird.“ Diese Erklärung führt jedoch offensichtlich zu der Nachfrage, warum `press_People.More` Teil des Plans ist – die Antwort sollte daher auch gleich dessen Vorhandensein mitbegründen, um solchen Nachfragen vorzugreifen: „`press_People.More` ist wiederum notwendig, um die Bedingung `inMode_Contacts` wahr zu machen, so dass der Task `select_Contacts.ContactForContactable` ausgeführt werden kann.“ Setzt man dies so fort, dann ergibt sich damit eine Kette von im Folgenden auch als *Argumente* bezeichneten einschrittigen, miteinander verknüpften Erklärungen.

Es stellt sich nun die Frage, wo diese Kette enden kann. Die Antwort hierauf ist abhängig von der Art der vom Nutzer gestellten Frage: Ist eine Tasknotwendigkeit zu erklären, muss sich ein solcher Punkt dadurch auszeichnen, dass er keine weitere Nachfrage durch den Nutzer motiviert, seine Notwendigkeit muss sozusagen selbsterklärend sein. Dies ist in der Aufgabenstellung gegeben, also in dem Task, der den initialen Weltzustand herstellt, den Teilen des initialen Tasknetzes und dem

Zielzustands-Task. Handelt es sich um die Erklärung der Gleichheit eines Parameters und eines Terms, so stellen die Planschritte, zu denen der Parameter und der Term gehören, die Start- und Endpunkte der Erklärung dar, da sie zueinander in Beziehung zu setzen sind. Das selbe gilt für eine Frage nach einer Anordnung zweier Tasks, da auch hier die Erklärung das Zustandekommen einer Beziehung zwischen den beiden erläutern muss.

Die für so eine Erklärung verwendeten Argumente lassen sich grob in eine von zwei Kategorien einordnen. Auf der einen Seite kann man das Vorhandensein eines Planmerkmals *konstruktiv* begründen, also anhand des Planungsvorgangs, der an irgendeiner Stelle zu einer Modifikation geführt hat, in deren Folge dieses Merkmal zustande gekommen ist. Für eine Planerklärung spielt also nicht nur der fertige Plan eine Rolle, sondern auch der Planungsverlauf, an dessen Ende dieser Plan steht. Diesen konstruktiven stehen die *funktionalen* Argumente gegenüber; Diese beziehen sich nicht auf das Zustandekommen des Plans, sondern auf die Funktion, die ein Bestandteil im finalen Plan erfüllt, und ergeben sich beispielsweise aus kausalen Links, wie im eingangs genannten Beispiel.

Der allgemeine Aufbau aller Arten von Planerklärungen wird in Abbildung 4.3 (S. 44) noch einmal so veranschaulicht, wie wir es auch für weitere Darstellungen von Erklärungen in dieser Arbeit beibehalten werden. Planschritte, hier A, B, C und D, werden durch funktionale (zwischen A und B sowie zwischen C und D) und konstruktive (zwischen B und C) Argumente miteinander in Verbindung gebracht, die auf Beziehungen im Plan, hier r, s und t, basieren.

Mit der Verfügbarkeit verschiedener Argumente hätte eine Erklärung der Notwendigkeit des Tasks `press_home.People` im oben verwendeten Beispiel also auch mit einem konstruktiven Argument beginnen können: „Der Task `press_home.People` ist notwendig für den Plan, weil er Teil der Implementierung von `enterMode_People` ist.“ Diese Erklärung würde aus anderen Argumenten aufgebaut werden, die auf anderen Eigenschaften des Plans basieren, als die zuerst angeführte. Tatsächlich lassen sich in den meisten Fällen eine große Anzahl unterschiedlicher Erklärungen für eine Frage des Nutzers finden. Diese können sich in Aufbau und Qualität deutlich unterscheiden, so dass die Notwendigkeit besteht, die möglichen Erklärungen dahingehend gegeneinander abzuwägen, welche die erfragte Eigenschaft des Plans am besten erklärt.

4.3 Die Suche nach einer guten Erklärung

Nachdem wir in den vorangegangenen Abschnitten mögliche Fragen und den Aufbau von Erklärungen von nachgefragten Planeigenschaften beschrieben haben, soll in diesem Kapitel untersucht werden, wie eine gute Erklärung zu finden ist. Dazu betrachten wir im nächsten Abschnitt, welche Eigenschaften eine Erklärung ausmachen, und entwickeln darauf aufbauend einen allgemeinen Ansatz zur Bewertung

ihrer Qualität. Abschnitt 4.3.2 setzt sich anschließend damit auseinander, wie diese Eigenschaften in einer guten Erklärung ausgeprägt sind. Auf der Grundlage der gewonnenen Erkenntnisse wird schließlich in 4.3.3 der in dieser Arbeit verfolgte Ansatz einer nutzerorientierten Planerklärung motiviert.

4.3.1 Bewertung von Erklärungen

Eigenschaften von Erklärungen

Wir wollen in diesem Abschnitt untersuchen, welche Merkmale einer Planerklärung ausschlaggebend für ihre Akzeptanz und Verständlichkeit sind. Die wichtigste Anforderung an eine Erklärung ist offensichtlich die, dass sie den erfragten Sachverhalt tatsächlich erklären, das heißt in ihrer Argumentation schlüssig sein muss. Dies kann erreicht werden indem eine geschlossene Argumentationskette verwendet wird, für deren Glieder jeweils eine Begründung vorliegt, wie es auch im Abschnitt 4.2 erläutert wurde. Wir setzen diese Eigenschaft für die folgenden Betrachtungen voraus.

Um interessantere Eigenschaften zu identifizieren können wir uns am oben beschriebenen Aufbau einer Planerklärung orientieren. Wir unterscheiden hier zwischen formalen und inhaltlichen Eigenschaften. Formale Eigenschaften finden wir auf zwei Ebenen; Erstens auf der Ebene der Erklärung insgesamt: diese besitzt eine gewisse Anzahl von Argumenten, und man kann die Art ihrer Argumente zueinander in Kontext setzen, beispielsweise welche Typen von Argumenten aufeinander folgen. Zweitens auf der Ebene der einzelnen Argumente; Hier finden sich ebenfalls formale Eigenschaften, nämlich zum einen ob sie auf einem funktionalen oder konstruktiven Zusammenhang basieren, zum anderen die genaue Art der Argumente an sich, das heißt welche Folgerung aus den dem Argument zu Grunde liegenden Planeigenschaften gezogen wurde. Vor allem werden in den Argumenten jedoch die Inhalte sichtbar, welche eine Erklärung ausmachen. Direkt erkennbar sind die Domänenelemente, auf die sich das Argument bezieht, nämlich die Planschritte, die in Verbindung gebracht werden, sowie die Verbindung, die zwischen ihnen besteht.

Eine allgemeine Bewertungsfunktion

Wie gezeigt wurde lassen sich die Eigenschaften auf unterschiedlichen Ebenen der Planerklärung finden. Es liegt daher nahe, sich auch bei der Bewertung einer Erklärung auf diese Ebenen zu stützen. Die Qualität einer Erklärung ergibt sich also zum einen aus ihren Eigenschaften auf der Erklärungsebene, zum anderen aus der Qualität ihrer einzelnen Argumente. Für sich genommen können wir mit dieser Herangehensweise allerdings noch wenig anfangen – was wir noch benötigen, sind Maßstäbe um die Bedeutung der inhaltlichen sowie formalen Eigenschaften jeweils untereinander und auch insgesamt in Relation setzen, und ihnen einen Wert

zuweisen zu können. Beispielsweise muss es möglich sein, abzuwägen, ob ein weiteres Argument, welches der Erklärung ja zusätzliche Information über den Plan hinzufügt, die „Kosten“ rechtfertigt, den Umfang der Erklärung dabei zu vergrößern.

4.3.2 Maßstäbe für die Bewertung

Nachdem wir die relevanten Eigenschaften einer Erklärung bestimmt und eine Bewertungsmethode entwickelt haben, soll in diesem Abschnitt untersucht werden, wie diese Eigenschaften ausgeprägt sein müssen, so dass eine gute Erklärung entsteht, um einen Maßstab für eine Bewertung von Erklärungen zu erhalten. Wir wollen uns dabei an einem Grundsatz der Informationsübermittlung orientieren, nämlich dass eine Erklärung „kurz und prägnant“ sein sollte. Diese zwei Anforderungen richten sich, formell ausgedrückt, an den inhaltlichen sowie den formalen Aufbau einer Erklärung, und bilden die Grundlage für die in dieser Arbeit angestellten Überlegungen.

Für die angestrebte *Kürze* der Erklärung kann kein absoluter Wert als Ziel gesetzt werden – ein Plan von vergleichsweise hoher Komplexität bringt wahrscheinlich zwangsweise mit sich, dass auch ihn betreffende Fragen einer eher komplexen, also aus einer größeren Zahl von Argumenten bestehenden Erklärung bedürfen. Dennoch sollte die Erklärung möglichst kompakt gehalten werden, denn ihre Verständlichkeit für den Nutzer steht in direktem Zusammenhang zu ihrem Umfang. So ist klar dass eine Erklärung, die sich auf wenige wichtige Punkte stützt, einfacher nachzuvollziehen ist, als eine mit einer langen Kette von Argumenten, die im Detail die Zusammenhänge im Plan erläutert. An dieser Gegenüberstellung von pointierten und eher ausschweifenden Erklärungen lässt sich auch die geforderte Eigenschaft der *Prägnanz* festmachen. Einerseits sollen die verwendeten Argumente alle Informationen tragen, die der Empfänger der Erklärung zu ihrem Verständnis und somit auch des nachgefragten Aspekts benötigt. Andererseits ist es zu vermeiden, wenig aussagekräftige Argumente zu verwenden, die die Erklärung verkomplizieren ohne dabei ihrem Empfänger viel Informationen zu vermitteln.

Einen allgemeinen Anhaltspunkt für diesen Informationsgehalt stellt die Abstraktionsebene dar, auf der sich die referenzierten Domänenelemente befinden. So besitzen abstrakte Tasks intrinsisch einen höheren Informationswert als ein Planschritt des sie implementierenden Tasknetzes, denn die Implementierung im Sinne des HTN-Planens bringt ja gerade mit sich, dass erst die Gesamtheit der Subtasks die gewünschten Effekte auf der darüberliegenden Ebene liefert. Mit diesem höheren Informationsgehalt geht einher, dass die Bezugnahme auf einen abstrakten Task statt einem Aufreihen seiner Subtasks auch der Kürze der Erklärung dient, denn dadurch kann unter Umständen die einzelne Referenzierung jedes Schrittes einer Implementierung vermieden werden. Abbildung 4.4 auf Seite 44 zeigt anhand eines Ausschnitts

einer Methodendefinition, wie die Bedeutung von Tasks auf unterschiedlichen Abstraktionsebenen gerade bei PANDA durch die Verwendung von Dekompositionsaxiomen betont wird. Die Tasks im Tasknetz der Methode do_configure_Contact liefern zusammen die beiden Effekte, die mit Hilfe des ebenfalls in der Abbildung dargestellten Axioms die Nachbedingung des implementierten abstrakten Tasks realisieren. Die Argumentation des höheren Informationsgehalts abstrakter Tasks lässt sich also auch auf Relationen übertragen. Andererseits können für die Erklärung eines Plans Aspekte von Bedeutung sein, die nur auf einer niedrigeren Abstraktionsebene zum Vorschein kommen. In diesem Fall besitzt eine Erklärung auf einem höheren Abstraktionsniveau zur Begründung einer nachgefragten Planeigenschaft nicht den Wert einer Variante auf primitiverer Ebene, deren Argumente sich explizit auf die relevanten primitiven Planbestandteile stützen.

Wir haben nun Anhaltspunkte für den Umfang und den Informationsgehalt einer Erklärung, mussten jedoch gleichzeitig feststellen, dass beispielsweise die in der Erklärung vorkommenden Planschritte und Relationen in Abhängigkeit von der Frage besser oder schlechter geeignet sind, diese zu beantworten. Weitere allgemeine Kriterien für „gute“ Domänenelemente konnten nicht gefunden werden. Daher sind für die Bewertung der Domänenelemente tiefergehende Ansätze zu ermitteln, die diese Elemente individuell betrachten.
Ein Versuch, diese Bewertung mit Hilfe uninformierter Algorithmen vorzunehmen, bietet hierfür jedoch wenig Erfolgsaussicht. Information über die Wichtigkeit der Sorten und Relationen einer Domäne kann zwar, beispielsweise durch eine Analyse der Häufigkeit ihrer Referenzierung, extrahiert werden, und auch die Anzahl von Tasks, die eine bestimmte Bedingung wahr machen, kann über deren Relevanz Aufschluss geben; Jede solche Analyse erlaubt aber primär Rückschlüsse auf die Wichtigkeit der jeweiligen Elemente für die Domäne, weniger für den Empfänger einer Erklärung.

Die Situation bei der Betrachtung der verschiedenen Argumenttypen gestaltet sich ähnlich schwierig, wie allgemeine Kriterien für die Nützlichkeit von Domänenelementen in der Erklärung zu bestimmen. Es besteht praktisch keine Möglichkeit, den Wert der verschiedenen Arten von Argumenten, aus denen eine Erklärung aufgebaut werden soll, objektiv zu ermitteln. Wir haben bereits beschrieben dass diese sich grundsätzlich unterscheiden, da funktionale Argumente hauptsächlich Aufschluss darüber geben, welche Funktionen ein Planschritt im finalen Plan ausübt, konstruktive Argumente hingegen auf der Entstehung des Plans aufbauen. Entsprechend sind diese Argumenttypen unterschiedlich gut für eine Erklärung geeignet, je nachdem, welche Art von Information vom Nutzer gesucht wird. Daher ist auch eine allgemeine Bewertung der verschiedenen Argumente nicht möglich.

Die Bedeutung der Eigenschaften

Wir haben im vorangegangenen Abschnitt festgestellt, dass die Maßstäbe zur Bewertung von Planerklärungen unter unterschiedlichen Umständen variieren. Diese Eigenschaften besitzen jedoch noch eine weitere wichtige Dimension, nämlich wieviel Bedeutung ihnen zugemessen wird. Dabei können in unterschiedlichen Anwendungsfällen unterschiedliche Prioritäten auftreten; Beispielsweise kann für ein Erklärungssystem, welches seine Erklärungen auf dem Display eines Mobiltelefons darstellt, die Anzahl der verwendeten Argumente von höchster Bedeutung sein, da der Darstellungsbereich stark begrenzt ist. Steht zu diesem Zweck hingegen ein normales PC-System zur Verfügung, spielt der Umfang möglicherweise eine geringere Rolle, ist aber wie erläutert wurde für das Verständnis der Erklärung immer noch ein tragender Faktor. Der entscheidende Unterschied ist jedoch, dass der Umfang der Erklärung im ersten Fall faktisch zu groß sein kann, so dass die Erklärung definitiv nicht ausgegeben werden kann. Im zweiten Fall hingegen ist eine kürzere Erklärung vielleicht im Allgemeinen besser, eine längere Erklärung auszugeben aber nicht grundsätzlich ausgeschlossen.

Anhand dieses Beispiels wird klar, dass bei der Bewertung von Erklärungen prinzipiell zwei Fälle zu unterscheiden sind. Einerseits kann eine Eigenschaft eine Ausprägung besitzen, die die Erklärung, unabhängig von ihren weiteren Merkmalen, unbrauchbar macht. Wir bezeichnen eine solche Ausprägung im Folgenden als *Defizit*. Andererseits können Eigenschaften eine Erklärung allgemein besser oder schlechter machen, ohne sie direkt von einer Verwendung auszuschließen; Solche Eigenschaften machen Erklärungen untereinander vergleichbar, und die auf diesem Weg bestimmte Qualität der Erklärung nennen wir ihren *Wert*. Ob eine bestimmte Ausprägung einer Eigenschaft in einer Anwendungssituation jedoch ein Defizit darstellt, oder nur den Wert der Erklärung beeinflusst, lässt sich, wie am Beispiel deutlich wurde, nicht allgemein festlegen.

4.3.3 Informierte Planerklärung

Mit den in den letzten Abschnitten gewonnenen Erkenntnissen, dass nicht allgemein definierbar ist, was eine gute Erklärung ausmacht, und dass selbst die Relevanz einzelner Eigenschaften situationsbedingt sehr unterschiedlich sein kann, scheint eine Bewertung einer Erklärung anhand generischer Maßstäbe nur schwer realisierbar.

Kommen wir, um einen anderen Ansatz zu motivieren, auf den Blickwinkel zurück, dass einem Nutzerassistenzsystem eine Erklärungskomponente zur Seite gestellt werden soll. Würde ein Mensch diese Aufgabe übernehmen, so müsste er zunächst ein Experte bezüglich der zu Grunde liegenden Domäne sein, um in der Lage zu sein, die Erklärungen so weit wie möglich auf für die gegebene Problemstellung wichtige

Aspekte der Domäne aufzubauen. Er könnte daneben auch anhand der Art der Frage die Absicht dahinter erkennen, was ihm ermöglicht, die richtigen Argumente für eine Erklärung zu benutzen. Ebenso wäre ihm von Beginn des Assistenzvorgangs an zumindest ungefähr bekannt, welches Wissen der Nutzer über die Domäne besitzt, so dass er vermeiden könnte, sich in seinen Erklärungen auf diesem unbekannte Elemente der Domäne zu stützen. Er könnte zudem seine Erklärung auch an die verwendete Modalität der Kommunikation anpassen, und ein Phänomen durch eine sprachliche Äußerung anders erklären als bei Zuhilfenahme einer Aufzeichnung der Zusammenhänge.
Insgesamt ergibt sich eine Vorgehensweise, bei der die Erklärung weniger an allgemeinen, und stattdessen stark an vom konkreten Anwendungsfall abhängigen Maßstäben ausgerichtet wird. Auf Grund des Mangels an allgemeingültigen Kriterien für gute Erklärungen erscheint dies als eine gute Methode der Erklärungsfindung und es ist anzunehmen, dass die Verständlichkeit einer Erklärung mit zunehmender Ähnlichkeit zu einer auf diese Weise von einem Experten erzeugten wächst. Es liegt also nahe, die selben Informationen auch bei der maschinellen Erzeugung von Erklärungen einzusetzen. Aus diesem Grund wird in dieser Arbeit der Ansatz einer *nutzerorientierten Planerklärung* verfolgt, die eine anwendungs- und nutzerspezifische Definition von Defiziten und Werten vorsieht.

Wir geben nun einen Überblick über die nächsten Kapitel, die den von uns gewählten Ansatz zur Entwicklung eines Frameworks zur nutzerorientierten Planerklärung beschreiben werden. Zunächst entwickeln wir in Kapitel 5 einen Prozess, der die Ausgabe einer solchen nutzerorientierten Erklärung ermöglicht. Dieser Prozess besteht aus drei Schritten: Erstens aus der Herstellung von Roherklärungen (Kapitel 5.1). Zweitens aus der Befreiung dieser Roherklärungen von ihren Defiziten, wofür diese Defizite in Klassen eingeteilt und Methoden zu deren Beseitigung beschrieben werden (Kapitel 5.2). Drittens der Auswahl einer dieser Erklärungen zur Präsentation auf Basis der in Abschnitt 4.3.1 beschriebenen Vorgehensweise zur Bewertung von Erklärungen. Die Definition von Defiziten und Werten, die zur Umsetzung dieser Schritte notwendig sind, wird durch die Integration externer Wissensquellen umgesetzt, die Informationen über die *Domäne* der Planerklärung, den *Nutzer* sowie die *Schnittstelle* zwischen Nutzer und Erklärungssystem zur Verfügung stellen. Repräsentation, Inhalt und die genaue Verwendung dieser Wissensquellen wird in Kapitel 6 beschrieben. Abbildung 4.5 auf Seite 45 gibt den Ablauf des Erklärungsprozesses, die Schnittstelle zwischen den Grundfunktionen des Frameworks und darauf aufbauenden Anwendungen sowie die beteiligten Wissensquellen noch einmal in einem grafischen Überblick wieder.

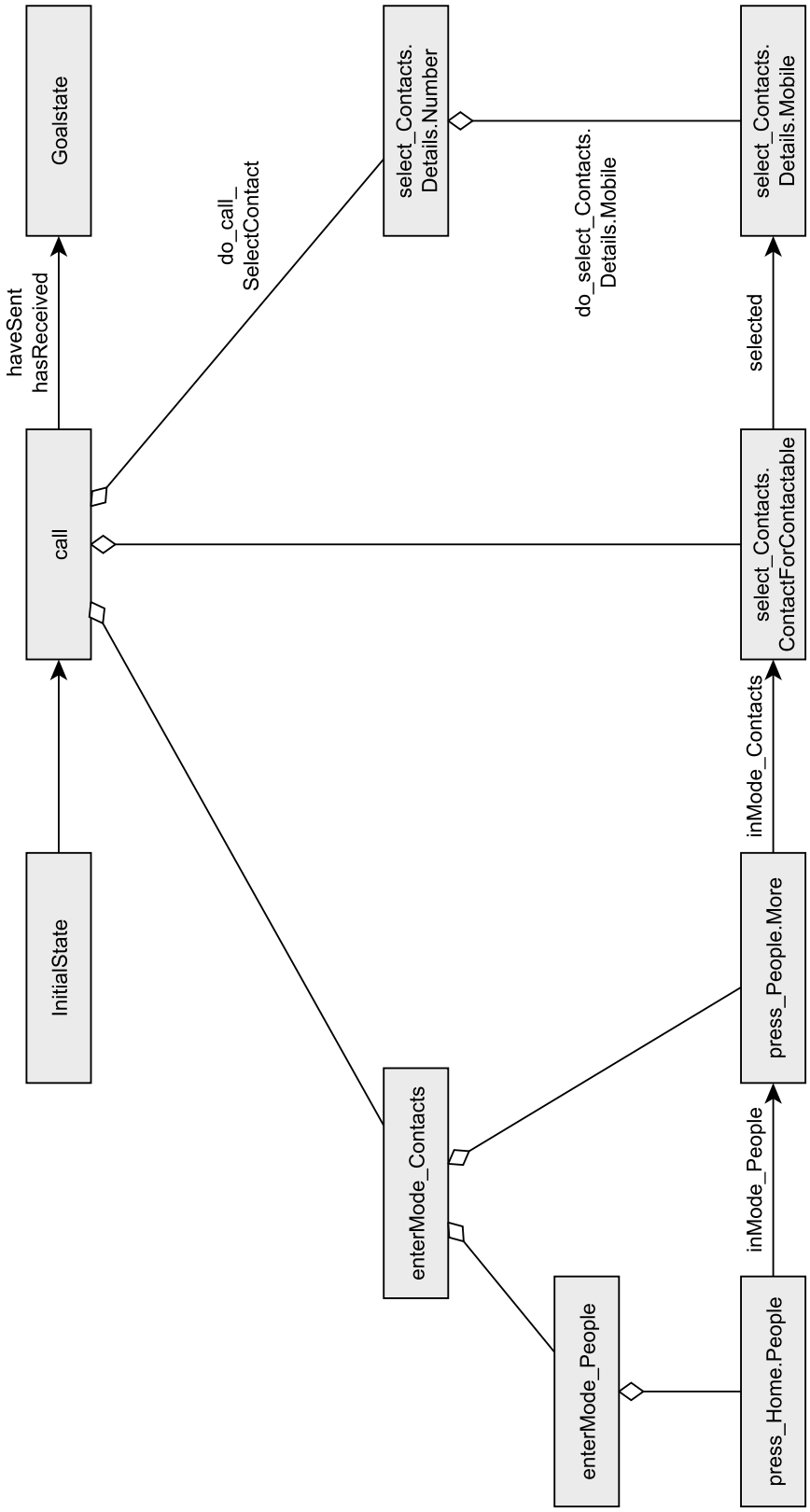


Abbildung 4.2: Ein Plan zur Tätigung eines Anrufs

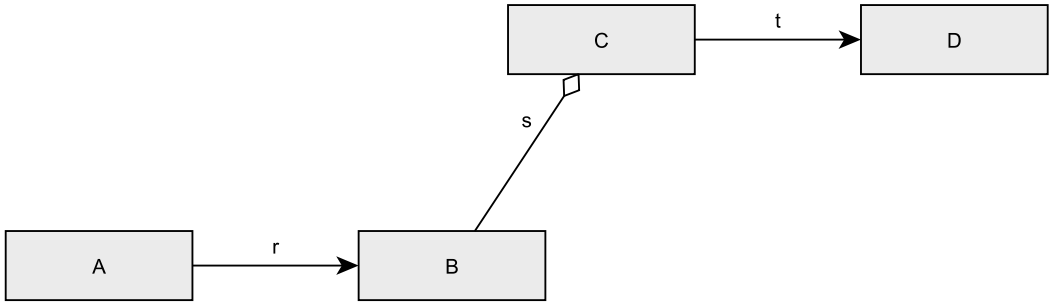
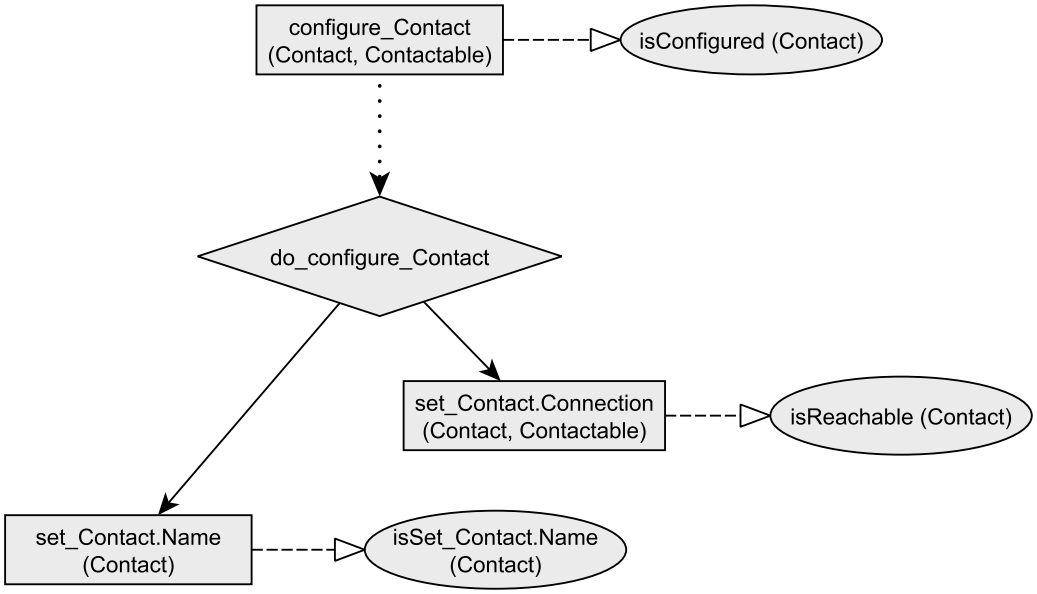


Abbildung 4.3: Der allgemeine Aufbau einer Planerklärung



$$isConfigured\_Contact\,(?Contact) \quad \Leftrightarrow \quad isReachable\,(?Contact) \wedge isSet\_Contact.Name\,(?Contact)$$

Abbildung 4.4: Eine Implementierung des Tasks configure_Contact(Contact, Contactable)

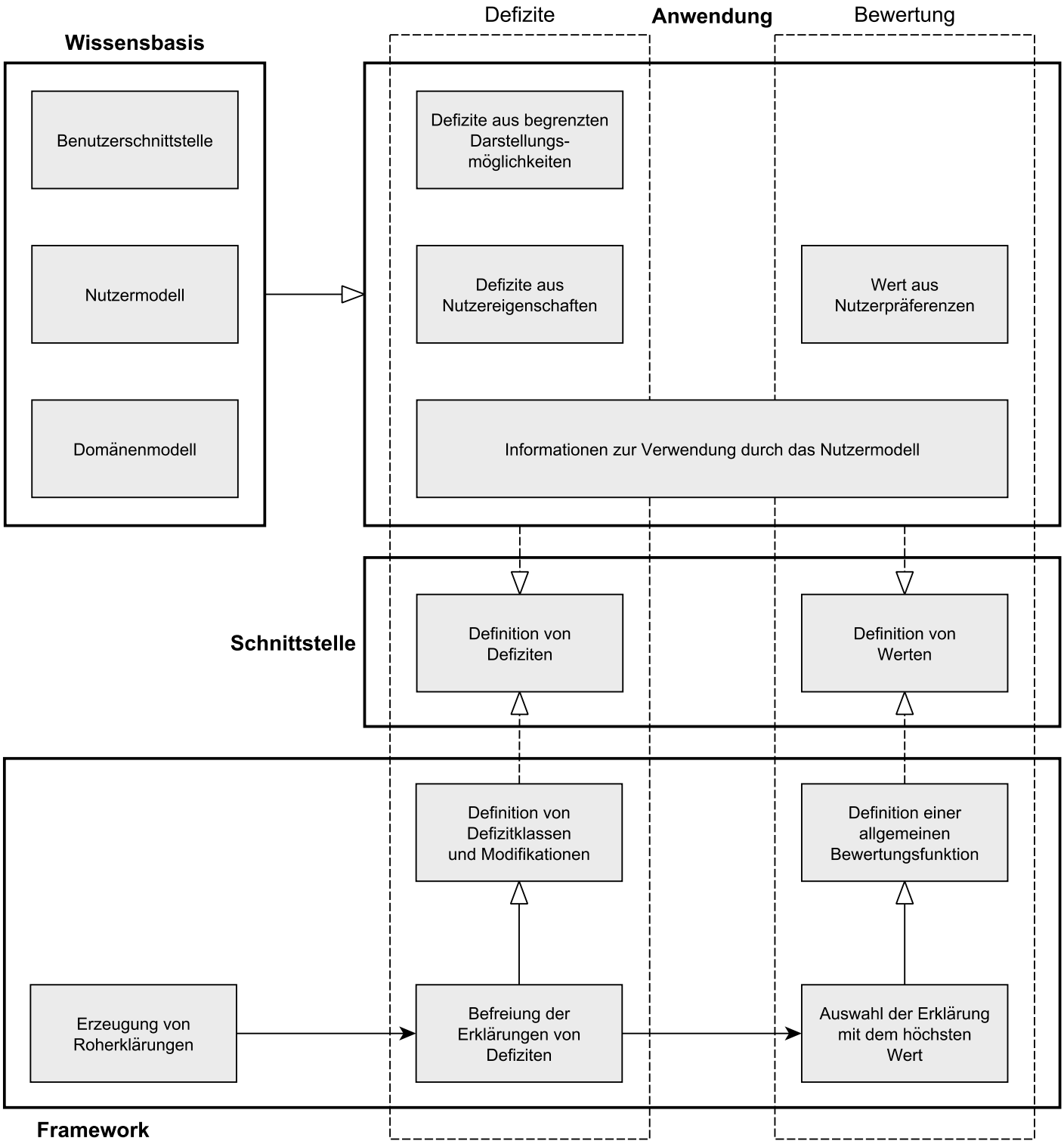


Abbildung 4.5: Eine Übersicht des Ansatzes zur nutzerorientierten Planerklärung

5 Ein Ansatz zur nutzerorientierten Planerklärung

In diesem Kapitel soll ein Ansatz vorgestellt werden, wie der Prozess der Erklärung eines Plans allgemein realisiert werden kann. Um einen möglichen Ablauf dafür zu finden betrachten wir zunächst die Ausgangssituation, in der ein Plan und eine Anfrage des Nutzers dazu vorliegt. Wir haben im vorangegangenen Kapitel bereits festgestellt, dass es meist eine große Anzahl von Möglichkeiten zur Erklärung dieser Anfrage gibt. Davon ausgehend sind zwei Vorgehensweisen denkbar.

Eine Option besteht darin, zu versuchen, eine Erklärung von Grund auf so aufzubauen, dass sie frei von Defiziten und möglichst wertvoll für den Nutzer ist. Dieser Ansatz hat aber mehrere Nachteile; Einerseits ist der Suchraum für Erklärungen stark verzweigt, weshalb man, um auf diesem Weg ein gutes Resultat zu erzielen, eine Strategie benötigt, die sehr fein in der Qualität der zur Auswahl stehenden Erklärungsschritte unterscheiden kann. Andererseits gibt es Qualitätsmerkmale, die auf der Erklärungsebene definiert sind, so dass zudem jede Erklärung komplett entwickelt werden müsste, um eine vollständige Bewertung durchführen und ihre Freiheit von Defiziten garantieren zu können.

Wir verfolgen dagegen einen Ansatz, bei dem zuerst eine große Zahl möglicher Erklärungen auf Grundlage der im Plan bestehenden Zusammenhänge erzeugt, und anschließend eine von diesen ausgewählt wird, die die gegebenen Anforderungen möglichst gut erfüllt. Dabei müssen wir allerdings auf den im letzten Kapitel behandelten Aspekt der Defizite von Erklärungen zurückkommen. Wir haben festgestellt, dass das Vorhandensein eines einzelnen Defizits eine Erklärung bereits unbrauchbar macht, unabhängig davon, wie nahe sie ansonsten den bestehenden Anforderungen kommt. Damit würden aber viele Erklärungen ausgeschlossen, die von einem solchen Defizit abgesehen gut geeignet wären – möglicherweise würde sich sogar herausstellen, dass keine der gefundenen Erklärung akzeptabel ist. Grundsätzlich wäre dies, solange nicht der zuletzt genannte Extremfall eintritt, kein Problem, es läuft aber unserem Ziel zuwider, eine möglichst gute Erklärung auszugeben. Daher ist es von Interesse, Möglichkeiten zu finden, Erklärungen von ihren Defiziten zu befreien, so dass das Repertoire an auswählbaren Erklärungen möglichst groß und letztlich eine möglichst zufriedenstellende Erklärung ausgegeben werden kann. Dabei muss allerdings gewährleistet sein, dass auch nach Anwendung solcher Modifikationen wieder

eine Erklärung vorliegt, die die Frage des Nutzers durch eine schlüssige Argumentation beantwortet.

Greifen wir, um zu herauszufinden, wie solche Modifikationen grundsätzlich vor sich gehen müssten, nochmals den Plan auf, anhand dessen der Aufbau von Erklärungen ermittelt wurde (Abb. 4.2 auf S. 43). Nehmen wir beispielsweise an, dass die dort begonnene Erklärung für die Notwendigkeit von `press_home.People` ein Defizit hat, weil der Planschritt `press_people.More` nicht verwendet werden kann. Würden wir nun diesen Planschritt einfach aus der Erklärung entfernen, wäre dieses Defizit zwar nicht mehr vorhanden; Die Argumentationskette der Erklärung wäre jedoch zerrissen, da der Planschritt ja der Begründung der Notwendigkeit von `press_Home.People` diente.

Planerklärungen besitzen jedoch eine Eigenschaft, die uns ermöglicht, die Argumentationskette wieder zu schließen: Die Argumente sind in einem gewissem Rahmen übertragbar. Die Erklärung einer Tasknotwendigkeit beispielsweise beruht ja gerade darauf, die nachgefragte Notwendigkeit bis auf die Problemstellung zurückzuführen – *letztlich* dient also auch in unserem Beispiel der Planschritt `press_people.More` dem Erreichen des Zielzustands. Man kann diese Eigenschaft nun aufgreifen, und die Notwendigkeit von `press_Home.People` so erklären: „`press_Home.People` ist notwendig, weil er den kausalen Link über `inMode_Contacts` für `select_Contacts.ContactForContactable` sichert.“ Dieser Zusammenhang ist im Plan zwar so nicht gegeben, aber *letztlich* sichert `press_Home.People` diesen kausalen Link, denn über einen Zwischenschritt besteht diese Verbindung ja durchaus. Wir können damit auch eine so modifizierte Erklärung als gültig einstufen, und besitzen nun einen Ansatzpunkt, wie Erklärungen von Defiziten zu befreien sind. Wir werden in Abschnitt 5.2 die hier als Beispiel verwendete Modifikation der Erklärung formalisieren, und auch für andere Defizite Möglichkeiten zeigen, diese zu beheben. Ehe wir dazu kommen, stellen wir im nächsten Abschnitt einen Formalismus vor, mit dem Roherklärungen für verschiedene Fragen zu Plänen gefunden werden können. Mit diesem Formalismus und der Möglichkeit, die mit ihm erzeugten Roherklärungen von Defiziten zu befreien, ergibt sich damit ein Ansatz zur Planerklärung, der in drei Schritten abläuft:

1. Erzeuge alle möglichen Roherklärungen
2. Bereinige alle Roherklärungen von ihren Defiziten
3. Bewerte die resultierenden defizitfreien Erklärungen und gib diejenige mit der höchsten Bewertung aus

5.1 Erzeugung von Roherklärungen

Wir wollen in diesem Abschnitt einen Überblick des in [See11] entwickelten Formalismus' geben, der sogenannte Roherklärungen für Eigenschaften eines vom `PANDA`-

Planungssystem generierten Plans liefert. Der dabei verfolgte Ansatz ist es, Eigenschaften wie beispielsweise die Notwendigkeit eines Tasks in diesem Plan formal zu beweisen. Dazu werden diese Eigenschaften, das Wissen über den Plan, also sein Aufbau, die in ihm bestehenden Zusammenhänge etc., sowie seine Entwicklungshistorie unter Verwendung einer formalen Semantik der Prädikatenlogik erster Stufe als Formeln repräsentiert. Nachgefragte Eigenschaften können anschließend mit Hilfe von zu diesem Zweck entwickelten Axiomen aus den im Plan bestehenden kausalen und konstruktiven Beziehungen abgeleitet werden. Da jede dazu herangezogene Eigenschaft des Plans selbst bewiesen sein muss, ergibt sich eine verkettete Struktur, wie sie in Kapitel 4.2 bereits propagiert wurde. Jeder Beweis wird unter diesem Blickwinkel zu einem Glied in der Argumentationskette der erzeugten Erklärung. Indem die Beweise der Planeigenschaften unter Verwendung der unterschiedlichen Ableitungen erbracht werden, ergeben sich eine Vielzahl von möglichen Roherklärungen für eine erfragte Eigenschaft.

Für das Erklärungssystem wurden bisher Axiome zur Beantwortung dreier Arten von Fragen entwickelt, die auch in dieser Arbeit behandelt werden: die Frage warum ein Task für den Plan notwendig ist, warum eine Variable und eine Term gleich sind, und warum eine bestimmte zeitliche Anordnung zweier Tasks vorliegt. Im nächsten Abschnitt wird zunächst gezeigt, wie funktionale und konstruktive Beziehungen im Plan abgeleitet werden können, die die Grundlage für alle Beweise bilden. Der darauf folgende Abschnitt erläutert dann, wie diese Beziehungen verwendet werden, um nachgefragte Planeigenschaften zu beweisen.

5.1.1 Axiome zum Beweis von Beziehungen im Plan

Der Formalismus beinhaltet eine Anzahl von Axiomen, mit denen funktionale sowie konstruktive Beziehungen in Plänen abgeleitet werden können, von denen in diesem Abschnitt einige vorgestellt werden sollen.
Zur Ableitung konstruktiver oder auch Dekompositionsbeziehungen gibt es nur ein Axiom, welches sich direkt auf die HTN-typische Zerlegung von Tasks im Planungsverlauf stützt. Eine Dekompositionsbeziehung wird also aus der Anwendung einer Methode m abgeleitet, wodurch ein Task t_1 durch die Dekomposition eines Tasks t_2 zum Plan hinzugefügt, und dabei t_2 aus dem Plan entfernt wird:

$$\forall t_1, t_2 : Task; m : Methode\,.\,[DekompositionsBeziehung(t_1, m, t_2) \Leftrightarrow$$
$$\exists d : Dekomposition\,.$$
$$[Hinzugefuegt(t_1, d) \wedge Entfernt(t_2, d) \wedge VerwendeteMethode(m, d)]]$$

Funktionale oder auch Kausalbeziehungen können im Gegensatz zu Dekompositionsbeziehungen auf unterschiedliche Weise aus einem Plan abgeleitet werden. Die naheliegendste Möglichkeit ist die Folgerung einer Kausalbeziehung aus einem kausalen Link zwischen den Schritten des Plans: Erfüllt ein Task t_1 eine Vorbedingung

für einen weiteren Task t_2, so trägt er zu dessen Ausführbarkeit bei, und erfüllt damit eine Funktion im Plan. Dies wird im Formalismus durch das folgende Axiom abgebildet:

$$\forall t_1, t_2 : Task; \phi : Formel \,.\, [KausalBeziehung(t_1, \phi, t_2) \Leftarrow$$
$$\exists cl : KausalerLink \,.$$
$$[t_1 = Produzent(cl) \wedge t_2 = Konsument(cl) \wedge \phi = Bedingung(cl)]]$$

Mit Hilfe dieses Axioms lassen sich Kausalbeziehungen zwischen Tasks ableiten, die sich auf der selben Abstraktionsebene befinden. Man könnte jedoch auch argumentieren, dass ein Planschritt, der eine Vorbedingung für einen folgenden Planschritt erfüllt, nicht nur zur dessen Ausführbarkeit, sondern auch zu der seines Supertasks beiträgt – die Kausalbeziehung wird also auf den übergeordneten Task propagiert. Die entsprechende Ableitung wird mit einem Axiom möglich, welches eine Dekompositions- und eine Kausalbeziehung kombiniert:

$$\forall t_1, t_2 : Task; \phi : Formel \,.\, [KausalBeziehung(t_1, \phi, t_2) \Leftarrow$$
$$\exists t_3 : Task \,.\, [KausalBeziehung(t_1, \phi, t_3)] \wedge$$
$$\exists m : Methode \,.\, [DekompositionsBeziehung(t_3, m, t_2)]]$$

Neben den zwei hier gezeigten Axiomen verwendet der vorgestellte Formalismus noch einige weitere zur Ableitung von Kausalbeziehungen, auf die jedoch nicht weiter eingegangen werden soll, da Aufbau und Verwendung dieser Axiome sich nicht grundsätzlich von den beschriebenen unterscheiden. Wir wenden uns stattdessen mit dem nächsten Abschnitt dem Beweisen von Planeigenschaften zu, wofür diese Kausal- und Dekompositionsbeziehungen die Grundlage bilden.

5.1.2 Axiome zum Beweis von erfragten Eigenschaften

Die Axiomensysteme zur Ableitung von Eigenschaften des Plans werden für alle behandelten Fragen auf die grundsätzlich gleiche Art entwickelt. Wir stellen diese Vorgehensweise hier in Auszügen am Beispiel der Erklärung einer Tasknotwendigkeit dar, und verweisen für eine vollständige Beschreibung der Systematik für die Erklärung dieser und der anderen Fragen auf [See11].
In einem ersten Schritt wird der Basisfall, aus dem sich eine Tasknotwendigkeit ergibt, in Axiomen festgehalten. Dieser Basisfall ist gegeben, wenn der Task Bestandteil der Problemstellung war. Die Notwendigkeit der Problemstellung kann und muss offensichtlich nicht weiter begründet werden, und kann daher eine Erklärung abschließen, ohne zu Nachfragen zu führen. Etwas komplexer gestalten sich Axiome, mit denen sich eine Tasknotwendigkeit aus grundlegenden Kausal- und Dekompositionsbeziehungen ableiten lässt: Besteht im Plan eine Kausalbeziehung zwischen zwei Tasks t_1 und t_2 über einer Formel ϕ, und ist t_2 notwendig für den Plan, so ist auch die Notwendigkeit von t_1 gegeben, denn t_1 macht damit die Ausführung von t_2 möglich. Analog dazu ergibt sich die Notwendigkeit eines Tasks t_1,

wenn er im Zuge einer Dekomposition von t_2 durch die Methode m in den Plan eingefügt wurde. Wichtig ist in beiden Fällen, dass auch t_2 für den Plan notwendig sein muss – auf diese Weise wird die Notwendigkeit, Argumente zu verketten, um eine vollständige Erklärung zu erhalten, auch auf formaler Ebene abgebildet. Die Axiome für die Basisfälle und die einfachen Ableitungen lauten damit folgendermaßen:

Axiome für den Basisfall:

$$Notwendig(InitialerTask)$$
$$Notwendig(Zielzustandstask)$$
$$\forall t : Task\,.\,[Notwendig(t) \Leftarrow Hinzugefuegt(t, InitialesTasknetz)]$$

Axiome für einfache Kausal- und Dekompositionsbeziehungen:

$$\forall t_1 : Task\,.\,[Notwendig(t_1) \Leftarrow$$
$$\exists t_2 : Task; \phi : Formel\,.\,[KausalBeziehung(t_1, \phi, t_2) \wedge Notwendig(t_2)]]$$
$$\forall t_1 : Task\,.\,[Notwendig(t_1) \Leftarrow$$
$$\exists t_2 : Task; m : Methode\,.\,[DekompositionsBeziehung(t_1, m, t_2) \wedge Notwendig(t_2)]]$$

Mit diesen vier Axiomen ist die Notwendigkeit jedes Tasks, den ein Plan enthält, beweisbar. Jeder solche Beweis verkörpert, um von der hier vorgestellten formalen Sichtweise wieder zu einer Erklärung zurückzukommen, ein Argument, wobei sich aus den Axiomen für die Ableitung von Tasknotwendigkeiten direkt aus Kausal- und Dekompositionsbeziehungen klar unterscheidbar funktionale – bei Ableitung der Notwendigkeit eines Tasks auf Basis einer Kausalbeziehung – und konstruktive – bei Ableitung auf Basis einer Dekompositionsbeziehung – Argumente ergeben. Neben diesen Argumenten, die den Kern einer Erklärung bilden, stellen die Beweise für die Beziehungen, auf denen sie basieren, eher eine zusätzliche Detailebene dar, denn wie beispielsweise eine Kausalbeziehung genau abgeleitet wurde ist für das Verständnis der Erklärung nicht so relevant wie die Tatsache, dass diese Beziehung überhaupt besteht.

5.2 Behandlung von Defiziten

In diesem Abschnitt sollen Möglichkeiten gezeigt werden, wie Roherklärungen so modifiziert werden können, dass Defizite entfernt werden, die Schlüssigkeit der Erklärung aber erhalten bleibt. Für die Existenz eines Defizits sind grundsätzlich beliebige Gründe denkbar, zu Beginn des Kapitels wurde beispielsweise die Tatsache,

dass ein Task dem Nutzer nicht bekannt war genannt. Ein anderer Grund könnte darin bestehen, dass der Task für den Nutzer nicht interessant ist. Unabhängig davon, was die tatsächliche Ursache ist, lassen sich die vielfältigen Defizite aber in wenige Klassen einteilen, für deren Behandlung in diesem Abschnitt einige Methoden entwickelt werden sollen.

Wir orientieren uns zur Definition dieser Defizitklassen wieder am Aufbau von Erklärungen und ihren Argumenten. Wie gezeigt wurde besteht eine Erklärung aus einer Anzahl von Argumenten, die wiederum auf Beziehungen im Plan basieren. Diese Beziehungen bestehen aus zwei Planschritten, die durch einen funktionalen (eine Formel) oder konstruktiven (eine Methode) Zusammenhang in Verbindung stehen. Ausgehend von diesem Aufbau sind vier Defizitklassen zu definieren, wobei wir mit dem Begriff *illegal* Erklärungselemente bezeichnen, die aus einem zunächst nicht näher zu bestimmenden Grund nicht in einer Erklärung verwendet werden dürfen:

- Unter **Planschritt-Defiziten** werden alle Defizite zusammengefasst, die in der Referenzierung eines illegalen Planschrittes begründet sind.
- Alle Defizite, die auf Grund der Referenzierung einer illegalen Verbindung bestehen, fallen in die Klasse der **Verbindungs-Defizite**.
- Die Verwendung einer illegalen Art von Argument bedeutet ein Defizit der Klasse der **Argument-Defizite**.
- Ein **Überlange-Erklärung-Defizit** besteht, wenn die Erklärung aus zu vielen Erklärungsschritten aufgebaut ist.

Bei der Entwicklung von Modifikationen gibt es prinzipiell zwei Möglichkeiten, wie mit einem Element einer Erklärung, welches defizitbehaftet ist, verfahren werden soll; Denkbar sind eine Ersetzung des Elements, oder aber seine Entfernung. Die erste Variante hat den Vorteil, dass die Erklärung wenig oder keine Information verliert, aber auch zwei schwerwiegende Nachteile. Das Ersatzstück ist natürlich so zu wählen, dass es in die Erklärung passt, also wie das defizitäre Element auf Zusammenhängen im Plan beruht. Zum einen ist aber vorab keineswegs klar, ob das ersatzweise eingefügte Element nicht selbst wieder zu einem Defizit in der Erklärung führt. Noch schwerer wiegt, dass mit dem im letzten Abschnitt beschriebenen Formalismus alle möglichen Roherklärungen gefunden werden können, das heißt durch eine solche Modifikation würde nur eine Erklärung erzeugt, die bereits vorliegt, und damit nichts gewonnen. Insgesamt scheint diese Variante daher nicht sinnvoll.

Das defizitbehaftete Element vollständig aus der Erklärung zu eliminieren bringt andere Schwierigkeiten mit sich, die teilweise bei der Entwicklung des Beispiel am Anfang des Kapitels 5 schon angedeutet wurden. Die zentrale Anforderung ist bei jeder Veränderung der Erklärung, dass die Argumentationskette trotz der Auslassung erhalten und die Argumentation an sich schlüssig bleiben muss. Wir werden

in der Beschreibung der Modifikationen zeigen, dass diese Bedingungen stets erfüllt sind. Ein formal weniger bedeutsamer, in der praktischen Anwendung aber kritischer Nachteil besteht dagegen darin, dass die weggelassene Information eben ersatzlos gestrichen wird, die Erklärung also an Information und damit an Wert für den Nutzer verliert. Dieser Nachteil wird durch die Anordnung der Schritte, zuerst alle verfügbaren Erklärungen von ihren Defiziten zu befreien und ihre Bewertung erst anschließend vorzunehmen, jedoch so gut wie möglich ausgeglichen, denn damit wird sichergestellt, dass die Erklärungen erst in der Form, in der sie letztlich präsentiert würden, bewertet werden. Basierend auf den vorangegangenen Überlegungen werden wir daher in den nächsten Abschnitten Modifikationen beschreiben, die Defizite einer Erklärung durch Abstraktion von den betroffenen Elementen beheben.

Als Grundlage für die Erläuterung der Modifikationen dient eine Roherklärung für die Notwendigkeit des Tasks `press_Home.People` im vorher bereits aufgegriffenen Plan aus Abbildung 4.2, Seite 43, wobei sich die Vorgehensweise auf alle Arten von Erklärungen übertragen lässt, da die Ansatzpunkte in den grundlegenden Eigenschaften von Erklärungen liegen. Die Erklärung ist in Abbildung 5.1 (S. 58) dargestellt, und beinhaltet drei Defizite, die durch die rote Färbung hervorgehoben werden: Ein Planschritt-Defizit in der Referenzierung von `press_People.More`, ein Argument-Defizit im konstruktiven Argument zwischen `select_Contacts.Details.Mobile` und `call`, und ein Verbindungs-Defizit in der Verwendung der Relation `haveSent`, auf welcher das funktionale Argument zwischen `call` und dem `Goalstate`-Task basiert.
Um die Anwendung der entwickelten Modifikationen allgemein zu beschreiben greifen wir auf eine abstrakt gehaltene Erklärung zurück, die aus vier Tasks A, B, C und D besteht, die mit Argumenten auf Basis von Beziehungen r, s und t verbunden sind:

$$A \xrightarrow{r} B \xrightarrow{s} C \xrightarrow{t} D$$

5.2.1 Behandlung von Planschritt-Defiziten

Die Behandlung eines Planschritt-Defizits durch Auslassung des betroffenen Tasks führt zunächst dazu, dass die Argumentationskette unterbrochen wird. Im oben gezeigten Beispiel würde sich das so äußern, dass die Notwendigkeit von `press_Home.People` durch einen kausalen Link begründet wird, für den kein Konsument existiert. Auf der anderen Seite der entstandenen Lücke befindet sich der Task `select_Contacts.ContactForContactable` als Konsument eines kausalen Links, der nun keinen Produzenten mehr besitzt. Diese beiden Blickwinkel liefern uns aber auch Lösungsansätze zum Schließen der entstandenen Lücke.

Eine Möglichkeit besteht darin, einen neuen Endpunkt für die Kausalbeziehung zu suchen, die vom Task `press_Home.People` ausgeht. Die Notwendigkeit des Tasks `press_Home.People` würde dann damit begründet, dass sein Effekt, die Bedingung `inMode_People` wahr zu machen, als Vorbedingung vom nächsten in der Erklärung folgenden Task, also `select_Contacts.ContactForContactable`, konsumiert wird. Der Plan stützt diese Erklärung allerdings nicht, denn `select_Contacts.ContactForContactable` hat diese Vorbedingung nicht. Es ist aber in der ursprünglichen Erklärung nachzuvollziehen, dass dieser Effekt für die Ausführung des Plans notwendig ist, wenn auch nicht auf direkte Weise wie es mit dieser Modifikation der Erklärung dargestellt wird, sondern mit einer Indirektion über `press_People.More`, die wir nun auslassen.
In einer andere Variante wird ein Ersatz für den Produzenten der Vorbedingung `inMode_Contacts` von `select_Contacts.ContactForContactable` gesucht. Die Erklärung würde dann also besagen, dass `press_Home.People` die Vorbedingung `inMode_Contacts` herstellt. Auch diese Behauptung hat im Plan so keine Basis, aber analog zur vorherigen Argumentation lässt sich sagen, dass `press_Home.People` *letztlich* sehr wohl dazu beiträgt, diese Bedingung wahr zu machen – wiederum über einen Zwischenschritt, der nun ausgelassen wird.

Prinzipiell wäre denkbar, die Suche nach einem neuen Nachfolger bzw. Vorgänger weiter als auf den nächsten in der Erklärung verwendeten Planschritt auszudehnen. Dies wäre jedoch nicht im Sinne eines möglichst gering zu haltenden Wertverlusts der Erklärung, und wird daher nicht weiter verfolgt.
Wird für die Behebung des Planschritt-Defizits in unserer Beispielerklärung die zweite vorgeschlagene Modifikation angewendet, sieht die Erklärung anschließend wie in Abbildung 5.2, Seite 59 dargestellt aus, wobei der Teil der Erklärung, der dabei verändert wurde, blau hervorgehoben ist.

Um die beiden gefundenen Modifikationsmöglichkeiten allgemein zu beschreiben wird an der oben formulierten allgemeinen Erklärung ein Planschritt-Defizit am referenzierten Task C angenommen; Die Anwendung der beschriebenen Modifikationen führt dann zu den folgenden Varianten:

$$A \xrightarrow{r} B \xrightarrow{s} C \xrightarrow{t} D$$

Suche eines neuen Nachfolgers:

$$A \xrightarrow{r} B \xrightarrow{s} D$$

Suche eines neuen Vorgängers:

$$A \xrightarrow{r} B \xrightarrow{t} D$$

Durch die aus diesen Modifikationen resultierende Rekombination von Planschritten und Argumenten können unter bestimmten Umständen Erklärungen erzeugt

werden, die aus Planungssicht unlogische Argumente verwenden. Durch die Suche nach einem Nachfolger kann zum Beispiel eine Konstellation entstehen, bei der die Notwendigkeit eines Task damit begründet wird, dass er Teil der Zerlegung eines primitiven Tasks ist, was natürlich unmöglich ist. Aus diesem Grund sollten Bestandteile von Erklärungen, die verändert wurden, entsprechend kenntlich gemacht werden, so dass bei der Verbalisierung der Erklärung der Umstand, dass die Erklärung nicht vollständig und in ihrer ursprünglichen Form dargestellt werden kann, berücksichtigt werden kann. Die Abbildungen der zunehmend modifizierten Beispielerklärung geben diesen Sachverhalt wieder, indem für veränderte Argumente gestrichelte statt durchgezogene Pfeile verwendet werden.

Abschließend sei bemerkt, dass die Modifikationen, wie auch anhand dieser Formalisierung erkennbar ist, nicht immer anwendbar sind. Besteht am ersten oder letzten in einer Erklärung referenzierten Task ein Planschritt-Defizit, gibt es natürlich keinen ersatzweise referenzierbaren Vorgänger bzw. Nachfolger. Diese Fälle sind jedoch ohnehin nicht sinnvoll zu behandeln, da beispielsweise die Entfernung des Zielzustand-Tasks auf Grund eines Defizits die Erklärung ihres logischen Abschlusses berauben würde.

5.2.2 Behandlung von Argument-Defiziten

Wie bei der Eliminierung eines Planschritts bedeutet auch das Weglassen eines ganzen Erklärungsschrittes einen Bruch in der Argumentationskette; Kann in unserem Beispiel das Argument zur Erklärung der Notwendigkeit von `select_Contacts.Details.Number` nicht verwendet werden, so gibt es zunächst auch keinen Grund mehr, die Notwendigkeit von `call` zu begründen, wodurch die Erklärung nicht auf einen der Basisfälle zurückgeführt werden kann. Es muss also wieder eine Möglichkeit gefunden werden, diese Lücke zu schließen.

Eine solche Möglichkeit besteht darin, die Notwendigkeit von `select_Contacts.Details.Number` mit einer anderen Argumentation zu begründen. Entsprechend der Eigenschaft der Argumentationsketten, dass ein weiter vorne gemachter Erklärungsschritt direkt oder indirekt zu allen folgenden führt, kann daher die Erklärung der Notwendigkeit des nachfolgenden Schritts, in unserem Beispiel `call`, herangezogen werden. Diese Vorgehensweise entspricht mit allen daraus entstehenden Folgen der Suche nach einem neuen Vorgänger für die Begründung von C, wie sie im letzten Abschnitt formalisiert wurde.
Ein anderer Ansatz kann es sein, den Grund, aus dem die Notwendigkeit von `select_Contacts.Details.Number` überhaupt erklärt werden muss, zu beseitigen. Dies bedeutet das Argument, welches die Verwendung von `select_Contacts.ContactForContactable` im Plan begründet, direkt aus einer Beziehung zu `call` abzuleiten, und den Zwischenschritt über `select_Contacts.Details.Number` auszulassen. Die Anwendung dieser Modifikation erfolgt genau analog zur Ersetzung des Nachfolgers von

`select_Contacts.ContactForContactable`, was bereits als Methode zur Eliminierung eines Planschritts bestimmt werden konnte. Abbildung 5.3 auf Seite 60 zeigt die aus dieser Modifikation hervorgehende Beispielerklärung.

Die Anwendung dieser Modifikationen führt demnach zu den folgenden Varianten der Ausgangserklärung, wenn am Argument, welches auf einer Verbindung von B und C basiert, ein Argument-Defizit angenommen wird:

$$A \xrightarrow{r} B \xrightarrow{s} C \xrightarrow{t} D$$

Verwendung des nachfolgenden Arguments:

$$A \xrightarrow{r} B \xrightarrow{t} D$$

Eliminierung der Notwendigkeit des Arguments:

$$A \xrightarrow{r} C \xrightarrow{t} D$$

5.2.3 Behandlung von Verbindungs-Defiziten

Ein Verbindungs-Defizit zu beheben gestaltet sich deutlich einfacher als die Behebung der bisher behandelten Defizite, da hierbei die Struktur der Erklärung nicht geschädigt wird. Warum dies so ist, lässt sich am Aufbau von Dekompositions- und Kausalbeziehungen leicht nachvollziehen. In unserem Beispiel aus der Smartphone-Domäne besteht ein solches Verbindungs-Defizit an der Kausalbeziehung, mit der die Notwendigkeit von `call` begründet wird. Diese Beziehung besteht aus zwei Tasks, in diesem Fall dem genannten `call` und dem `GoalstateTask`, der den Zielzustand repräsentiert, und der konkreten Formel, über der der kausale Zusammenhang besteht, hier `haveSent`. Die Tatsache, dass dieser kausale Zusammenhang besteht, lässt sich jedoch auch ohne weiteres ohne die Formel ausdrücken – eine Verbalisierung dieses Erklärungsschrittes könnte dann eben lauten „Der Task 'call' ist notwendig für den Plan, da er eine Vorbedingung für den 'GoalstateTask' erfüllt." Die Erklärung bleibt auch in dieser Form verständlich, `haveSent` nicht explizit als dieser Beziehung zu Grunde liegende Formel aufzuführen ist also lediglich die Aussparung eines Details.

Entsprechend dieser Überlegungen ist zur Behebung eines Verbindungs-Defizits lediglich die konkrete Beziehung auszulassen, in unserem allgemeinen Beispiel die defizitäre Beziehung s, auf der das Argument zwischen B und C basiert:

$$A \xrightarrow{r} B \xrightarrow{s} C \xrightarrow{t} D$$

Auslassung der Beziehung:

$$A \xrightarrow{r} B \longrightarrow C \xrightarrow{t} D$$

Durch Behebung des Verbindungs-Defizits an der Beispielerklärung ist diese endlich frei von Defiziten, und in dieser Form in Abbildung 5.4 (S. 61) zu sehen.

5.2.4 Behandlung von Überlange-Erklärung-Defiziten

Die Klasse der Überlange-Erklärung-Defizite nimmt unter den hier diskutierten Defizitklassen eine Sonderrolle ein. Es ist sinnvoll, Erklärungen zu kürzen, denn wie in Kapitel 4.3.2 diskutiert wurde kann ein zu großer Umfang eine Erklärung durchaus unbrauchbar machen. Dieses Problem ist aber im Gegensatz zu den drei Defizitklassen, die auf Argumenten definiert sind, einerseits konkreter, da nur genau eine Eigenschaft einer Erklärung, eben eine zu umfangreiche Erklärung, zu diesem Defizit führen kann. Andererseits liefert es, da es auf der Erklärungs- statt auf der Argumenteebene definiert ist, keinen Ansatzpunkt zu seiner Behebung: So liegt zwar die Beseitigung dieses Defizits durch die Entfernung eines Arguments aus der Erklärung nahe, welches Argument der Erklärung man dabei weglässt ist jedoch nicht festgelegt. Unabhängig von diesen Besonderheiten ist die Behandlung solcher Defizite jedoch einfach zu bestimmen, denn es kann dazu auf die in Abschnitt 5.2.2 beschriebenen Mechanismen zur Behandlung von Argument-Defiziten zurückgegriffen werden, die ja gerade ein Argument aus einer Erklärung entfernen.

5.2.5 Anmerkungen zum Ansatz

Es stehen nun für alle Defizitklassen, die in dieser Arbeit behandelt werden sollen, Modifikationen zur Verfügung, aus denen sich einige Implikationen für den zu Beginn dieses Kapitels beschriebenen Ansatz ergeben. Erstens führt die Uneindeutigkeit der anzuwendenden Modifikationen, die vor allem bei der Behandlung von Überlange-Erklärung-Defiziten, aber auch von Planschritt- und Argument-Defiziten auftritt, dazu, dass aus jeder Erklärung durch Behebung ihrer Defizite eine Vielzahl von Nachfolgern erzeugt werden kann. Zur letztlichen Ausgabe an den Nutzer stehen damit umso mehr Erklärungen zur Auswahl, je mehr Defizite die einzelnen Roherklärungen hatten. Diese Vergrößerung der Kandidatenzahl wird vor allem in einer praktischen Anwendung des beschriebenen Ansatzes zu berücksichtigen sein.

Zweitens führt die Anwendung der Modifikationen zu veränderten Erklärungen, teilweise zu einer Rekombination der verwendeten Beziehungen und Schritte des Plans zu neuen Argumenten. Dabei besteht die Möglichkeit, dass die Behandlung eines Defizits implizit weitere mit auflöst. Dies ist beispielsweise denkbar, wenn ein Argument entfernt worden ist, das auf einer Beziehung basiert, welche ebenfalls als defizitär markiert war. Andererseits können durch die Modifikation auch neue Defizite an der Erklärung produziert worden sein. Aus diesen Gründen ist jede der entwickelten Erklärungen also nicht nur einmalig zu Beginn, sondern nach jeder

Modifikation erneut auf Defizite zu untersuchen, um sicherzustellen, dass am Ende der Reihe von Modifikationen wirklich eine defizitfreie Erklärung steht, aber auch nicht mehr Information weggelassen wird als notwendig.

Zwei Aspekte des zu Beginn des Kapitels vorgestellten Ansatzes wurden bisher noch vernachlässigt: Zum einen wurden zwar Modifikationen zur Behebung von Defiziten entwickelt, aber es gibt noch keine Information darüber, wann tatsächlich ein Defizit vorliegt, um die Anwendung dieser Modifikationen auszulösen. Zum anderen ist der Schritt der Selektion, der auf Grundlage der Bewertung der defizitfreien Erklärungen auszuführen ist, zwar theoretisch beschrieben, aber es wurde noch nicht bestimmt, welche Eigenschaften welchen Wert besitzen. In Kapitel 4.3.2 wurde festgestellt, dass Defizite wie auch Werte nicht generell, sondern vom konkreten Anwendungsfall abhängig anzugeben sind. Dieses Kapitel schließt daher die Beschreibung der unabhängigen Grundfunktionen des Frameworks ab, und wir wenden uns mit dem nächsten Kapitel diesen offenen Punkten zu.

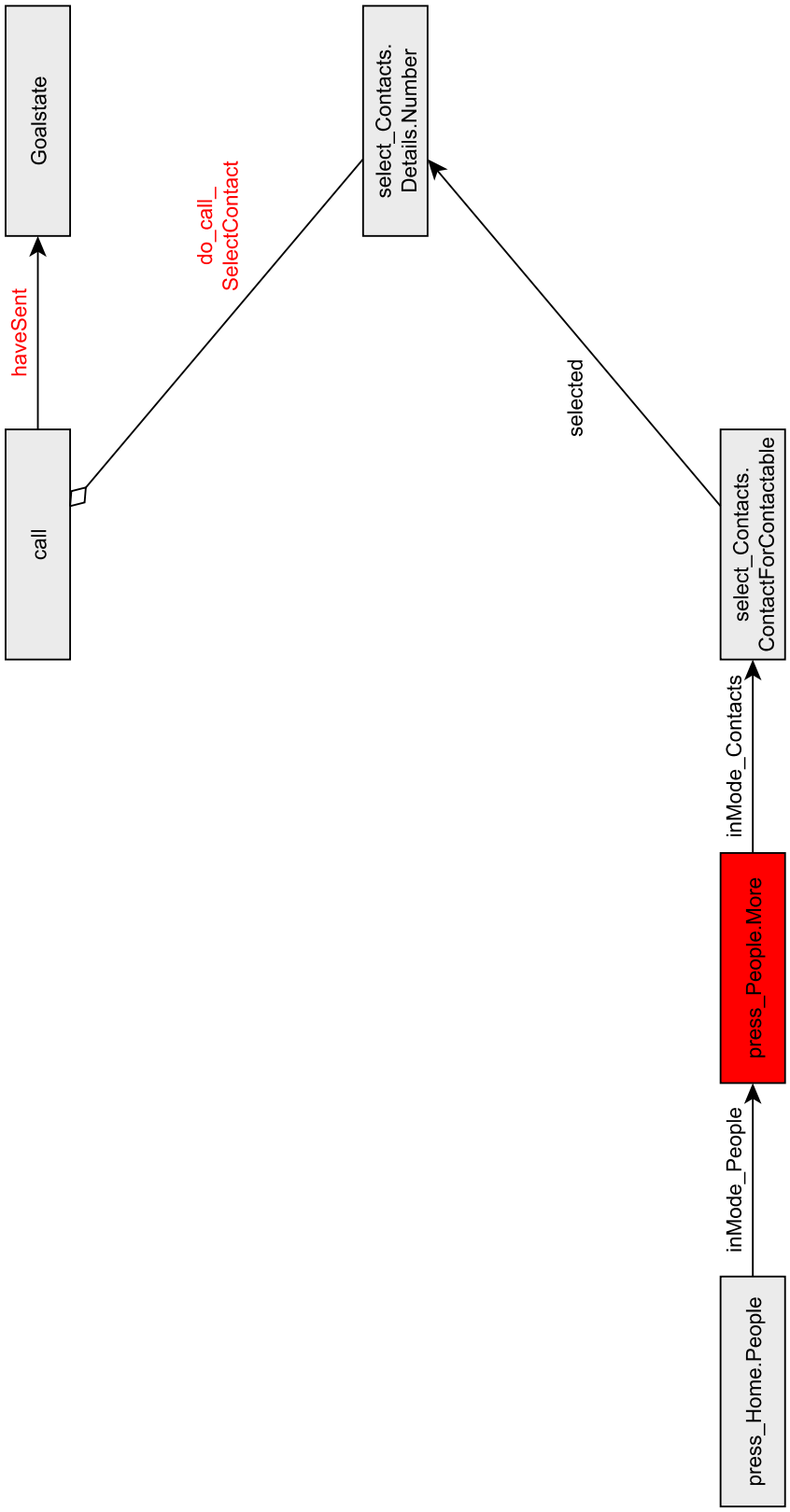


Abbildung 5.1: Eine Roherklärung für die Notwendigkeit von `press_Home.People` im Plan aus Abbildung 4.2. Rot hervorgehobene Elemente zeigen Defizite an.

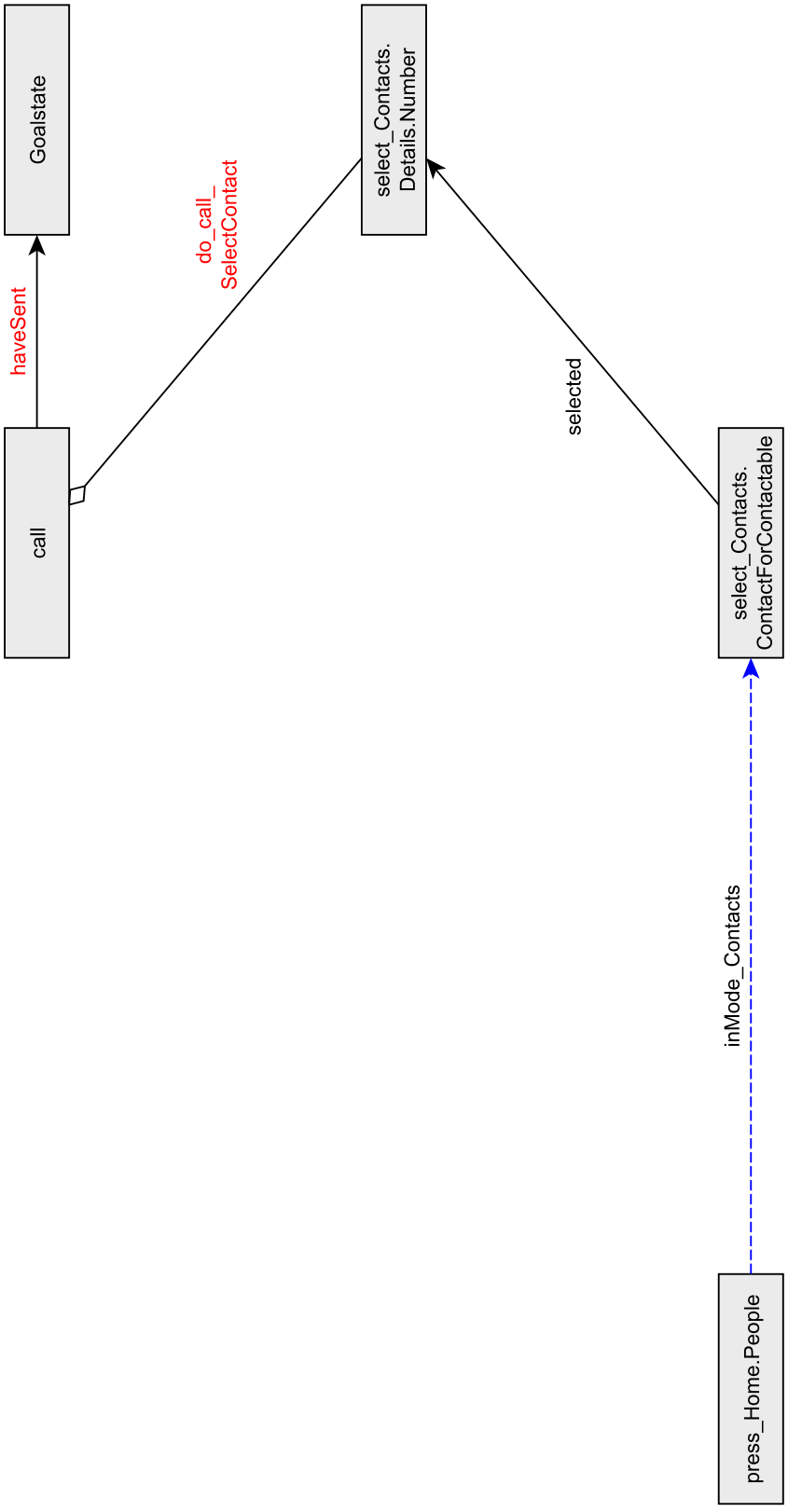


Abbildung 5.2: Die Erklärung nach der Suche nach einem neuen Vorgänger für `select_Contacts.ContactForContactable`

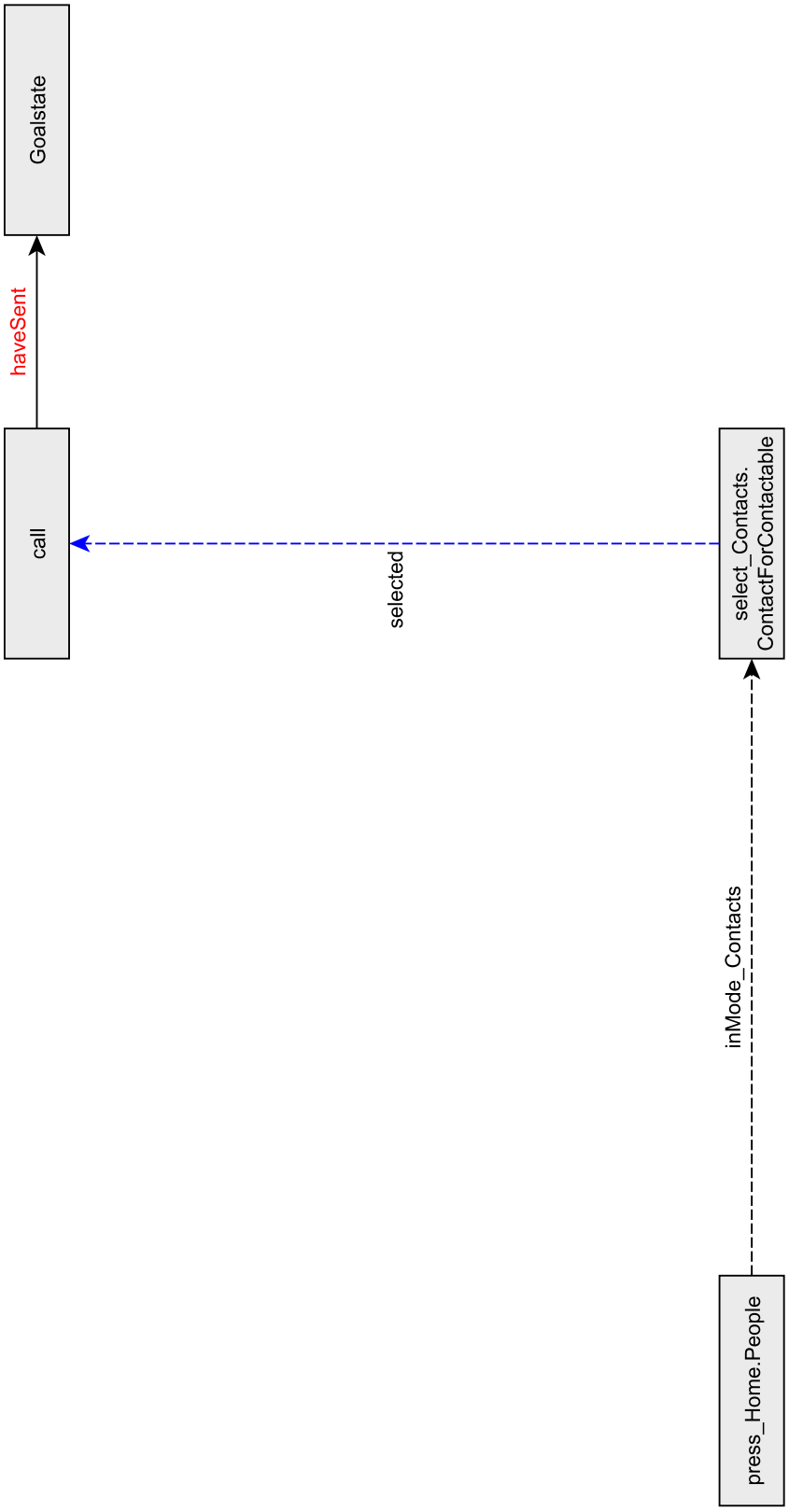


Abbildung 5.3: Die Erklärung nach der Behebung des Argument-Defizits durch Eliminierung der Notwendigkeit des Arguments

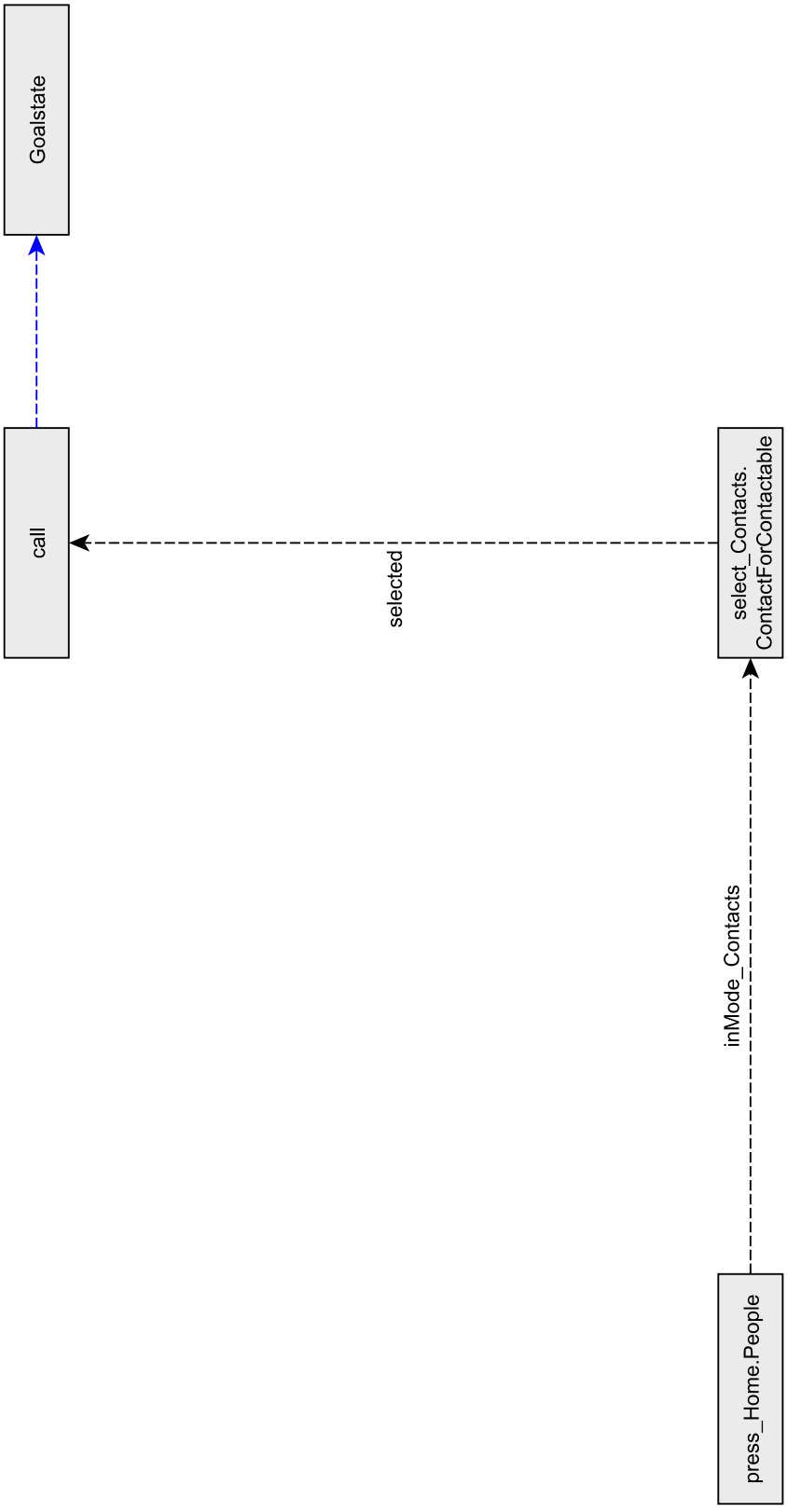


Abbildung 5.4: Die Erklärung nach Behandlung des Verbindungs-Defizits

6 Wissensbasierte Detektion von Defiziten und Bewertung von Erklärungen

Wir haben im vorangegangenen Kapitel einen Ansatz vorgestellt, wie die Erklärung von Plänen durch die Erzeugung von Roherklärungen, deren Befreiung von Defiziten und schließlich einer Selektion auf Grundlage einer Bewertung dieser Erklärungen erfolgen kann. Die Vorgehensweise wurde bislang sehr allgemein gehalten, indem sich die entwickelten Modifikationen nur an Defizit-Klassen orientieren statt an konkreten Defiziten; Ebenso wurde noch nicht festgelegt, was genau den Wert einer Erklärung ausmacht, denn wie in Kapitel 4.3 festgestellt wurde, muss die Definition dieser Defizite und Werte vom konkreten Anwendungsfall abhängig gemacht werden. In diesem Kapitel setzen wir uns nun zum einen damit auseinander, wie diese nutzerbasierten Definitionen dem Framework zur Verfügung gestellt werden können. Zum anderen werden Mechanismen vorgestellt, die Defizite detektieren und daraufhin die Anwendung von Modifikationen anstoßen, sowie die Bewertung von Erklärungen für die abschließende Selektion realisieren können. Insgesamt beschreibt dieses Kapitel also die Schnittstelle zwischen dem Framework und darauf basierenden Applikationen.

Wir untersuchen zunächst, warum eine wissensbasierte Repräsentation dieser Definitionen für den in dieser Arbeit gewählten Ansatz günstig ist. In Abschnitt 6.2 wird anschließend gezeigt, wie Erklärungen und die dem Plan zu Grunde liegende Domäne in der Wissensbasis dargestellt werden. Abschnitt 6.3 beschreibt die Funktionsweise der Schnittstelle zwischen dem Framework und darauf basierenden Anwendungen, die in der Definition von Defiziten und Werten und den darauf aufbauenden Mechanismen zur Defizit-Detektion und Erklärungsbewertung besteht. Die Abschnitte 6.4 bis 6.6 setzen sich schließlich damit auseinander, welche Rolle Wissen über den Nutzer, die Benutzerschnittstelle einer Applikation sowie die Domäne des zu erklärenden Plans für die Definition von Werten und Defiziten spielt.

6.1 Repräsentationsform

Um eine passende Repräsentationsform für die Defizit- und Wertdefinitionen zu finden, sollen zunächst die Aufgaben, die von diesem Teil des Frameworks zu erfüllen sind, genau betrachtet werden. Für die Bewertung einer Erklärung müssen relevante Eigenschaften gefunden und der Wert bestimmt werden, den sie zum Gesamtwert einer Erklärung beitragen. Es liegt nahe, dass dabei eine gewisse Grundmenge von Eigenschaften die Hauptrolle spielen wird, beispielsweise ob der Nutzer an bestimmten Domänenelementen interessiert ist. In diesem Fall würden also alle Elemente, die den Nutzer interessieren, eine *Klasse* von Elementen bilden. Ähnliches gilt auch für Eigenschaften, aus denen Defizite an einer Erklärung abzuleiten sind. Zum Beispiel würde an all den Argumenten, die einen dem Nutzer unbekannten Planschritt referenzieren, ein Planschritt-Defizit annotiert werden, und damit eine Klasse von „Unbekannter-Planschritt-Argumenten“ gebildet werden. Insgesamt sind also Erklärungen, ihre Argumente und deren zu Grunde liegenden Beziehungen im Plan in bestimmte Klassen einzuteilen, weshalb es naheliegt, eine Repräsentationsform zu wählen, die eine solche Klassifikation gut unterstützt. Eine weitere Anforderung, die die Repräsentation des Regelwerks erfüllen muss, ist, dass die Ausdrucksmächtigkeit für die Definition von Werten und Defiziten möglichst groß sein sollte, um auch komplexe Definitionen zu ermöglichen.

Wir setzen aus den genannten Gründen auf eine Formulierung der Wert- und Defizitdefinitionen durch eine deklarative Modellierung als Wissensbasis, mit einem an der in Kapitel 2.2.1 vorgestellten Beschreibungslogik orientierten Aufbau. Eine solche Wissensbasis erfüllt nicht nur die genannten Anforderungen, sondern bietet darüber hinaus weitere Vorteile für das Framework. Von zentraler Bedeutung ist, dass dies eine etablierte Darstellungsweise ist, für die eine Anzahl von Klassifikationsmechanismen zur Verfügung stehen, so dass diese für diese Arbeit nicht extra zu entwickeln sind. Diese Mechanismen können neben der Bildung von Klassen, wie es oben als sinnvoll beschrieben wurde, auch dazu dienen, die Konsistenz der Regelwerke zu prüfen. Beispielsweise sind Tests denkbar, die eine Menge von Definitionen daraufhin untersuchen, ob es überhaupt Erklärungen geben kann, die frei von Defiziten sind. Ein weiterer Vorteil besteht darin, dass die Regeln in einer Wissensbasis in expliziter Form vorliegen. Sie sind damit gut kommunizierbar, was dazu genutzt werden könnte, die Arbeit des Erklärungssystems selbst wiederum zu erklären, indem beispielsweise angegeben wird, auf Grundlage welcher Regel an einem Teil einer Erklärung ein Defizit detektiert wurde. Ebenfalls günstig ist es, dass Wissensbasen dieser Form jederzeit erweiterbar sind. Inwiefern diese Eigenschaft eine Rolle spielen kann, werden wir im abschließenden Ausblick der Arbeit genauer diskutieren.

Die in den nächsten Abschnitten folgenden Abbildungen zur Beschreibung des Aufbaus der Wissensbasis verwenden die in 6.1 gezeigte Darstellungsweise. Kardinali-

täten wurden hierbei aus Gründen der Übersicht ausgelassen und sind der Beschreibung der jeweils zu Grunde liegenden Objekte zu entnehmen.

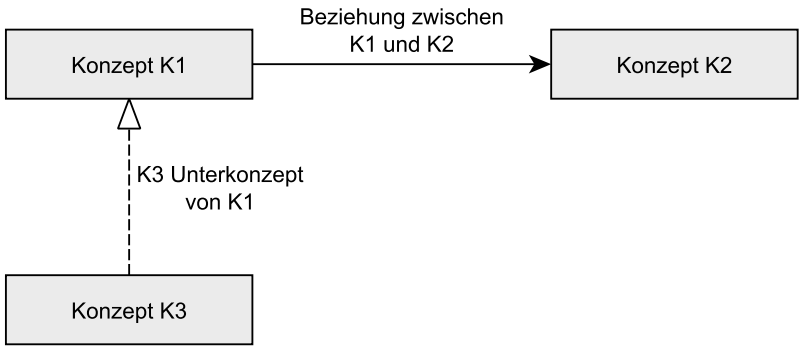


Abbildung 6.1: Legende für die Darstellung der Wissensbasis

6.2 Grundlegende Bestandteile der Wissensbasis

Um eine wissensbasierte Definition von Defiziten und Werten vornehmen zu können, ist zunächst eine Grundlage in Form einer entsprechenden Repräsentation von Erklärungen zu schaffen. Da Erklärungen sich auf Eigenschaften des zu erklärenden Plans beziehen, sind auch diese Eigenschaften zu repräsentieren. Pläne wiederum werden aus den Aktionen, Relationen etc. aufgebaut, die eine zu Grunde liegende Domäne zur Verfügung stellt, weshalb auch eine Repräsentation der Domäne notwendig ist.
Es sei an dieser Stelle noch angemerkt, dass, obwohl der gesamte Erklärungsprozess natürlich eng an den zu erklärenden Plan gekoppelt ist, dieser Plan nicht vollständig in der Wissensmodellierung dargestellt wird. Wir haben uns in dieser Arbeit stattdessen hauptsächlich auf Wissen gestützt, welches unabhängig vom konkret betrachteten Plan schon vor einem Erklärungsprozess verfügbar, also von dauerhafter Gültigkeit ist. Wir werden im abschließenden Ausblick jedoch den Aspekt der Landmarken eines Plans betrachten, deren Kenntnis sich in mehreren Bereichen der Planerklärung als nützlich erweisen könnte.

6.2.1 Domäne

Das Fundament unserer Wissensbasis bildet wie beschrieben eine Repräsentation der PANDA-Domäne, in der ein zu erklärender Plan begründet ist. Die Grundlage für eine solche Repräsentation sind Konzepte, die die Bestandteile von Domänen, und Relationen, die die dazwischen bestehenden Beziehungen allgemein wiedergeben. Wir verzichten an dieser Stelle auf eine Aufführung all dieser Bestandteile und

verweisen stattdessen auf Kapitel 2.1.3, wo der Aufbau entsprechender Domänen beschrieben wird. Bei der Modellierung wurde von einigen Aspekten abstrahiert, die zwar für die Abläufe im Planungssystem von Bedeutung sind, dem Nutzer aber nicht präsentiert werden und daher auch nicht für den Erklärungsvorgang von Bedeutung sind. Dazu zählen zum einen Dekompositionsaxiome sowie in Methoden definierte kausale Links und Variablenbedingungen, die komplett weggelassen wurden; Zum anderen wird nicht der genaue Aufbau von Formeln in Vor- und Nachbedingungen von Tasks, sondern lediglich die in ihnen vorkommenden Relationen nachgebildet. Da in Abschnitt 6.6 dieses Kapitels noch einige Erweiterungen der Repräsentation der Domäne vorgeschlagen werden, wird für eine grafische Darstellung dieser Repräsentation auf Abbildung 6.5 auf Seite 81 verwiesen.

6.2.2 Erklärung

In Kapitel 4.3.1 wurde beschrieben, dass eine Untersuchung der Eigenschaften von Erklärungen auf zwei Ebenen erfolgen muss. Einerseits ist die Sichtweise einzunehmen, dass eine Erklärung aus einer Anzahl von Argumenten aufgebaut ist, um beispielsweise den Umfang einer Erklärung zu bestimmen. Andererseits sind die einzelnen Argumente zu betrachten, die jeweils eine Planeigenschaft erklären, wozu sie sich wiederum auf andere Planeigenschaften stützen.
Diese zwei Sichtweisen sind auch in der Wissensbasis zu modellieren, um Anforderungen auf den entsprechenden Ebenen umsetzen zu können. Zur Darstellung von Erklärungen als Ganzem dient das Konzept `Roherklärung`, und zur Beschreibung ihres Aufbaus aus mehreren einzelnen Erklärungen eine Relation `bestehtAus` sowie ein entsprechendes Konzept `Erklärung` für ihre Argumente. Der Beweis von Planeigenschaften durch Erklärungen kann mit Hilfe eines Konzepts `Planeigenschaft` und einer Relation `erklärtDurch`, die die Verbindung zu der zugehörigen Erklärung herstellt, modelliert werden. Da die Erklärungen dazu wieder auf Planeigenschaften zurückgreifen, wird das Modell noch um eine Relation `basiertAuf` ergänzt.

Zu diesen Basiskonzepten kommen noch Unterkonzepte für eine genauere Klassifikation. Argumente werden zunächst noch in konstruktive und funktionale Argumente aufgeteilt; Die grundlegenden Planeigenschaften sind dagegen weiter aufzuschlüsseln. Zum einen sind die unterschiedlichen Beziehungen die im Plan herrschen zu modellieren, wofür das Konzept `Beziehung`, und diesem untergeordnet `KausalBeziehung` sowie `DekompositionsBeziehung` in die Wissensbasis integriert werden. Beziehungen bestehen stets zwischen zwei Planschritten, was durch die Relationen `hatVorgänger` und `hatNachfolger` ausgedrückt wird. Da Kausalbeziehungen über gewissen Formeln der `PANDA`-Domäne bestehen, stehen diese darüber hinaus durch eine Relation `überFormel` mit einer solchen in Beziehung, während Dekompositionsbeziehungen mit einer Relation `durchMethode` mit einer Methode der zu Grunde liegenden Domäne verbunden werden. Im Plan bestehende

Fakten, wie beispielsweise konkrete kausale Links, werden ebenfalls als entsprechende Beziehungen modelliert, da sie nicht grundsätzlich anders zu behandeln sind als diese; Der Unterschied zu den allgemeinen abgeleiteten Beziehungen wie zum Beispiel einer Kausalbeziehung besteht lediglich darin, dass Fakten nicht weiter zu erklären sind. Schließlich ist noch wiederzugeben, dass Argumente oder Beziehungen durch Modifikationen verändert worden sein können, so dass sie nicht mehr präzise die Eigenschaften des Plans darstellen. Hierzu dient eine Relation `modifiziert`, mit der angegeben werden kann, falls eine solche Veränderung vorgenommen wurde.

Die allgemeine Modellierung von Planerklärungen ist mit diesen Konzepten und Relationen abgeschlossen, und wird in Abbildung 6.2 (S. 79) noch einmal in einer Übersicht gezeigt, in der auch die im nächsten Abschnitt vorgestellten nötigen Erweiterungen eingeschlossen ist. Für die Erklärung konkreter Planeigenschaften, beispielsweise einer Tasknotwendigkeit, ist die Wissensbasis gegebenenfalls um entsprechende Planeigenschaften und Argumenttypen zu erweitern. Da diese für den Aufbau der Wissensbasis an sich aber keine Rolle spielen, wird an dieser Stelle auf eine Auflistung dieser Elemente verzichtet.

6.3 Defizite und Werte – Schnittstelle zwischen Framework und Anwendung

Wie zu Beginn des Kapitels beschrieben sind für unseren Ansatz noch zwei Aspekte zu klären: Einerseits, wie Defizite und Werte von Erklärungen definiert, und wie andererseits mit diesen Definitionen anschließend Defizite gefunden und der Wert einer Erklärung bestimmt werden können. In den nächsten beiden Abschnitten wird jeweils zunächst erläutert, wie eine Repräsentation von Defiziten beziehungsweise Werten in der Wissensbasis bezüglich der darin abgebildeten Erklärungen erfolgen kann. Anschließend werden Mechanismen beschrieben, die die Defizit-Detektion und Erklärungsbewertung umsetzen. Diese Mechanismen stellen dabei einen festen Teil des Frameworks dar, wohingegen die Anwendungsseite die Definitionen liefern muss, um diese Mechanismen zu steuern.

6.3.1 Definition und Detektion von Defiziten

In Kapitel 5.2 wurde beschrieben, dass alle relevanten Defizite von Roherklärungen auf einige Basisfälle zurückzuführen sind. Es ist daher naheliegend, die Klassifikation von Erklärungen und ihren Bestandteilen zur Detektion von Defiziten zu nutzen, und eine Erklärung in „frei von Defiziten“ und „mit Überlange-Erklärung-Defizit“, ihre Argumente in „frei von Defiziten“ und „mit Argument-Defizit“ etc. einzuteilen. Die Detektion von Erklärungsdefiziten kann dann zum Beispiel so ablaufen, dass

geprüft wird, ob eine Erklärung als „mit Überlange-Erklärung-Defizit“ zu klassifizieren ist oder nicht. Ist dies der Fall, dann kann dies als Auslöser zur Modifikation der Erklärung an der betroffenen Stelle verwendet werden. Um diese Aufteilung zu realisieren wird nun beispielsweise für das Konzept `Argument` ein Unterkonzept `Argument-Defizit` gebildet. Dieser Konzeptdefinition werden jedoch keine weiteren Restriktionen auferlegt, denn dieser feste Teil des Modells soll ja gerade keine Aussagen darüber machen, wann ein Argument defizitär ist. Auf die selbe Weise werden auch die anderen Defizitklassen modelliert, so dass sich insgesamt ein Aufbau von Erklärungen und den an ihr definierten Defizitklassen ergibt, die in Abbildung 6.2 zu sehen ist. Die Abbildung zeigt dabei in der Konzepthierarchie eine Abweichung von den Defizitklassen, die bisher verwendet wurden. Ein Planschritt-Defizit fehlt, stattdessen sind ein `Vorgänger-` und ein `Nachfolger-Defizit` aufgeführt. Diese Abweichung liegt darin begründet, dass Beziehungen ja gerade zwei Planschritte in Verbindung bringen. Würde nun einfach ein Planschritt-Defizit an einer Beziehung detektiert, wäre nicht klar, welcher der beiden Planschritte aus der Erklärung zu eliminieren ist. Daher wird zwischen Vorgänger- und Nachfolger-Defiziten unterschieden, so dass eine eindeutige Identifikation des zu entfernenden Tasks ermöglicht wird. Die Behandlung dieser Defizite entspricht dabei der bisher für Planschritt-Defizite beschriebenen.

Damit sind die Basisfälle der Defizite modelliert, die aber eben nur die abstrakten Defizitklassen verkörpern – wie sind nun konkrete Defizite zu definieren? Zudem besteht neben dieser offenen Frage auch ein Problem im vorgeschlagenen Detektions-Mechanismus. Wenn nämlich den Defizit-Konzepten keine Restriktionen auferlegt werden, würde zum Beispiel jedes Argument als defizitär klassifiziert. Diese beiden Schwachpunkte lassen sich jedoch folgendermaßen auflösen: Da beispielsweise jedes konkrete Argument-Defizit im Grunde genommen eine spezielle Variante der allgemeinen Argument-Defizit-Klasse darstellt, liegt nahe, auch hier wieder auf die Bildung von Unterkonzepten zurückzugreifen. Jede Eigenschaft eines Arguments, die ein Defizit der Argument-Defizit-Klasse ergeben soll, ist also als Unterkonzept von `Argument-Defizit` abzubilden. Damit lässt sich nun auch der beschriebene Mechanismus zur Detektion von Defiziten präzisieren: Ein Element einer Erklärung besitzt ein Defizit einer bestimmten Defizitklasse, wenn es mindestens einem der Unterkonzepte des entsprechenden Defizitklassen-Konzepts zuzuordnen ist.
Wichtig ist an dieser Stelle, dass für die Behandlung von Defiziten nicht zwischen den Hauptbestandteilen der Erklärung und den Beweisen für die darin verwendeten Beziehungen unterschieden wird. Auch wenn von der Erklärung möglicherweise nur die Argumente und die Beziehungen gezeigt werden, auf denen diese basieren, sollen beispielsweise Nachfragen zu Details, wie eben der Ableitung solcher Beziehungen, behandelt werden können. Daher macht es Sinn, die Erklärung in all ihren Bestandteilen konsistent mit den in der Wissensbasis definierten Defiziten zu halten, also auch in den Details für eine defizitfreie Erklärung zu sorgen.

6.3.2 Bewertung von Erklärungen

Wir haben in Kapitel 4.3.1 einen allgemeinen Ansatz gezeigt, mit dem eine Erklärung bewertet werden kann. Die Bewertung erfolgt dabei auf der Grundlage von Eigenschaften, die nutzer- und anwendungsspezifisch festzulegen sind. Um eine solche Bewertungsfunktion umzusetzen ist also zu bestimmen, welche dieser Eigenschaften bei einer Erklärung vorliegen. Daher ist eine ähnliche Vorgehensweise angebracht, wie sie bereits zur Detektion von Defiziten gedient hat: Jede erwünschte Eigenschaft wird durch ein Unterkonzept des Erklärungsbestandteils ausgedrückt, der diese Eigenschaft besitzen kann. Soll beispielsweise ein Argument, welches einen für den Nutzer interessanten Planschritt referenziert, zum Wert der Erklärung beitragen, ist dies durch ein Unterkonzept des Konzepts `Argument` mit einer entsprechenden Restriktion zu modellieren.

An dieser Stelle unterscheidet sich die Bewertung einer Erklärung in mehreren Punkten von der Detektion von Defiziten. Erstens war für die Erkennung von Defiziten nicht relevant, ob nur eine oder mehrere Eigenschaften erfüllt sind, die ein bestimmtes Defizit bedeuten. Dies ist bei der Bewertung nicht der Fall, denn es sind alle Attribute einzubeziehen, die zum Wert einer Erklärung beitragen können. Zweitens besitzen die Defizitklassen kein Pendant zum Wert einer Eigenschaft, es spielt nur eine Rolle, ob ein bestimmtes Defizit vorliegt oder nicht. Unterschiedlichen Eigenschaften kommt dagegen unterschiedliche Bedeutung zu, je nachdem wie wichtig sie im konkreten Anwendungsfall sind. Aus diesem Grund sind die Unterkonzepte, die die verschiedenen erwünschten Attribute verkörpern, mit einem Wert zu versehen, durch den die Relevanz der Attribute in Beziehung zueinander gesetzt werden kann. Der dritte Unterschied liegt darin, dass die Bewertung einer Erklärung nicht alle ihre Bestandteile einbeziehen sollte. Wie in Kapitel 5.1 beschrieben wurde, werden die Beziehungen, auf denen die Argumente einer Erklärung fußen, ebenfalls formal abgeleitet, um eine vollständig auf den Plan bezogene Erklärung zu erhalten. Die Ableitungen dieser Beziehungen sind jedoch für die Präsentation der Erklärung nur von untergeordneter Bedeutung, da sie eher Detailinformationen darstellen. Aus diesem Grund erfolgt die Bewertung einer Erklärung nur auf Grundlage ihrer eigenen Eigenschaften sowie der ihrer Hauptargumente.
Insgesamt muss die Bewertung einer Erklärung dann, ähnlich der Detektion von Defiziten, aber unter Beachtung dieser Unterschiede, folgendermaßen ablaufen: Für die Erklärung selbst, ihre Argumente und die von den Argumenten verwendeten Beziehungen wird jeweils geprüft, als welche Unterkonzepte ihres Typs sie zu klassifizieren sind. Die Bewertung der Erklärung insgesamt ergibt sich dann aus der Summe der Werte, die diesen Unterkonzepten jeweils zugeordnet sind.

6.4 Wissen über den Nutzer

Wie in Kapitel 4.3.3 vorgeschlagen soll die Definition von Defiziten und Werten auf Grundlage von Eigenschaften und Präferenzen des Nutzers erfolgen. Eine Zusammenstellung von Wissen über den Nutzer, aus der solche Definitionen abgeleitet werden können, nennen wir *Nutzermodell*.

Wir müssen uns zunächst die Frage stellen, welche Informationen für die Bildung eines solchen Nutzermodells herangezogen werden sollen. Das zentrale Kriterium hierfür ist, wie nützlich diese Informationen für das Erklärungssystem sind, das heißt ob und in welchem Maß, sie zur Verbesserung der Erklärung beitragen können. Dabei ist festzuhalten, dass grundsätzlich unendlich viel Information über den Nutzer gesammelt werden kann, von der jedoch nur ein sehr geringer Teil direkt in der Erklärungsfindung verwendbar ist. Diesen relevanten Teil können wir wiederum in zwei Kategorien einteilen:

- Direkt verwertbare Information:
  Dieser Klasse ist jegliche Information zuzuordnen, die die Eigenschaft besitzt, direkt im Erklärungsprozess verwendet werden zu können, da sie zu Rückschlüssen auf den gewünschten formalen oder inhaltlichen Aufbau einer Erklärung befähigt. Ein Beispiel für eine solche Information wäre „Der Nutzer möchte kurze Erklärungen erhalten."
- Indirekt verwertbare Information:
  Hierzu gehören Informationen, die erkennbar relevant für die Planerklärung sind, für sich genommen jedoch nicht genug Aussagekraft besitzen um direkten Einfluss auf die Erklärungsfindung zu üben. Solche Informationen können aber mit anderen kombiniert werden, um Wissen der ersten Kategorie abzuleiten. Um obigen Wunsch nach kurzen Erklärungen zu motivieren, könnte beispielsweise eine Information „Der Nutzer ist vergesslich" vorliegen.

Auch bei dieser Einordnung ist festzustellen, dass die zweite Klasse, abhängig von den auf sie angewendeten Schlussfolgerungsmechanismen zur Ableitung von Informationen der ersten Klasse, fast beliebig groß definiert werden kann. Daher soll auch ihre Betrachtung nicht Teil dieser Arbeit sein. Wir konzentrieren uns stattdessen darauf zu untersuchen, in welche Bereiche das direkt verwendbare Wissen über den Nutzer gegliedert werden kann, und wie sich dieses Wissen schließlich zur Erklärungsfindung einsetzen lässt.

Im folgenden Abschnitt wird zunächst allgemein dargelegt, welches Wissen für den Erklärungsprozess von Nutzen sein kann. Anschließend beschäftigen wir uns damit, wie dieses Wissen zur Defizit- und Wertdefinition in der Wissensbasis zu repräsentieren ist.

6.4.1 Verwertbarkeit von Information

Um zu ermitteln, welche Informationen über den Nutzer für die Erklärungserzeugung verwendet werden können, soll der Aufbau einer Erklärung als Motivation dienen. Diese besteht, wie in Kapitel 4.2 dargelegt wurde, aus Argumenten, die sich einerseits über ihre typabhängige intrinsische Information, andererseits über die Eigenschaften des Plans, die sie referenzieren, definieren.
Wir haben mit den *konstruktiven* sowie *funktionalen* zwei Grundtypen von Argumenten charakterisiert. Konstruktive Argumente leiten sich, wie erläutert wurde, prinzipiell aus dem Zustandekommen des Plans ab, und vermitteln daher auch entsprechende Informationen. Können wir nun auf irgendeine Weise darauf schließen, dass der Nutzer primär an diesem Zustandekommen des Plans interessiert ist, beispielsweise weil er dessen Korrektheit überprüfen möchte, sollte das Erklärungssystem bevorzugt auf diesen Typ von Argumenten zurückgreifen. Dagegen sollte eine Nachfrage, die von Unklarheiten in der Ausführung des Plans motiviert ist, eher mit einer Erläuterung der in ihm bestehenden kausalen Zusammenhängen, also mit Hilfe von funktionalen Argumenten, beantwortet werden. Besitzen wir also Informationen darüber, welchem dieser beiden Aspekte das Interesse des Nutzers gilt, ermöglicht uns dies eine bessere „Interpretation“ der Frage und damit auch die Erzeugung einer in ihrem formalen Aufbau besseren Antwort.

Betrachten wir die inhaltliche Zusammensetzung, so lässt sich auch hier die Kenntnis der Motivation, die hinter der Frage des Nutzers steht, als Einflussfaktor identifizieren. So kann der Nutzer an bestimmten Aspekten der Domäne besonders interessiert sein, oder aber gewisse Bereiche gerade ausblenden wollen. Die in der Smartphone-Domäne modellierte Vermittlung von Informationseinheiten lässt sich beispielsweise in deren Verwendung für unterschiedliche Aufgaben sowie die zu ihrer Übermittlung getätigte Kommunikation aufteilen, wobei ein Nutzer möglicherweise nur an einem dieser Bereiche interessiert ist. Daneben kann an den inhaltlichen Interessen des Nutzers auch die Abstraktionsebene ausgerichtet werden, auf der die Erklärung argumentieren soll. Beispielsweise könnte ein Nutzer als Erklärung eher eine Übersicht bevorzugen, so dass primitive Tasks der Domäne in einer Erklärung nicht verwendet werden sollten.

Weiterhin ist offensichtlich, dass eine Planerklärung, die auf dem Nutzer unbekannten Elementen aufbaut, unzufriedenstellend ausfallen muss, und damit entweder einen weiterhin uninformierten Nutzer zurücklässt oder zu weiteren Nachfragen führt. Mit dem Begriff „bekannt“ ist hier die Bedeutung eines Domänenelements gemeint, was ein gewisses Spektrum für diese Eigenschaft ergibt. Dies wird besonders deutlich an der Betrachtung eines abstrakten Tasks, den eine Domäne enthält. Dessen Bedeutung zu kennen kann heißen, zu wissen, welche Aufgaben mit ihm erfüllt werden können, also im Planungsterminus welche Nachbedingungen er hat. Es kann aber auch heißen, dass seine Vorbedingungen ebenfalls bekannt sind, und darüber hinaus nicht nur die expliziten Nachbedingungen, die der Task selbst be-

sitzt, sondern auch die seiner unterschiedlichen Implementierung. Es liegt in jedem Fall nahe, dass in einer Erklärung kein Planschritt referenziert werden sollte, dessen Bedeutung gänzlich unbekannt ist. Analog dazu können Kenntnisse davon, welche Bestandteile der Domäne dem Nutzer sicher bekannt sind, dazu genutzt werden, eine Erklärung zu finden, die sich gerade auf diese Elemente stützt. In beiden Fällen lässt sich die Information darüber, welche Teile der Domäne der Nutzer kennt, jedoch eventuell nicht nur direkt, sondern auch zur Ableitung weiteren Wissens verwenden. Gehören nämlich einige dieser Elemente dem selben übergeordneten Bereich der Domäne an, gilt ihre Bekanntheit bzw. Unbekanntheit möglicherweise ebenso für diesen gesamten Aspekt, so dass diese Klassifizierung auf den Aspekt übertragen und dessen Verwendung in einer Erklärung insgesamt entsprechend notiert werden könnte.

6.4.2 Verfügbarkeit von Information

Wir untersuchen nun, wie die im vorhergehenden Abschnitt ermittelten Informationen dem Erklärungssystem zur Verfügung gestellt werden können. Dabei ist zwischen zwei gegensätzlichen Ansätzen der Nutzermodellierung zu unterscheiden, nämlich einerseits einer *dynamischen*, andererseits einer *statischen* Herangehensweise.

Dynamische Modellierung

Bei der dynamischen Variante wird die Modellierung im Verlauf einer oder mehrerer Erklärungssitzungen zunehmend verfeinert, wodurch sich jedoch sofort eine wichtige Implikation bezüglich der Informationen, die auf diesem Weg erworben werden können, ergibt. Diese sind nämlich auf diejenigen Elemente beschränkt, die in den in diesen Sitzungen erzeugten Erklärungen enthalten waren. Domänenbestandteile, die noch nicht referenziert wurden, lassen sich auf diese Weise – zumindest direkt – ebensowenig bewerten wie nicht verwendete Arten von Argumenten. Eine statische, von Interaktion mit dem Nutzer im Rahmen der Planerklärung unabhängige Modellierung besitzt diesen Nachteil hingegen nicht.

Für eine dynamische Modellierung sind grundsätzlich zwei, auch in Kombination verwendbare Ansätze denkbar. Eine Möglichkeit besteht darin, das Erklärungssystem „lernen“ zu lassen, welche Erklärungen ein Nutzer für gut befindet, indem dieser Rückmeldung über ihm präsentierte Erklärungen liefert. Hierbei ergeben sich jedoch zwei Probleme. Einerseits hängt der Wert eines solchen Feedbacks entscheidend vom seinem Informationsgehalt ab. Um überhaupt Informationen auf diesem Weg extrahieren zu können ist daher mindestens eine getrennte Bewertung von Inhalt und Aufbau der Erklärung vorzunehmen. Die gewünschten formalen Eigenschaften einer Erklärung lassen sich dabei durch Lieferung von Extrembeispielen, also beispielsweise Erklärungen die nur auf funktionalen bzw. nur auf konstruktiven

Argumenten basieren, recht schnell eingrenzen. Mehr Schwierigkeiten bereitet der inhaltliche Aspekt, da hierfür der Umfang der Domäne ausschlaggebend ist, so dass sehr viel Spielraum bei der Gestaltung der Erklärung besteht und es entsprechend länger dauert, bis die gewünschten Inhalte identifiziert sind. Gleichzeitig sollte der Aufwand, den der Nutzer bei der Bildung einer informativen Rückmeldung betreiben muss, jedoch nicht die Funktion des Erklärungssystems beeinträchtigen. Hier muss also entweder ein passender Mittelweg gefunden, oder aber der nötige Umfang der Rückmeldungen mit zunehmendem Wissen des Systems reduziert werden.
Andererseits führt die Implementierung eines Lernprozesses dazu, dass das System zu Beginn eher uninformiert agiert und dabei potentiell schlechte Erklärungen liefert, was unserem grundsätzlichen Ziel widerspricht. Hinzu kommt, dass ein solcher Lernprozess parallel dazu auch auf Nutzerseite abläuft, denn dieser erweitert sein Wissen ja gerade durch den Erhalt von Erklärungen. Dies bewirkt, dass die selbe Erklärung zu unterschiedlichen Ausgabezeitpunkten unterschiedliche Bewertungen erhalten kann, und erschwert eine Umsetzung damit zusätzlich.

Ein anderer Ansatz der dynamischen Modellierung ergänzt das Erklärungssystem durch eine Komponente, die nicht von expliziter Kommunikation mit dem Nutzer abhängig ist, sondern selbstständig Informationen erfasst. Dies ist besonders hilfreich, wenn es sich bei der zu erklärenden Domäne um eine handelt, mit der der Nutzer interagiert, so dass die Aufrufe bestimmer Funktionen, die Nutzung zugehöriger Hilfssysteme etc. verfolgt werden können. Aus einem solchen Trackingprozess kann dann beispielsweise abgeleitet werden, dass der Nutzer mit der Bedienung der SMS-Funktionen seines Handys grundsätzlich vertraut ist, und Nachfragen zu diesem Bereich eher technischer Natur sind, so dass sich entsprechende Erklärungen auf einer detaillierten Ebene bewegen sollten.
Auf ähnliche Weise können auch aus der Benutzung des Erklärungssystems selbst Informationen gewonnen werden. Durch eine Auswertung der gestellten Fragen und der darauf präsentierten Antworten lässt sich feststellen, welche Zusammenhänge dem Nutzer bereits bekannt sind, und daher in weiteren Erklärungen bevorzugt zu verwenden sind. Die Bedeutung einer solchen Analyse wächst noch deutlich an, wenn das System auch die Wissensvermittlung beherrscht, dem Nutzer also bislang unbekannte Elemente der Domäne erklären kann. Ein Nachteil dieser automatischen Informationssammlung ist allerdings, dass nur Wissen bezüglich der Domäneninhalte generiert werden kann, also keine Informationen über zu verwendende Argumente gewonnen werden können.

Statische Modellierung

Den beschriebenen Möglichkeiten, ein Nutzermodell dynamisch zu generieren und erweitern, steht der statische Ansatz gegenüber, bei dem das Modell schon vor einer Erklärungserzeugung bereitzustellen, die dazu benötigten Informationen also außerhalb des Erklärungsprozesses zu sammeln sind. Dazu können dem Nutzer vorab

Fragen gestellt werden, anhand derer ein Modell auf Basis einer Selbsteinschätzung des Nutzers erstellt werden kann. Eine solche Selbsteinschätzung ist jedoch aus zweierlei Gründen mit Vorsicht zu genießen. Einerseits ist der Nutzer nicht unbedingt ehrlich in der Beantwortung der Fragen. Dieses Problem könnte für diese Arbeit jedoch vernachlässigt werden, da es sich um ein System handelt, das dem Nutzer helfen soll, und er sich durch falsche Angaben keinen Vorteil verschaffen kann. Andererseits sind auch ehrliche Antworten kritisch zu betrachten, da diese immer noch nach der subjektiven Wahrnehmung des Nutzers ausfallen. Zudem kann eine Vielzahl von äußeren Faktoren die Durchführung einer solchen Nutzerbefragung beeinflussen, so dass dieser Weg insgesamt eher kritisch zu betrachten ist. Er bietet jedoch den entscheidenden Vorteil, dass bereits mit wenigen, auf zentrale Informationen abzielenden Fragen ein aussagekräftiges Nutzerprofil erstellt werden kann, und dieses Aufschluss über inhaltliche als auch formale Eigenschaften gewünschter Erklärungen geben kann.

Zum Abschluss der Betrachtung dieser Ansätze ist festzustellen, dass diese sich durchaus nicht gegenseitig ausschließen. So lässt sich der Nachteil einer dynamischen Nutzermodellierung, zu Beginn uninformiert zu sein und daher bei der Erklärungsfindung wenig zielgerichtet vorzugehen, durch die Bereitstellung eines Basis-Nutzermodells ausgleichen. Auf ähnliche Weise profitiert ein statisches Modell davon, über ausgegebene Erklärungen informiert zu werden und auf diesem Weg aus der praktischen Anwendung weitere Informationen gewinnen zu können, die nicht durch eine subjektive Betrachtungsweise des Nutzers beeinträchtigt sind.

6.4.3 Repräsentation des Nutzermodells

Wir konnten in den letzten Abschnitten feststellen, dass sich Anforderungen des Nutzers auf alle Bestandteile von Erklärungen, und dabei auf deren inhaltlichen wie auch formalen Aufbau beziehen können. Es sind zum einen Defizite an sämtlichen Elementen von Erklärungen denkbar, so dass sich aus Nutzereigenschaften Defizite aller Klassen ergeben können. Zum anderen können verschiedenste Präferenzen auf Nutzerseite bestehen, weshalb auch die Definition von Werten an allen Teilen einer Erklärung zu erwarten sind. Diese Freiheit in der Modellierung entspricht auch dem gesteckten Ziel, den Erklärungsprozess so weit wie möglich auf die Bedürfnisse des Nutzers zuzuschneiden. Auf Grund dieser Freiheit ist es aber Sache jeder Anwendung unseres Frameworks, sinnvolle Definitionen abzuleiten, die die Anforderungen des Nutzers möglichst gut wiedergeben. Beispielsweise könnte, wenn sich in der Erfassung über den Nutzer herausstellt, dass dieser funktionale Argumente zur Erklärung von Plänen bevorzugt, ein entsprechender Wert im Nutzermodell definiert werden, der zu einer höheren Bewertung von entsprechend aufgebauten Erklärungen führt.

Bei den naheliegenden inhaltlichen Werten und Defiziten, die beschrieben wurden, sind zwei Grundausrichtungen zu erkennen. Es gibt einerseits allgemeine Anforde-

rungen an die in der Erklärung referenzierten Domänenelemente, wie beispielsweise, dass Erklärungen ohne primitive Tasks zu bevorzugen sind. Zum Ausdruck dieser Anforderungen durch Werte oder Defizite ist kein weiteres Wissen über die Domäne nötig, da nur auf ihre grundlegenden Klassen Bezug genommen wird. Andererseits können Anforderungen auch konkret die einem Plan zu Grunde liegende Domäne betreffen, wie beispielsweise die Definition von Defiziten in der Verwendung unbekannter Domänenelemente. Um diese ausdrücken zu können ist es notwendig, dass zur Erstellung des Nutzermodells auch eine Repräsentation der Domäne verfügbar ist, an welcher dann zum Beispiel annotiert werden kann, welche Tasks dem Nutzer bekannt sind. Unter anderem von dieser Notwendigkeit ist es auch motiviert, dass wir in dieser Arbeit einen rein statischen Ansatz zur Nutzermodellierung verfolgen. Welche weiteren Möglichkeiten eine dynamische Nutzermodellierung eröffnet ist eines der Themen, mit dem wir uns im Ausblick der Arbeit auseinandersetzen.

6.5 Wissen über die Schnittstelle

Neben den Präferenzen des Nutzers selbst kann auch Wissen über die Schnittstelle, die zur Präsentation der für ihn angepassten Erklärung dient, zur Entwicklung adäquater Erklärungen eingesetzt werden. Entscheidend ist hierbei, dass unterschiedliche Darstellungsmodi unterschiedlichen Beschränkungen unterworfen sind, was die Ausgabe der Erklärung betrifft. Solche Beschränkungen können beispielsweise technischer Natur sein. In Kapitel 4.3.2 wurde bereits ein Beispiel einer solchen Beschränkung genannt, dass nämlich eine Planerklärung bei einer grafischen Ausgabe auf dem Display eines Desktop-PCs ganz andere Möglichkeiten bietet als auf dem Display eines Mobiltelefons. Aus diesen Randbedingungen können sich Einschränkungen beispielsweise für die Anzahl darstellbarer Argumente ergeben, aber auch für die Typen von Argumenten, die verwendet werden können.

Die Modalität der Erklärungsausgabe stellt einen ähnlichen Faktor dar, der bei der Auswahl der Erklärung eine Rolle spielen kann. Eine Darstellung der Erklärung als Graph, wie sie auch in dieser Arbeit verwendet wird, steht dem Nutzer beispielsweise beliebig lang zur Verfügung, so dass er dahingehend keinen äußeren Einflüssen in seiner Erklärungsrezeption unterliegt. Eine Verbalisierung der Erklärung durch Sprachausgabe hingegen bringt mit sich, dass die Aufnahme der Erklärung durch den Nutzer nicht von ihm, sondern durch die Ausgabemodalität gesteuert wird. Denkbar ist dabei auch, dass eine Erklärung in mehreren Teilen ausgegeben wird, die in ihrem Umfang entsprechend angepasst sind.

Eine weitere Quelle von Einschränkungen in der Darstellbarkeit kann das Ziel der Anwendung selbst sein. So ist es denkbar, dass die Anwendung auf die Erklärung von Plänen einer bestimmten Domäne, und darin auf einen speziellen Bereich ausgerichtet ist, und beispielsweise nur primitive Tasks darstellt. Die Bewertung der

verfügbaren Erklärungen würde, wenn solche Beschränkungen der Darstellung nicht modelliert würden, möglicherweise verfälscht, da sie Werte einbeziehen würde, die letztlich von der Schnittstelle gar nicht präsentiert würden, und daher vielleicht zur Auswahl einer nicht optimalen Erklärung führen.

6.5.1 Repräsentation der Schnittstelle

In den meisten Fällen sind die Anforderungen, die sich aus Eigenschaften der Schnittstelle ergeben, in Form von Defiziten zu repräsentieren. Die Gründe dafür sind einerseits, dass Erklärungsteile nicht „ein bisschen“ dargestellt werden können, sondern entweder dargestellt werden oder nicht, was der vorgeschlagenen Behandlung von Defiziten entspricht. Andererseits wäre bei einer Definition von Werten an dieser Stelle zu gewährleisten, dass diese in einer angemessenen Beziehung zu denen stehen, die von anderer Quelle, insbesondere dem Nutzer, festgelegt werden. Es ist allerdings denkbar, dass eine Anwendung die vom Nutzermodell zu definierenden Werte einschränkt, um dies zu erreichen. Unabhängig davon sind die Bereiche einer Erklärung, die von Anforderungen der Schnittstelle eingeschlossen werden, ebenso weit zu fassen wie die des Nutzermodells, da sowohl Anforderungen an die formale Gestaltung der Erklärung als auch an die darzustellenden Inhalte bestehen können. Wie auch beim Nutzermodell hat daher ein Entwickler einer auf unserem Framework aufbauenden Anwendung zur Planerklärung die Aufgabe, aus den gegebenen Eigenschaften der Darstellung entsprechende Defizit- oder Wertdefinitionen abzuleiten. Soll zum Beispiel eine zeilenweise Ausgabe der Erklärungen in einem Display mit vier Textzeilen realisiert werden, könnte, um dazu passende Erklärungen zu erhalten, eine Definition eines Überlange-Erklärung-Defizits für Erklärungen mit mehr als vier Argumenten in die Repräsentation der Schnittstelle aufgenommen werden.

Eine andere Möglichkeit, Wissen über die Schnittstelle zu verwenden, wäre es, nicht die Einschränkungen einer bestimmten bereits gegebenen Schnittstelle abzubilden, sondern eine erwünschte Darstellungsweise zu modellieren, und dieses Modell dazu zu verwenden, eine entsprechende Schnittstelle zu generieren. Ein solcher modellgetriebener Ansatz liegt jedoch nicht im Fokus dieser Arbeit.

6.6 Wissen über die Domäne

Wissen über die Domäne, welches über ihren reinen Aufbau hinausgeht, kann eine wichtige Rolle bei der Steuerung der Erklärungsfindung spielen. In Kapitel 4.3.2 wurde festgestellt, dass es von von großer Bedeutung für die Qualität einer Erklärung ist, wie informativ ihre referenzierten Domänenelemente sind. Es hat sich

aber auch herausgestellt, dass dieser Informationsgehalt kaum allgemein bewertet werden kann. Dennoch können bei näherer Betrachtung zwei Aspekte Aufschluss darüber geben, ob ein Domänenelement für einen Nutzer informationstragend ist.

6.6.1 Technisch bedingte Domänenelemente

Da eine Domäne immer auf das zu Grunde liegende Planungssystem zugeschnitten ist, kann sie Bestandteile haben, die für die maschinelle Planerzeugung notwendig sind, die ein Mensch jedoch nicht betrachten würde, und die daher in einer Planerklärung nur wenig Information beitragen können. Dies manifestiert sich zum einen im Vorhandensein von Tasks, die sehr primitive, kleinteilige Schritte im Plan realisieren. So modelliert die Smartphone-Domäne die Möglichkeit, eine EMail mit Hilfe eines Smartphones zu versenden, was durch die in Abbildung 6.3, Seite 80 dargestellte Schrittfolge realisiert wird. Der primitive Task `press_EMail.NewEMail.Send()`, der das Drücken des „Senden“-Buttons im EMail-Client verkörpert, wird von einem Menschen, der auch nur ein geringes Maß an Erfahrung im Umgang mit Computersystemen hat, nie so explizit betrachtet werden. Die Notwendigkeit, den Versendevorgang durch eine solche Bestätigung zu starten, ist dem Nutzer klar, und wird daher implizit eingeplant. Die Verwendung dieses Tasks in einer Erklärung wäre daher, um den zentralen Kriterien der Kürze und Prägnanz zu genügen, eher schädlich als hilfreich, da andere an seiner Stelle verwendete Tasks dem Nutzer sehr wahrscheinlich mehr Information vermitteln könnten.

Eine weitere Konsequenz der Modellierung für ein Planungssystem ist es, dass die Domäne Bestandteile einschließt, die für den Nutzer unsichtbare Zusammenhänge realisieren. Abbildung 6.4 (S. 80) zeigt, dass für den vom Nutzer intuitiv als primitive Aktion interpretierten Task `press_Home()`, der die Rückkehr ins Hauptmenü des Mobiltelefons durch einen einfachen Tastendruck bezeichnet, in der Domäne tatsächlich eine Vielzahl von Implementierungen existieren. Diese sind notwendig um, je nachdem an welcher Stelle des Menüs die Taste gedrückt wird, unterschiedliche Effekte umzusetzen, die den internen Zustand des Systems verkörpern. Da diese Effekte aus Nutzersicht jedoch nicht existieren, sollte in einer Erklärung keine der primitiven Implementierungen, sondern der übergeordnete Task `press_Home()` verwendet werden.

6.6.2 Selbsterklärende Domänenelemente

Während technisch bedingte Domänenelemente zum Verständnis eines Plans kaum beitragen können, gibt es auch solche Elemente, die durch ihre Eigenschaft, gewissermaßen selbsterklärend zu sein, prinzipiell zur Verwendung in Erklärungen geeignet sind. Der Begriff „selbsterklärend“ bedeutet im Zusammenhang der Planerklärung, dass der Nutzer die Bedeutung solcher Domänenelemente für den Plan auch erfassen

kann, ohne vorher mit diesen vertraut gewesen zu sein, und insbesondere auch ohne ihre Details zu kennen. Als solche selbsterklärenden Elemente können beispielsweise die Tasks der Smartphone-Domäne angesehen werden, die zum Wechsel in die unterschiedlichen Modi dienen, zu denen unter anderem der Task `enterMode_Home()` gehört, mit dem, wie auch dieser im Domänenmodell verwendete Name schon nahelegt, in das Hauptmenü des Smartphones zurückgekehrt werden kann. An den anderen Tasks zum Moduswechsel treten allerdings auch wieder die Grenzen einer allgemeinen, nutzerunabhängigen Bewertung von Domänenbestandteilen zutage, denn auch das beschriebene Selbstverständnis ist von einer gewissen Grundkenntnis der zu erklärenden Domäne abhängig. Ist etwa ein Nutzer nicht mit der Benutzung solcher Menüstrukturen vertraut, dann wird sich ihm die Bedeutung dieser Tasks nicht auf die hier beschriebene Weise erschließen.

Schließlich ist festzustellen, dass zwischen selbsterklärenden und technischen Domänenbestandteilen Überschneidungen durchaus denkbar sind. Dies liegt nahe, da, wie im vorangegangenen Abschnitt beschrieben wurde, gerade sehr grundlegende Funktionen zu den technischen Bestandteilen einer Domäne zählen können, die auch entsprechend einfach zu verstehen sein können. Tatsächlich stellt der oben als Beispiel verwendete Task `press_EMail.NewEMail.Send()` einen solchen Schnittpunkt zwischen „zu primitiv und technisch für die Erklärung“, und „immer verwendbar, weil selbsterklärend“ dar. Aus dieser Sichtweise ergibt sich aber auch schon, dass dem Aspekt, dass der Task zu technisch ist, um verwendet zu werden, in der Praxis die höhere Bedeutung zugemessen werden muss: Die Eigenschaft der Selbsterklärung bedeutet, dass die Verwendung eines entsprechenden Elements auch ohne eine vorab gegebene Kenntnis seiner Funktion Information übermitteln kann, eben weil diese Funktion intuitiv erfassbar ist. Technische Bestandteile besitzen aber gerade den Nachteil, dass nur sehr wenig Information übermittelt werden kann.

6.6.3 Repräsentation von Wissen über die Domäne

Zusätzliches Wissen über die Domäne, beispielsweise die Kenntnis der in den letzten Abschnitten diskutierten Eigenschaften, kann, wie gezeigt wurde, hilfreich bei der Erklärungssuche sein. Es ist aber auch festzuhalten, dass sich aus diesem Wissen keine festen Anforderungen an die Erklärung ergeben. Stattdessen ist solches Wissen als Erweiterung der Möglichkeiten zu sehen, die sich bei der Spezifikation von Anforderungen aus dem Nutzermodell und der Schnittstelle bieten; Die Repräsentation des Domänenmodells wird daher um zwei Relationen `istTechnisch` und `istSelbsterklärend` erweitert, mit denen die entsprechenden Eigenschaften angegeben werden können. So könnte beispielsweise ein Planschritt-Defizit, das sich aus der Referenzierung unbekannter Tasks ergibt, dahingehend erweitert werden, dass der Task weder bekannt noch selbsterklärend, aber auch nicht technisch sein darf.

Im Sinne dieser Verfeinerung der zu definierenden Anforderungen wird auch ein weiteres Konzept eingeführt, nämlich der `Domänenaspekt`. Sowohl bei der Betrachtung von relevantem Wissen über den Nutzer als auch über die Benutzerschnittstelle hat sich ergeben, dass bestimmte inhaltliche Bereiche einer Domäne für eine konkrete Anwendung wichtiger sein können als andere. Das genannte Konzept sowie eine Relation `gehörtZu`, mit der die Zugehörigkeit der Domänenelemente angegeben werden kann (wobei ein Element auch mehreren Aspekten zugerechnet werden kann), können dazu verwendet werden, solche inhaltlichen Ausrichtungen zu modellieren. Auf diese Weise können inhaltliche Zusammenhänge in der Domäne explizit gemacht werden, die bislang nur durch die gegenseitige Referenzierung der Domänenbestandteile erkennbar waren. Beispielsweise lässt sich die Smartphone-Domäne so in einen Bereich gliedern, der das Verpacken und Entpacken von Informationen, und einen, welcher das Übermitteln dieser Informationen betrifft. Abbildung 6.5 auf Seite 81 zeigt die Repräsentation einer `PANDA`-Domäne einschließlich der hier beschriebenen Erweiterungen.

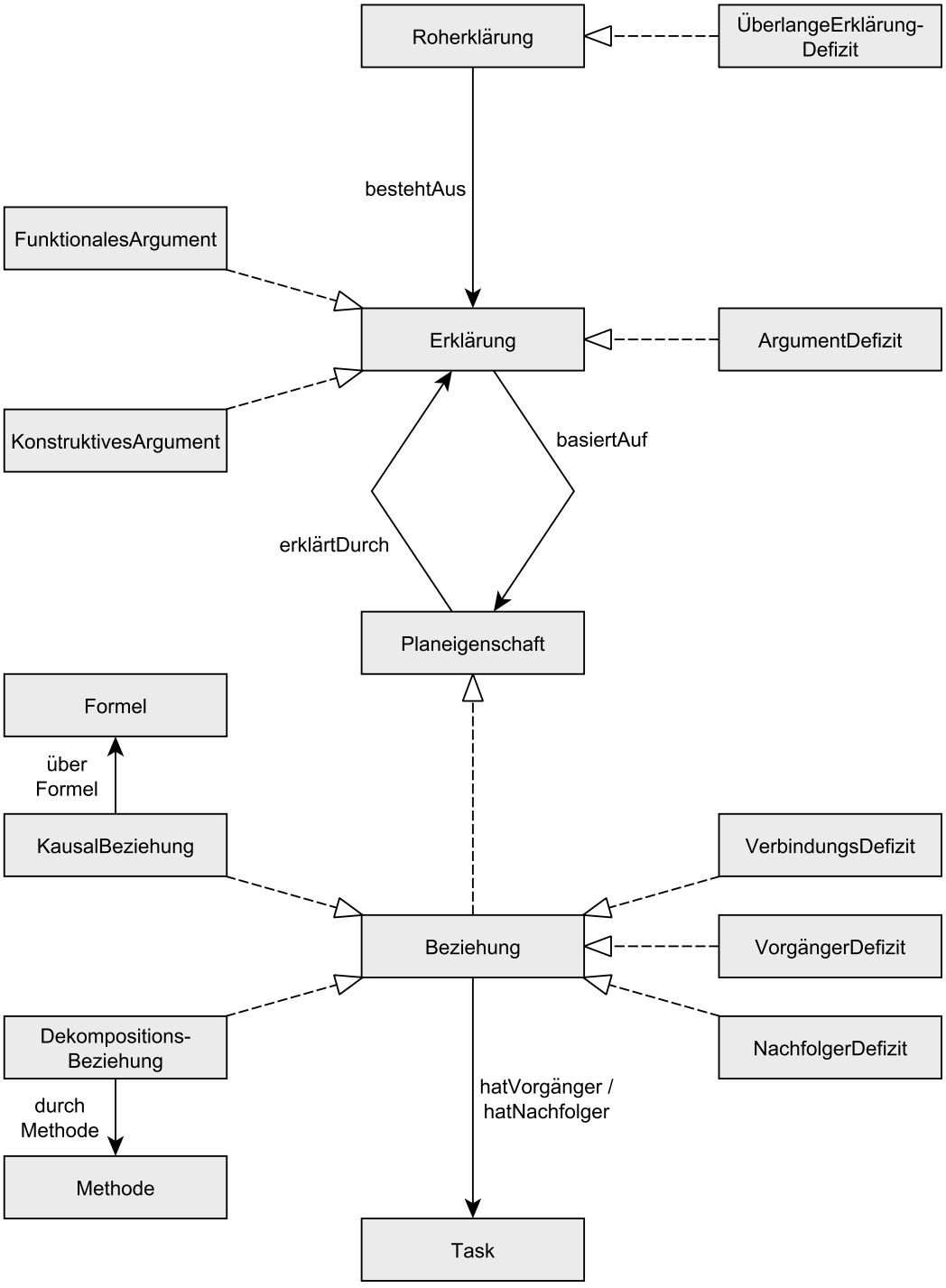


Abbildung 6.2: Repräsentation von Roherklärungen, Argumenten und Defizitklassen

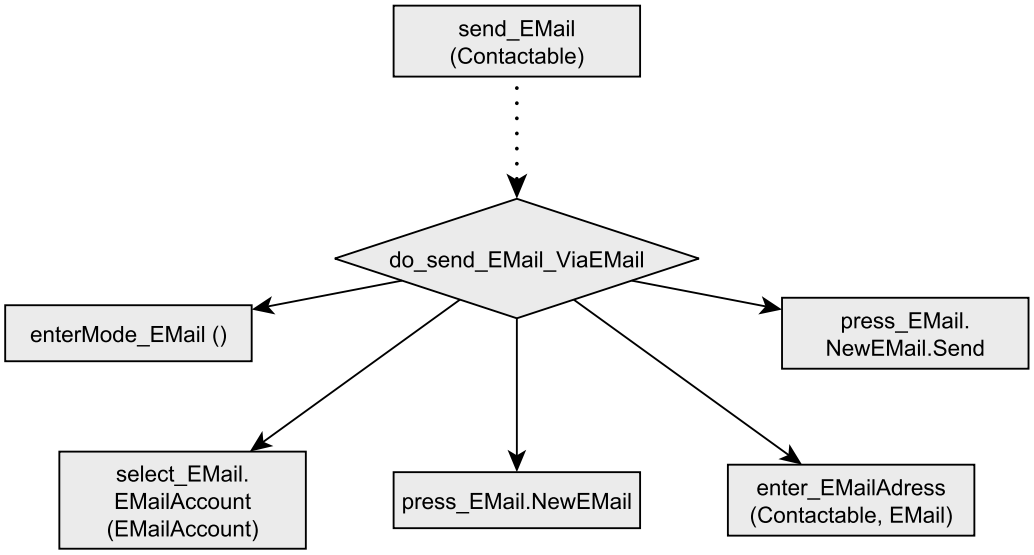


Abbildung 6.3: Eine Implementierung des Tasks send_EMail(Contactable)

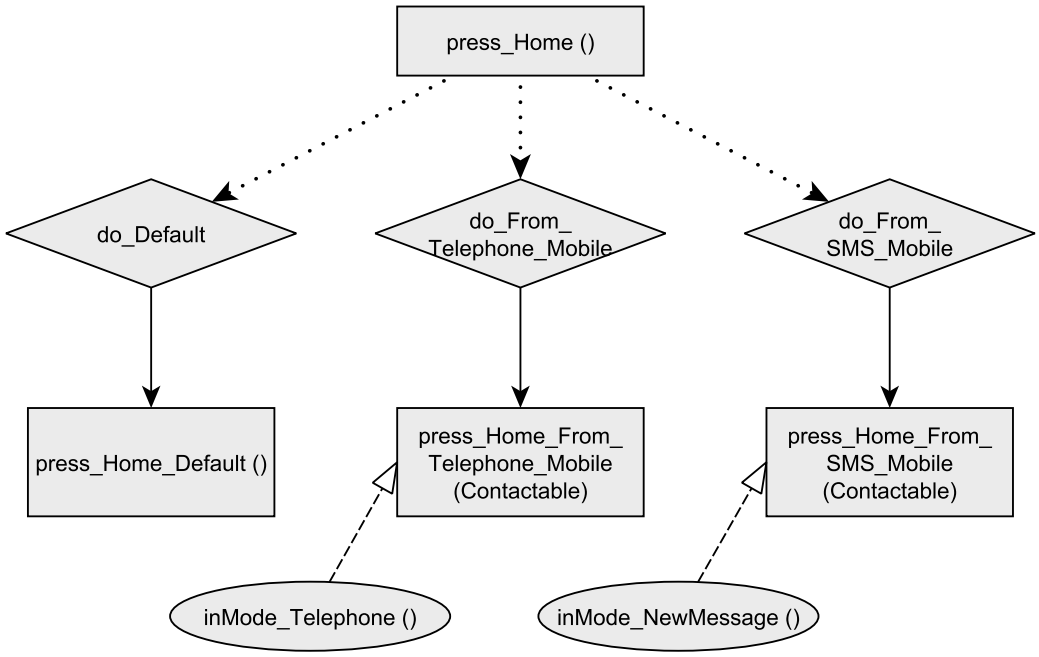


Abbildung 6.4: Einige Implementierungen des Tasks press_Home().
(Aus Darstellungsgründen werden die Methodenbezeichnungen verkürzt dargestellt)

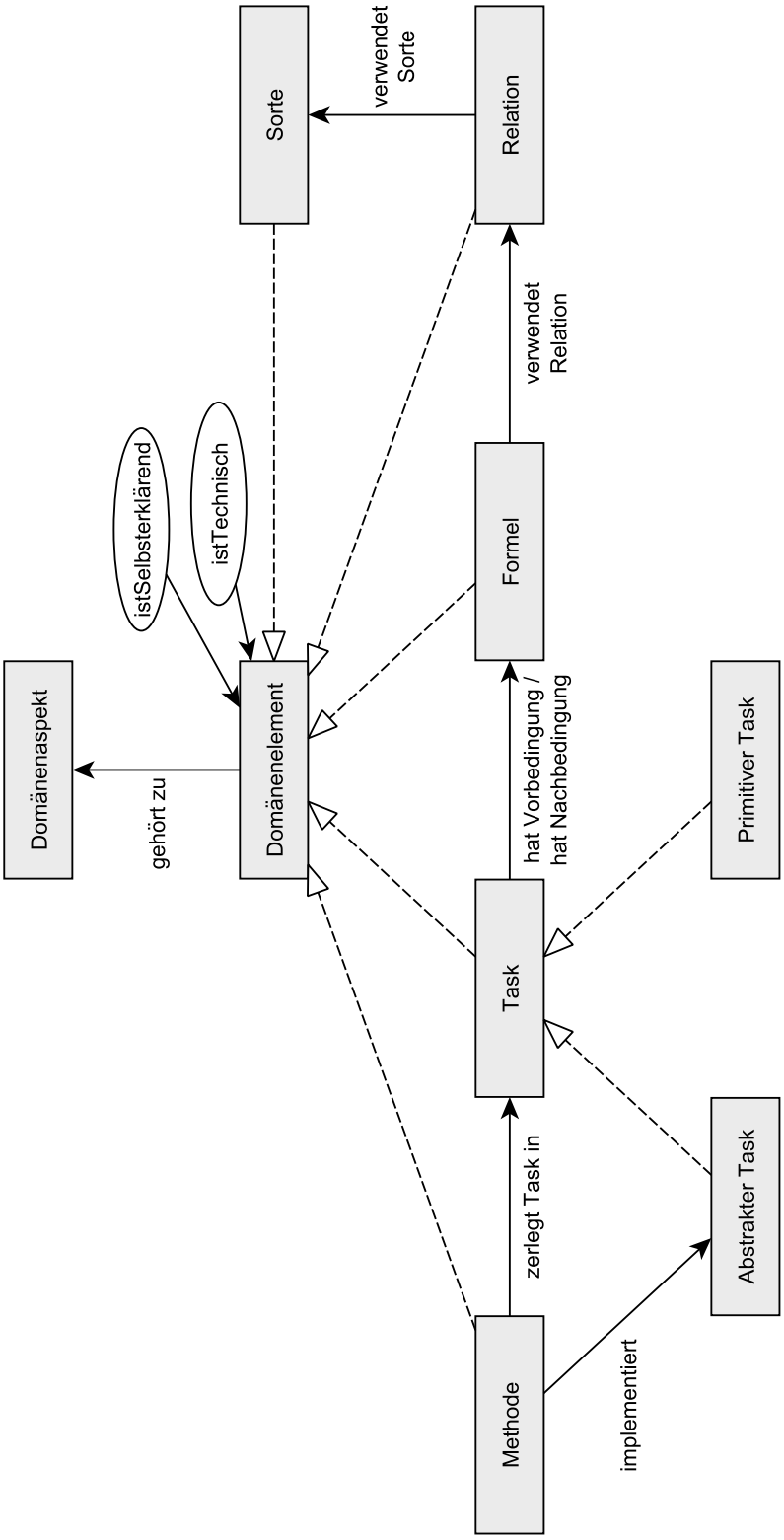


Abbildung 6.5: Repräsentation der Domäne

7 Prototypische Implementierung des Frameworks

Der in dieser Arbeit entwickelte Ansatz für eine nutzerorientierte Planerklärung wurde auch in einer prototypischen Implementierung umgesetzt. Dabei ist zum einen ein Framework entstanden, welches die beschriebenen Abläufe realisiert, zum anderen eine einfache Beispielanwendung, in der Pläne der Smartphone-Domäne auf Textbasis erklärt werden. Auf Grund der starken Ausrichtung auf das in JAVA implementierte PANDA-Planungssystem wurde dabei ebenfalls JAVA als Programmiersprache verwendet. Auf den nächsten Seiten soll ein Überblick über den Aufbau und die Funktionsweise der Implementierung des Frameworks gegeben werden. Die entstandene Beispielanwendung wird im Kapitel 7.6 vorgestellt.

Die Implementierung des Frameworks ist vor allem durch die Verwendung einer wissensbasierten Definition von Defiziten und Werten geprägt, wie sie im Ansatz beschrieben wurde. Die Form der Repräsentation dieser Wissensbasis war daher eine zentrale Frage bei der Implementierung, wobei wir uns aus mehreren Gründen für eine Modellierung in OWL entschieden haben. Ausschlaggebend für diese Wahl war, dass Bibliotheken für JAVA frei verfügbar sind, mit denen durch entsprechenden Code OWL-Ontologien erzeugt, modifiziert und untersucht werden können, und es eine Reihe von Implementierungen der Schlussfolgerungsmechanismen gibt, die wir zur Umsetzung unseres Ansatzes benötigen. Daneben hat auch eine Rolle gespielt, dass recht komfortable Editoren existieren, mit denen OWL-Ontologien erzeugt werden können, was nicht nur bei der Entwicklung des Frameworks selbst, sondern auch bei der Modellierung von zusätzlichem Wissen über Domänen sowie auf dem Framework basierenden Anwendungen hilfreich ist, da diese ja ein Nutzer- und Schnittstellenmodell zur Verfügung stellen müssen.
Wegen der zentralen Rolle der Wissensrepräsentation in unserem Framework dienen einige Teile der Implementierung zur Repräsentation von Domänen und Erklärungen. Neben der Erzeugung dieser Repräsentationen in OWL waren Möglichkeiten zu finden, diese zur Laufzeit der Erklärungsanwendung zu einer Wissensbasis zu vereinen, und darauf die Detektion von Defiziten und die Erklärungsbewertung zu realisieren.

Wir geben zunächst einen allgemeinen Überblick darüber, wie der Erklärungsprozess umgesetzt wurde (Abschnitt 7.1), und welche Bibliotheken dabei verwendet wurden (Abschnitt 7.2). Es folgt in Abschnitt 7.3 eine Beschreibung der Klassen,

die zur Interaktion des Systems mit der Wissensbasis entwickelt wurden. Wie Defizite und Werte von Anwendungen unserer Implementierung zu definieren sind, wird in Abschnitt 7.4 erläutert. Abschnitt 7.5 gibt schließlich einen genaueren Einblick in den Ablauf der einzelnen Schritte des Erklärungsprozesses.

7.1 Überblick über die Implementierung des Erklärungsprozesses

Anwendungen steht zur Umsetzung einer Planerklärungsfunktion die Klasse `Explainer` unseres Frameworks zur Verfügung. Diese bietet eine Methode zum Absetzen von Anfragen, die eine durch Verwendung eines Nutzer- und Schnittstellen-Modells sowie zusätzlichen Wissens über die Domäne für die Anwendung angepasste Erklärung generiert und zurückgibt. Intern wird dies so umgesetzt, dass unter Einsatz der Erklärbär-Bibliothek zunächst eine Menge von Roherklärungen erzeugt wird. Diese werden durch Generierung von entsprechenden OWL-Repräsentationen auf Defizite untersucht und von diesen befreit. Die nach dieser Operation defizitfreien Erklärungen werden dann für ihre Bewertung nochmals in die Wissensbasis eingespeist, so dass für jede Erklärung eine Bewertung erstellt werden kann. Die gemäß dieser Bewertung beste Erklärung wird schließlich an die Anwendung zurückgegeben. Dieser Ablauf wird in Abbildung 7.1 noch einmal dargestellt.

7.2 Verwendete Bibliotheken

Die Implementierung verwendet zur Umsetzung des Erklärungsprozesses mehrere externe Bibliotheken, die zur Interaktion mit einer Wissensbasis, zur Erzeugung von Roherklärungen sowie der Anbindung an das `PANDA`-Planungssystem benötigt werden, und hier kurz vorgestellt werden sollen.

OWL-API

Die OWL-API ist eine vornehmlich von der Universität Manchester entwickelte Bibliothek, die eine JAVA-Schnittstelle bietet, um die Interaktion mit OWL-Ontologien zu ermöglichen. Sie enthält sowohl Management-Klassen, mit denen solche Ontologien geladen, erzeugt, kombiniert und gespeichert werden können, als auch Klassen, mit denen lesender wie auch schreibender Zugriff auf die Inhalte von Ontologien realisiert werden kann, was an mehreren Stellen dieser Implementierung benötigt wurde.

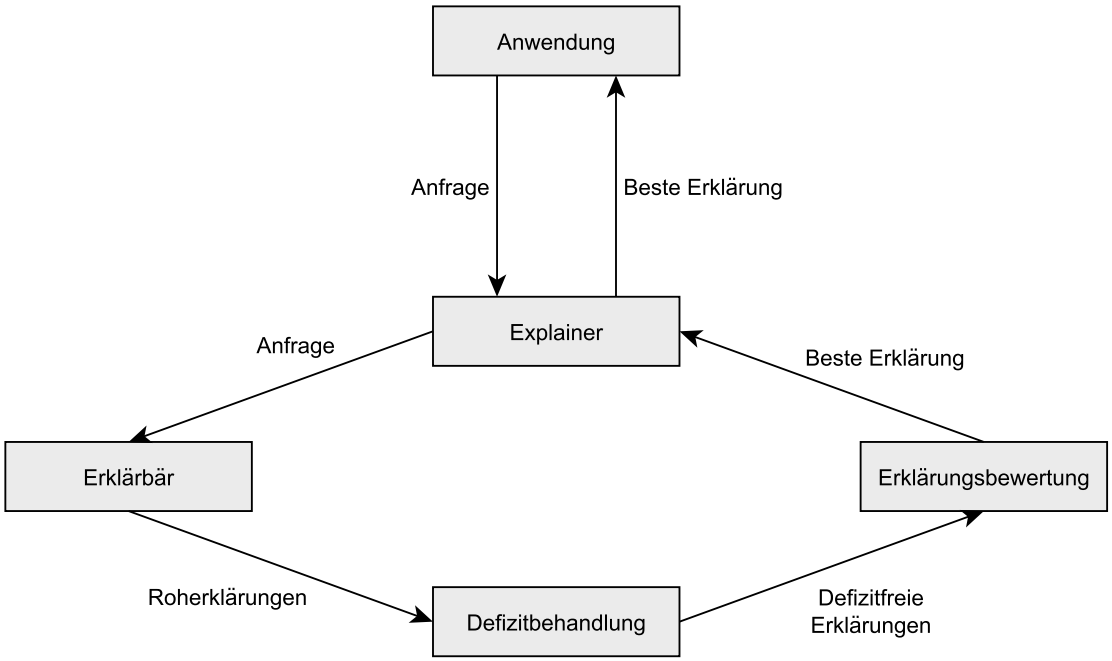


Abbildung 7.1: Ein Überblick über den Ablauf des Erklärungsprozesses

Pellet-Reasoner

Pellet ist ein sogenannter OWL *Reasoner*, das heißt er bietet eine Implementierung der Schlussfolgerungsmechanismen auf OWL-Ontologien, die im Grundlagenkapitel vorgestellt wurden. Für unsere Arbeit ist dabei besonders die Klassifikation von Individuen wichtig, die wir zur Detektion von Defiziten und der Bewertung von Erklärungen einsetzen.

PANDA

Da der vorgestellte Ansatz sich auf die Erklärung von Plänen des `PANDA`-Planungssystems bezieht, werden auch die Klassen der JAVA-Implementierung von `PANDA` in unserer Implementierung verwendet. Wir benötigen davon zum einen die Klassen, mit denen Domänen aus dem XML-Format eingelesen und dann in der Anwendung verkörpert werden, da wir eine OWL-Repräsentation von Domänen generieren müssen. Zum anderen werden die Klassen zur Darstellung von Plänen verwendet, sowohl zur Erzeugung einer Repräsentation der Pläne zur Laufzeit der Anwendung,

als auch zum Transport von Plänen zwischen einer Anwendung, unserem Framework und dem System zur Erzeugung von Roherklärungen.

Erklärbär

Die Bibliothek Erklärbär ist eine JAVA-Implementierung des in Kapitel 5.1 beschriebenen Ansatzes zur Erzeugung von Roherklärungen, und bildet daher neben `PANDA` die zweite Grundlage unserer Implementierung. Entsprechend verwenden wir die darin enthaltenen Klassen, um zur Beantwortung der von Anwendungen eingehenden Anfragen Roherklärungen zu generieren, die wir anschließend modifizieren und bewerten.

7.3 Erzeugung von OWL-Repräsentationen

7.3.1 Repräsentation von Domänen

Damit zusätzliches, über den formalen Aufbau von `PANDA`-Domänen hinausgehendes Wissen ausgedrückt und dem Erklärungssystem zur Verfügung gestellt werden kann, ist es notwendig, eine Abbildung dieser Domänen in Form einer OWL-Ontologie zu erstellen. Diese Funktionalität bietet die Klasse `PandaDomain2OWLBuilder`. Sie stellt dazu eine Main-Methode zur Verfügung, die als Parameter den Dateipfad zu einer abzubildenden Domäne im für das `PANDA`-System üblichen XML-Format und einen Pfad zum gewünschten Ausgabeort annimmt. Die OWL-Version der Domäne wird allerdings nicht direkt aus dem XML-Format generiert, sondern aus einem, mit Hilfe der Parser-Klasse zum Einlesen von Domänen aus dem `PANDA`-System erzeugten, entsprechenden JAVA-Objekt. Dieser Schritt erspart uns die erneute Analyse der Struktur der Domäne und der in ihr herrschenden Beziehungen. Intern wird dann aus Sorten, Relationen, Tasks und Methoden der Domäne eine OWL-Repräsentation, wie sie in 6.2.1 beschrieben wurde, aufgebaut. Die resultierende OWL-Repräsentation der Domäne kann dann zum einen um zusätzliche Informationen über die Domäne ergänzt werden, die im Erklärungsprozess verwendet werden können, zum anderen kann in der Nutzermodellierung Bezug auf die enthaltenen Bestandteile genommen werden, um beispielsweise auszudrücken, dass einem Nutzer gewisse Teile der Domäne bekannt sind.

7.3.2 Repräsentation von Erklärungen

Um Erklärungen in einer OWL-Ontologie abzubilden dient eine Klasse `OWLExplanationBuilder`. Die Abbildung wird durch eine rekursiv arbeitende

Methode `buildExplanation` realisiert, die als Rückgabe eine Menge von Axiomen liefert, die insgesamt die Erklärung in OWL-Form wiedergeben. Die Methode wird zunächst mit der `ExplanationBox` aufgerufen, die eine Erklärung als Ganzes kapselt. Innerhalb der Methode werden Axiome erzeugt, die Individuen und Relationen des gerade betrachteten Objekts darstellen, beim initialen Aufruf zum Beispiel also eine `RawExplanation`. Die zurückzugebenden Axiome werden dann durch rekursive Aufrufe von `buildExplanation` um diejenigen ergänzt, die für die Darstellung der weiteren Elemente der Erklärung notwendig sind, so dass letztlich ein Aufbau entsteht, wie er in Kapitel 6.2 vorgeschlagen wurde. Da die Defizitdetektion und Erklärungsbewertung, wie später noch genauer beschrieben wird, Referenzen auf bestimmte Bestandteile der OWL-Repräsentation benötigen (zur Bewertung beispielsweise auf alle Argumente), werden diese zusätzlich in eigenen Datenstrukturen abgelegt, auf die mit entsprechenden Methoden Zugriff gewährt wird.

7.4 Definition von Werten und Defiziten in der Implementierung

Sowohl bei der Definition von Defiziten als auch von Werten wurde in der Implementierung eine leichte Abwandlung in der Wissensrepräsentation vorgenommen. Um die Defizitdetektion und Erklärungsbewertung zu vereinfachen, die diese Definitionen verwenden, werden Defizite und Werte nicht als Subklassen der betreffenden Erklärungsbestandteile modelliert. Stattdessen wurden in der Wissensbasis Klassen angelegt, die die Definition von Defiziten und Werten außerhalb der Klassenhierarchie von Erklärungen bündeln. Unserem Ansatz entsprechend gibt es also zur Definition von Werten zwei Klassen `RawExplanationValues` für Roherklärungen sowie `ArgumentValues` für ihre Argumente, zu denen von der Anwendungsseite Subklassen zu definieren sind, die Eigenschaften mit Werten repräsentieren. Die genaue Höhe des jeweiligen Werts ist durch eine entsprechende Annotation der Subklasse anzugeben. Die Repräsentation von Defiziten erfolgt analog dazu durch entsprechende Klassen, die die vier behandelten Basisfälle von Defiziten verkörpern.

7.5 Die Umsetzung des Erklärungsprozesses im Detail

7.5.1 Die Explainer-Klasse

Die Schnittstelle für auf dem Framework basierende Anwendungen stellt wie im Überblick bereits beschrieben wurde die Klasse `Explainer` dar. Auf Grund dieser Schnittstellenfunktion ist hier auch die Konfiguration unseres Frameworks mit den

Nutzer- und Schnittstellenmodellen sowie der Domäne der zu erklärenden Pläne vorzunehmen. Dies geschieht bei der Instanziierung eines Objekts der Klasse, indem Pfade zu den OWL-Ontologien, die die genannten Modelle sowie das zusätzliche Wissen über die Domäne enthalten, als Parameter übergeben werden.

Intern werden bei der Instanziierung als erstes diese Ontologien geladen, wobei sie für späteren erneuten Zugriff, wenn ein `Explainer` mit einer teilweise geänderten Konfiguration benötigt wird, in einen Cache eingefügt werden. Die geladenen Ontologien werden anschließend zu einer großen Wissensbasis kombiniert, die damit alle Definitionen von Defiziten und Werten, sowie die Informationen über die Domäne beinhaltet. Um die im Verlauf der Erklärungserzeugung benötigte Klassifikation durchführen zu können, wird zusätzlich ein OWL-Reasoner erzeugt.

Mit diesen Schritten ist die Grundkonfiguration des Explainers abgeschlossen, so dass nun die Objekte erzeugt werden können, die die Erzeugung der Roherklärungen, die Defizitbehandlung sowie die Erklärungsbewertung und -auswahl umsetzen. Während zu ersterem eine statische Instanz des `BackwardSearchExplanationManager` aus der Erklärbär-Bibliothek dient, wird die Defizitbehandlung durch einen `FlawedExplanationHandler` und die Bewertung durch einen `ExplanationEvaluationHandler` übernommen. Diese benötigen Referenzen auf die Ontologie, den Reasoner sowie das Domänenmodell, weshalb diese Referenzen in Form einer Instanz der Klasse `ExplanationConfiguration` gebündelt übergeben werden. Der Explainer ist damit bereit, eine Anfrage anzunehmen.

7.5.2 Annahme einer Anfrage

Zur Anforderung einer Erklärung durch eine Anwendung dient die Methode `findExplanationFor`, die den zu erklärenden Plan und die zu erklärende Planeigenschaft, repräsentiert durch ein Objekt der `Information`-Klasse aus der Erklärbär-Bibliothek, als Parameter benötigt. Nachdem der `BackwardSearchExplanationManager` mit diesem Plan konfiguriert wurde, liefert er nach Aufruf seiner Methode `generateExplanations` mit der `Information` als Parameter eine Menge von Objekten der Klasse `ExplanationBox` zurück. Diese werden zur Defizitbehandlung und anschließenden Erklärungsbewertung an zu diesem Zweck von uns entwickelte Klassen weitergegeben, deren Umsetzung in den nächsten Abschnitten genauer beschrieben wird. Das Ergebnis der Behandlung der Anfrage und damit das Objekt zur Rückgabe stellt schließlich eine einzelne `ExplanationBox` dar.

7.5.3 Defizitbehandlung

Zur Behandlung von Defiziten wurde eine Klasse `FlawedExplanationHandler` implementiert. Ihre Schnittstelle nach außen stellt eine Methode `handle` dar, die eine Menge von Erklärungen als Parameter annimmt, und auch wieder eine Menge von Erklärungen zurückgibt. Die Behandlung der Defizite einer Erklärung erfolgt durch eine Schleife, in deren Verlauf die Erklärung immer wieder auf Defizite untersucht, aus der Anwendung jeder Modifikation zur Behebung jedes gegebenen Defizits ein Nachfolger der Erklärung generiert, und aus diesen der Beste zur weiteren Bearbeitung ausgewählt wird. Dieser Vorgang wird so lange wiederholt, bis eine defizitfreie Version gefunden wurde. Wurde jede der eingehenden Roherklärungen so behandelt, stehen am Ende dieses Schritts eine Menge von Erklärungen, die frei von den durch die Anwendung definierten Defiziten sind. Abbildung 7.2 gibt einen Überblick über die Klassen, die an der Defizitbehandlung beteiligt sind.

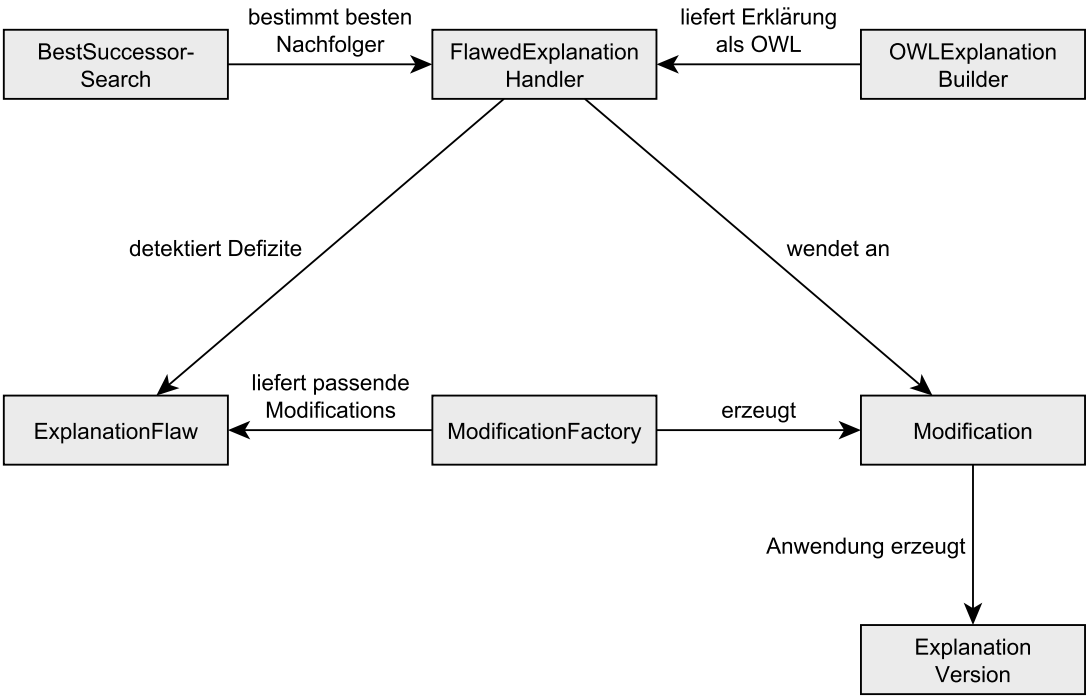


Abbildung 7.2: Überblick über die an der Behandlung von Defiziten beteiligten Klassen

Detektion von Defiziten

Um die Zugehörigkeit von Erklärungsbestandteilen in die verschiedenen Defizitklassen zu prüfen, müssen zunächst alle von der Anwendung definierten Defizite

ermittelt werden. Dazu werden bereits bei der Instanziierung eines `FlawedExplanationHandler` alle diese Klassen gesammelt, indem eine Anfrage an den Reasoner nach den Subklassen der verschiedenen Defizitklassen gerichtet wird. Ist nun in der oben beschriebenen Schleife eine Erklärung auf Defizite zu überprüfen, muss diese zunächst in die Ontologie eingefügt werden, wozu ein `OWLExplanationBuilder` verwendet wird. Dieser liefert die Individuen, die die Erklärungsbestandteile repräsentieren, und anschließend den von der Anwendung vorgegebenen Defiziten zugeordnet werden. Ist eine solche Zuordnung erfolgreich, besteht ein Defizit der übergeordneten Defizitklasse, für welches dann Modifikationen zur Behebung zu ermitteln sind.

Modifikation von Erklärungen

Um die Modifikationen zu bestimmen, die ein Defizit beseitigen können, wird zunächst ein Objekt der Klasse `ExplanationFlaw` erzeugt. Dieses beinhaltet die Defizitklasse und das betroffene Element der Erklärung, so dass eindeutig zuzuordnen ist, an welchem Element der Erklärung das Defizit genau besteht (siehe auch Kapitel 6.3.1, wo die Problematik für Planschritt-Defizite erläutert wurde). Der `ExplanationFlaw` wird dann als Parameter beim Aufruf der Methode `getModifications` der Klasse `ModificationFactory` verwendet, die eine oder mehrere passende Modifikationen zurückliefert.

Für die Behebung unserer Defizitklassen sind letztlich nur zwei verschiedene Modifikationen nötig. Die einfachere der beiden, die Entfernung einer Verbindung zur Behebung eines Verbindungsdefizits, wird durch eine Modifikation der Klasse `RemoveConnectionModification` umgesetzt. Die für alle anderen verwendete Modifikation ist die `RemovePlanstepModification`, die einen Planschritt der Erklärung eliminiert. Je nachdem, welche Art von Defizit dadurch behandelt werden soll, ist dafür ein anderer Planschritt aus dem Umfeld des betroffenen Objekts, bzw. im Falle eines Planschritt-Defizits der Planschritt selbst, zu entfernen, und die damit zusammenhängenden Erklärungen zu modifizieren.

Nachdem die passenden Modifikationen ermittelt wurden kann die Defizitbehandlung mit der Anwendung aller Modifikationen für jeden `ExplanationFlaw` fortgesetzt werden. Dabei werden allerdings Modifikationen zur Behebung von Überlange-Erklärung-Defiziten nur angewendet, wenn eine Erklärung keine Defizite auf der Argument-Ebene besitzt. Dies dient dem Zweck, erst gezielt Planschritte zu eliminieren, wodurch die Erklärung ja ebenfalls verkürzt wird, und so eine möglicherweise unnötige Auslassung von Information wenn es geht zu vermeiden. Wurde eine Erklärung im Verlauf einer Modifikation verändert, sollte dies kenntlich gemacht werden, damit die Anwendung gegebenenfalls eine andere Darstellung für diese Schritte wählen kann. Zu diesem Zweck wurde eine Container-Klasse `ExplanationVersion` implementiert, die die ursprüngliche Erklärung und die Modifikation, die zu der vorliegenden Fassung der Erklärung geführt hat, beinhaltet. Um den Zugriff auf

diese Versionen zu ermöglichen, wurde zudem die `Explanation`-Klasse aus der Erklärbär-Bibliothek um entsprechende Methoden erweitert.

Wurden für eine Erklärung alle Modifikationen einer „Detektionsrunde“ durchgeführt, steht eine Anzahl von möglichen Nachfolgern zur Auswahl. Unsere Implementierung fährt an dieser Stelle nicht mit der Weiterentwicklung all dieser Nachfolger fort, sondern verwirft alle bis auf den Vielversprechendsten. Wir haben uns für diesen Schritt entschieden, weil die Anzahl von Erklärungen sonst extrem anwachsen würde, wie im theoretischen Teil der Arbeit bereits erläutert wurde. Um die Auswahl des zu verwendenden Nachfolgers zu treffen, wird eine Implementierung der Klasse `BestSuccessorSearch` verwendet; In unserer Implementierung wird dabei derjenige Nachfolger selektiert, der die wenigsten Defizite besitzt.

7.5.4 Bewertung von Erklärungen

Zur Bewertung der Erklärungen kommt die Klasse `ExplanationEvaluationHandler` zum Einsatz. Sie setzt die Bewertungsfunktion unseres Ansatzes in einer Methode `handle` um, die die von Defiziten befreiten Erklärungen als Parameter erhält. Die Vorgehensweise basiert auch hier wie bei der Defizit-Detektion auf einer Zuordnung der Teile der Erklärung zu den die verschiedenen Werte repräsentierenden OWL-Klassen. Bei der Instanziierung werden daher alle OWL-Klassen bestimmt, die die von der Anwendung definierten Werte verkörpern, also die Subklassen von `RawExplanationValues` und `ArgumentValues`. In der `handle`-Methode werden dann die Erklärung und ihre Argumente darauf überprüft, als Instanz welcher dieser Klassen sie zu klassifizieren sind. Dazu werden die Erklärungen einem `OWLExplanationBuilder` übergeben, der OWL-Repräsentationen der Erklärungen erzeugt und zusätzlich die Individuen, die die Roherklärung und ihre Argumente abbilden, gesondert zur Verfügung stellt. Jede erfolgreiche Zuordnung dieser Individuen zu einer der Wert-Klassen bedeutet, dass ein solches Element die dort definierten erwünschten Eigenschaften besitzt, und daher einen Wert zur Erklärung beiträgt, der aus der Annotation der Klasse entnommen wird. Ist jede Erklärung auf diese Weise vollständig überprüft, können ihre Gesamtwerte verglichen, und die Erklärung mit dem höchsten Gesamtwert zurückgegeben werden.

7.6 Anwendungsbeispiel ShellClient: Textbasierte Planerklärung in der Smartphone-Domäne

Um den Einsatz des Frameworks demonstrieren zu können wurde im Rahmen dieser Arbeit auch eine einfache Anwendung implementiert, die die textbasierte Planerklärung in der Smartphone-Domäne ermöglicht. Zur Umsetzung dieser Anwendung wurde das in [ler08] vorgestellte System *PandaX* zum Mixed Initiative Planen so

erweitert, dass seine Client-Server-Architektur auch zu anderen Zwecken als dem Planen verwendet werden kann. Wir behandeln an dieser Stelle allerdings nur die Bereiche des Systems, die für unsere Zwecke zu modifizieren waren, sowie die zusätzlich zur Umsetzung der Anwendung implementierten Klassen. Für eine genaue Beschreibung der zu Grunde liegenden Architektur wird daher auf [ler08] verwiesen.

Wir beschreiben im nächsten Abschnitt, wie die Benutzung des Systems abläuft, wobei insbesondere auf die zentrale Fragen, mit denen sich Entwickler von Anwendungen auf Basis unseres Frameworks auseinandersetzen müssen, eingegangen wird. Dazu zählt, welche Anfragen überhaupt möglich sind, was vor allem von der Darstellung der Pläne und den verfügbaren Interaktionsmöglichkeiten abhängt. Daneben ist natürlich die Darstellung von Erklärungen an sich von Bedeutung, da diese nicht nur die Verständlichkeit von Erklärungen für den Nutzer, sondern auch den Entwurf eines Schnittstellenmodells für die Erklärungserzeugung beeinflusst. Im anschließenden Abschnitt 7.6.2 gehen wir auf einige Modifikationen ein, die an der zu Grunde liegenden Architektur zur Nachrichtenvermittlung und -Behandlung vorgenommen wurden, um unsere Anwendung darauf aufbauen zu können. Wir beschließen das Kapitel mit der Beschreibung der Nutzer- und Schnittstellenmodelle sowie der Annotationen, die an der Smartphone-Domäne vorgenommen wurden.

7.6.1 Der ShellClient

Unsere Anwendung ist so aufgeteilt, dass ihre Serverseite eine Instanz der `Explainer`-Klasse kapselt, deren Steuerung durch Nachrichten erfolgt, die von der Clientseite eingehen. Nachdem am Client die Verbindung zu einem Server hergestellt wurde, muss der Server zunächst informiert werden, welches Nutzermodell verwendet werden soll. Wir haben zu diesem Zweck ein Nutzermodell mit Namen „user“ erstellt, zu dessen Verwendung der Server mit dem Befehl „hello user“ angewiesen wird. Anschließend besteht die Möglichkeit, sich eine Liste der beim Server verfügbaren Pläne ausgeben zu lassen. Dies geschieht durch Eingabe des Befehls „show plans“. Der Server liefert daraufhin eine Liste von IDs von Plänen, aus denen der Nutzer durch Eingabe von „show plan <ID>“ einen zur Darstellung auswählen kann.

Der ShellClient versieht bei der Anzeige von Plänen die darin vorkommenden Planschritte mit IDs. Diese sind notwendig, um dem Nutzer die Adressierung seiner Anfragen zu ermöglichen, also um anzugeben, die Notwendigkeit welches Tasks oder die Anordnung welcher Tasks erklärt werden soll; Eine Möglichkeit für Anfragen zur Gleichheit von Variablen besteht in dieser Anwendung nicht, da die Darstellung mit zusätzlichen IDs für jeden Parameter der Tasks, die zur Auswahl der Parameter der Anfrage benötigt werden, sehr unübersichtlich würde. Das Absetzen dieser Anfragen erfolgt dann mit „query Task <ID>“ bzw. „query Before <ID> <ID>“. Ist eine Anfrage abgesetzt, wird diese an die Serverseite kommuniziert, wo eine

passende Erklärung generiert und anschließend an den Client übertragen wird. In der Darstellung der Erklärung werden, wie in Abbildung 7.3 zu sehen ist, die IDs wieder aufgegriffen, um die Beziehung zwischen den Bestandteilen des Plans und der Erklärung zu verdeutlichen, was vor allem dann nötig ist, wenn ein Task in einem Plan und damit gegebenenfalls auch in der Erklärung mehrmals vorkommt.

Abbildung 7.3: Die Ausgabe eines Plans und einer Erklärung im ShellClient

7.6.2 Anpassung der Kommunikationsstruktur

Ein zentraler Bestandteil des hier implementierten Systems ist der Austausch und die Behandlung von Nachrichten zwischen Client und Server. Da die Basisarchitektur ursprünglich nur dazu gedacht war, Zugriff auf ein auf einem entfernten Server laufendes `PANDA`-Planungssystem zu gewähren, war der Hauptansatzpunkt für unsere Implementierung die Behandlung verschiedener Nachrichten, die durch eine Klasse `MessageHandler` realisiert wird.

Generalisierung der MessageHandler-Klasse

Die `MessageHandler` stellen in dieser Architektur die Schnittstellen zu den verschiedenen Subsystemen dar, die auf Client- sowie Serverseite in die Anwendung integriert sind. Um diese Subsysteme steuern zu können registrieren sich `MessageHandler` bei ihrer Anwendung und geben dabei an, für welchen Typ von Nachrichten sie zuständig sind. Ab dem Moment dieser Registrierung können

eingehende Nachrichten dann ihrem Typ entsprechend den registrierten Handlern zugeordnet und von diesen verarbeitet werden.

Der `ServerExplanationMessageHandler` beispielsweise kapselt einen `Explainer`, und übernimmt alle den Bereich der Planerklärung betreffenden eingehenden Nachrichten, die als Subklassen der von `Message` abgeleiteten Klasse `ExplanationMessage` implementiert sind. Um neben der Erweiterung des Systems auch die bisherige Funktionalität zur interaktiven Planung zu erhalten, wurden zusätzlich zu den Klassen für die Planerklärung auch entsprechende Ableitungen von `MessageHandler` und `Message` zur Anzeige von Plänen, der Anwendung von Modifikationen etc. entwickelt. Abbildung 7.4, S. 95 zeigt einen Ausschnitt der Architektur von Client und Server, die dabei letztlich entstanden ist.

Nachrichten für die Planerklärung

Für die Planerklärung wurden in der Anwendung drei Subklassen von `ExplanationMessage` ausgeprägt. Um dem `Explainer` die Pfade zu Nutzer- und UI-Modell zu übermitteln verwendet der Client Nachrichten der Klasse `ModelSetupMessage`, Anfragen an das Erklärungssystem werden mit Nachrichten der `QueryMessage`-Klasse abgesetzt. Beide Nachrichten werden auf Serverseite vom `ServerExplanationMessageHandler` entgegengenommen, der mit den Dateipfaden den `Explainer` konfiguriert bzw. die enthaltenen Erklärungsanfragen an das Erklärungssystem weiterleitet. Um eine daraufhin erzeugte Erklärung an den Client zu übermitteln, versendet der Server eine `DeliverExplanationMessage`, aus der der `ClientExplanationMessageHandler` die Erklärung entnimmt und ausgibt.

7.6.3 Wissensmodelle

Bei der Umsetzung der Anwendung waren auch ein Modell zur Repräsentation der Schnittstelle und ein exemplarisches Nutzermodell zu entwickeln, sowie eine Annotation der Smartphone-Domäne mit zusätzlich zu verwendendem Wissen vorzunehmen. Da die Darstellungsmöglichkeiten des ShellClient relativ unbeschränkt sind, wurde im Modell der Schnittstelle lediglich ein Überlange-Erklärung-Defizit für Erklärungen mit mehr als sechs Argumenten definiert. Das Nutzer- und das Domänenmodell werden in den folgenden Abschnitten kurz vorgestellt.

Domänenmodell

In der Smartphone-Domäne lassen sich zwei große *Domänenaspekte* identifizieren. Einen großen Bereich der Domäne machen Informationen und ihre Verwendung aus, um beispielsweise Kontakte oder Termine anzulegen. Zu diesem

`Informationsaspekt` gehören also beispielsweise die Sorte `Information` und ihre Subsorten, sowie Tasks und Methoden zum Erstellen von Erinnerungsereignissen, wie `create_Task`. Die Teile der Domäne, die zum Verpacken von Informationen in Nachrichten und deren Übertragung gehören, bilden dagegen den `Nachrichtenaspekt`. Ihm ist zum Beispiel der Task `transferMessage` mit seinen implementierenden Methoden und deren Subtasks zuzuordnen, ebenso wie die `Message`-Sorte mit Subsorten sowie die Relationen, die den Besitz von Information ausdrücken.

Einige Teile der Domäne konnten als eher *technisch* identifiziert werden. Hierunter fallen zum einen die Tasks, die das Drücken von Buttons modellieren, welche in der Domäne mit dem Präfix `press_` versehen sind. Zum anderen sind die verschiedenen Implementierungen und zugehörigen Subtasks von `enterMode_Home` für den Nutzer wenig informativ, da sie nur dazu dienen, für die Modellierung relevante Nebeneffekte umsetzen. Schließlich wurden in der Domäne auch einige Bestandteile als *selbsterklärend* markiert, wie beispielsweise die Tasks, die den Wechsel in die verschiedenen Modi ermöglichen, und die Relationen, die dem Setzen dieser Modi dienen.

Nutzermodell

Unser Beispielnutzer ist ein mit der Bedienung von Smartphones grundsätzlich vertrauter Nutzer, weshalb technische Tasks wie das Drücken bestimmter Schaltflächen für ihn nicht interessant sind, was durch die Definition von Planschritt-Defiziten bei der Verwendung solcher Tasks in das Nutzermodell übertragen wurde. Inhaltlich ist er hauptsächlich daran interessiert, wie mit dem Gerät Informationen übermittelt werden können. Um dies abzubilden wurde eine Subklasse von `ArgumentValues` definiert, die der Referenzierung von Tasks des `Nachrichtenaspekts` der Domäne einen Wert verleiht. Was die formalen Aspekte der Erklärungen angeht, so bevorzugt er funktionale Argumente, die ihm einen einfachen Überblick über die Zusammenhänge auf niedriger Abstraktionsebene bieten, weshalb eine weitere Subklasse von `ArgumentValues` in das Nutzermodell aufgenommen wurde.

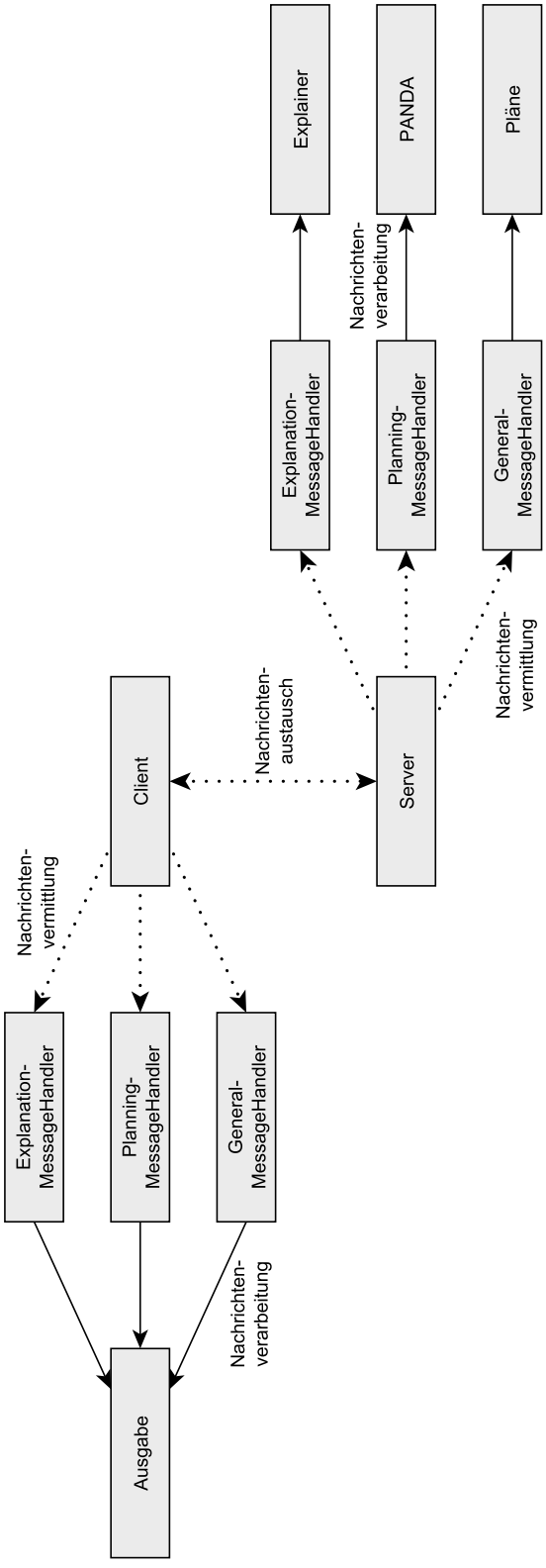


Abbildung 7.4: Nachrichtenvermittlung und -verarbeitung im ShellClient

8 Verwandte Arbeiten

Es gibt einige Arbeiten, die sich ebenfalls mit den in dieser Arbeit berührten Bereichen beschäftigen, worunter vor allem solche zu den Themen *Mixed Initiative Planning* und Erklärung fallen. Wir wollen an dieser Stelle einen kurzen Überblick über einige dieser Arbeiten geben.

8.1 Mixed Initiative Planning

Arbeiten zum Thema *Mixed Initiative Planning*, das heißt dem kooperativen Planen von Planungssystem und Nutzer, besitzen in zwei Aspekten Überschneidungen mit dieser Arbeit. Zum einen wird durch eine Beteiligung des Nutzers am Planungsprozess das Vertrauen in das Planungssystem allgemein und in die Qualität eines gemeinsam entwickelten Plans im speziellen gestärkt, da dessen Zustandekommen direkt verfolgt werden kann. Zum anderen müssen Pläne aus der rein formalen Repräsentation, die das Planungssystem benötigt, in eine vom Nutzer verständliche Form gebracht werden, was der Darstellung von Erklärungen ähnelt, wie sie in von Anwendungen des in dieser Arbeit vorgestellten Frameworks vorgenommen werden muss. Der Unterschied liegt jedoch darin, dass sie nicht wie im in dieser Arbeit gezeigten Ansatz die Erklärung explizit machen, sondern durch eine naheliegende Präsentationsform dem Nutzer ermöglichen, den Plan zu erkunden und letztlich zu verstehen.

In [SBGB09] wird ein Mixed Initiative Planning System vorgestellt, in dem ein Nutzer einen Planraum erforscht und erweitert, der gleichzeitig auch durch ein hybrides Planungssystem bearbeitet wird. Das Planungssystem wird dabei durch eine Ausrichtung an den vom Nutzer betrachteten Plänen gesteuert, und versucht, dessen Entscheidungen anhand eines Nutzermodells vorauszusehen, so dass das Resultat einer vom Nutzer ausgewählten Planentwicklung nach Möglichkeit bereits vorberechnet wurde.

Eine Anzahl von Mixed Initiative Planning Systemen wird mit Hilfe von Constraint Solution Plannern realisiert. Ihnen gemein ist eine Vorgehensweise, bei der der Nutzer Constraints für ein Planungsproblem definiert, wobei er ein speziell auf die jeweilige Anwendung und Domäne ausgerichtetes Interface verwendet, welches auch zur Präsentation des Plans dient. Im Falle des in [Sei07] vorgestellten Systems dient dazu eine Landkarte, auf der zu besichtigende Sehenswürdigkeiten ausgewählt

werden können. Das System entwicklet anhand dieser Nutzereingaben eine Route, die zu den gewählten Orten führt. Die Notwendigkeit, eine bestimmte Route zu fahren, wird dabei durch die Hervorhebung erreichter Sehenswürdigkeiten verdeutlicht. Ein weiteres System welches ebenfalls auf eine Landkarte zur Planentwicklung und -darstellung zurückgreift wird in [ASF+94] beschrieben. Dort wird das Ziel verfolgt, ein Assistenzsystem zu entwickeln, welches innerhalb der TRAINS-Domäne, in der Zugfahrten zu planen sind, zu einer bidirektionalen natürlichsprachlichen Kommunikation mit dem Nutzer in der Lage ist. [CFOG00] beschreibt das Planungssystem MACHINE, mit dem Produktionsprozesse entwickelt werden. Als Benutzerschnittstelle kommt dabei eine Darstellung mittels Petri-Netzen zum Einsatz, die auch vor dem Einsatz maschineller Planungssysteme zur Prozessplanung verwendet wurden. Das MAPGEN-System (s. [ACBC+04]) schließlich dient der Missionsplanung für die Roboter, die von der NASA zur Marserkundung eingesetzt werden. Auch dort wurde eine den Nutzern des Systems aus ihrer bisherigen Arbeit vertraute Benutzeroberfläche verwendet, in diesem Fall GANTT-Charts.

8.2 Erklärung

Auch mit der Erklärung von Beweisen bzw. Plänen haben sich einige Arbeiten bereits auseinandergesetzt. Ein Ansatz zur Erklärung von Plänen auf natürlichsprachliche Weise wird in [BBH+10] beschrieben. Dort wird eine Ableitung von Erklärungen aus den Zusammenhängen im Plan vorgeschlagen, die anschließend mit Hilfe von entsprechenden Annotationen an der zugrunde liegenden Domäne verbalisiert und dem Nutzer präsentiert werden.

In [BBG+11] wird ein planbasiertes Assistenzsystem beschrieben, welches einen Nutzer bei der Lösung verschiedener Aufgaben unterstützt. Das System generiert Handlungspläne für den Nutzer unter Einbezug seiner Eigenschaften und ist in der Lage, ein Fehlschlagen dieser Pläne durch Planreparatur abzufangen. Darüber hinaus werden Aspekte diskutiert, die in einer Planerklärung eine Rolle spielen können, mit der das System das Verständnis des Nutzers für den Plan und damit sein Vertrauen in das Assistenzsystem weiter verstärken kann.

In [GKE+10] wird eine Methode präsentiert, mit der Erklärungen dafür entwickelt werden können, dass ein Planungssystem keinen Plan zur Lösung eines Problems finden konnte. Die so gefundenen Erklärungen haben die Form von Ausreden, das heißt sie erklären, dass unter gewissen Veränderungen der Problemstellung ein Plan hätte gefunden werden können. Eine Erklärung wird dabei dann als gut befunden, wenn sie möglichst allgemein ist, und möglichst wenige Veränderungen vorschlägt.

Das in [Fie01] beschriebene System *P.rex* ist ein System zur Erklärung von Theorem-Beweisen. P.Rex funktioniert im Gegensatz zum in dieser Arbeit vorgestellten System dialoggesteuert, und ist auf den Aspekt der Wissensvermittlung ausgerichtet.

Dazu wird im laufenden Dialog mit dem Nutzer dessen Wissensstand modelliert, um an die Nutzerkenntnisse angepasste Erklärungen liefern zu können.

9 Zusammenfassung und Ausblick

9.1 Zusammenfassung

Wir haben in dieser Arbeit ein Framework entwickelt, welches zur Umsetzung von Systemen zur nutzerorientierten Planerklärung eingesetzt werden kann. Dazu wurden zunächst allgemeine Merkmale für die Qualität von Erklärungen, und eine Funktion zur Bewertung dieser Qualität ermittelt. Darauf aufbauend wurde ein Prozess entworfen, der die angestrebte nutzerorientierte Planerklärung realisiert. Dieser generiert zunächst verschiedene mögliche Erklärungen, modifiziert diese zur Vermeidung bestimmter Ausschlusskriterien, sogenannter *Defizite*, und wählt anschließend auf der Grundlage verschiedener *Werte* die für die gegebene Anwendungssituation beste zur Ausgabe an den Nutzer aus. Defizite sowie Werte werden dabei von der das Framework einsetzenden Anwendung in deklarativer Form definiert und dienen zur Steuerung der Abläufe des beschriebenen Prozesses, womit sie die Schnittstelle zwischen Framework und Anwendung darstellen.

Die Spezifikation dieser Schnittstelle besteht daher in der Beschreibung einer Wissensbasis, in der Planungsdomänen und Erklärungen repräsentiert werden, sowie Vorgaben, wie Defizite und Werte darin abzubilden sind. Wir haben darüber hinaus ermittelt, woher deren Definitionen abgeleitet werden können, und dabei Eigenschaften des Nutzers, die zur Darstellung verwendete Schnittstelle sowie die dem Plan zu Grunde liegende Domäne als Informationsquellen identifiziert. Schließlich wurde eine im Rahmen der Arbeit erstellte prototypische Implementierung des Frameworks, die den im theoretischen Teil entwickelten Ansatz umsetzt, sowie eine darauf aufbauende Beispielanwendung beschrieben.

9.2 Ausblick

Während der Erstellung dieser Arbeit wurde einige Ansatzpunkte für mögliche Erweiterungen gefunden, von denen wir in diesem Kapitel einige vorstellen wollen. Im Mittelpunkt steht hier die Idee, die Wissensbasis, die zur Defiziterkennung und Bewertung von Erklärungen verwendet wird, dynamischer zu gestalten. Es sind mehrere Punkte denkbar, an denen dadurch eine Verbesserung des Erklärungsvorgangs erreicht werden könnte, und die in unserer Implementierung verwendete OWL-API

bietet bereits entsprechende Möglichkeiten, die Wissensbasis zu diesem Zweck zur Laufzeit zu verändern.

Dynamische Nutzermodellierung

Als ein Ansatzpunkt ist eine dynamischere Nutzermodellierung aufzuführen. Die Vorteile, die sich durch eine Ergänzung unseres statischen Modells auf diesem Weg ergeben würden, wurden teilweise bereits in der Arbeit behandelt. Es wäre zum Beispiel denkbar, die Informationen im Nutzermodell zu korrigieren, falls diese sich als nicht zutreffend herausstellen. Die Eigenschaft von Domänenelementen, selbsterklärend zu sein, wurde diesbezüglich bereits als möglicher Grenzfall eingestuft, weil sie gewisse Grundannahmen über die damit konfrontierten Nutzer macht. Es könnte aber auch sein, dass ein Nutzer beispielsweise die Funktion eines Tasks vergessen hat. Ein statisches Nutzermodell kann solche Änderungen während des Erklärungsvorgangs nicht umsetzen. Eine Anwendung, die dem Nutzer entsprechende Möglichkeiten bietet, Rückmeldung über erhaltene Erklärungen zu geben, könnte hier ansetzen, und eine Markierung eines Tasks als nicht mehr bekannt in das Modell übernehmen.

Ein anderes Ansatzpunkt ergibt sich aus einem Themengebiet, welches eine große Schnittemenge mit unserem Ansatz besitzt, nämlich aus der *Wissensvermittlung*. Wir haben die Unkenntnis des Nutzers von Domänenbestandteilen bisher als starkes Indiz dafür betrachtet, diese in der Erklärung zu vermeiden. Tatsächlich wären aber zum Beispiel Fragen über die Notwendigkeit von Tasks, die der Nutzer nicht kennt, von erheblichem Interesse, denn ihre Beantwortung würde nicht nur das Verständnis des Nutzers für einen Plan, sondern darüber hinaus noch seine Kenntnis der Domäne verbessern. Die Einsatzgebiete solcher Systeme, die Planerklärung zur Wissensvermittlung einsetzen, wären aber wahrscheinlich nicht die gleichen wie von Assistenzsystemen, denen Planerklärung eher zur Bildung von Vertrauen in ein Planungssystem und der Unterstützung eines Nutzers bei verschiedenen Tätigkeiten dient.

Kürzen von Erklärungen

Eine weitere Eigenschaft von Erklärungen, die häufig zu einem Defizit führt, wurde in der Anzahl der verwendeten Argumente identifiziert. Welches Argument entfernt werden soll, um dieses Defizit zu beseitigen, wird bisher aber nur über die anschließende Auswertung der Nachfolger entschieden. Hier wäre sowohl im Sinne der Qualität der Erklärung als auch der Laufzeit der Erklärungserzeugung wünschenswert, gezielter vorgehen zu können. Es wäre beispielsweise denkbar, dass unterschiedliche Argumente bei der Ausgabe durch eine Nutzerschnittstelle unterschiedlich viel Platz benötigen, so dass die Auslassung eines „großen“ Arguments der zweier „kleiner“ Argumente vorzuziehen wäre, da dabei weniger Information verloren ginge. Eine

Möglichkeit besteht hier darin, bei der Detektion von Defiziten nicht nur auf das statische UI-Modell zurückzugreifen, sondern eine Erklärung im Verlauf der Defizitdetektion der Anwendung zu übergeben und von dieser analysieren zu lassen. Da die Mächtigkeit der Wissensrepräsentation begrenzt ist, könnte sich aus dieser Analyse eine präzisere Detektion von Defiziten dieser konkreten Erklärung ergeben.

Die Idee, die Anwendung zur Laufzeit in die Defizitdetektion einzubinden, ließe sich im Übrigen auch auf die Bewertung von Erklärungen übertragen. Da die genaue Abbildung eines Darstellungsalgorithmus' in der Wissensbasis kaum realisierbar ist, könnte stattdessen von der Nutzerschnittstelle direkt im Verlauf der Erklärungsbewertung erfragt werden, welche Teile einer konkreten Erklärung dargestellt werden, und daher bei der Bewertung der Erklärung einbezogen werden sollten.

Auch unabhängig von echten Defiziten kann das Kürzen von Erklärungen zur Qualität des Erklärungsprozesses beitragen, denn Kürze und Prägnanz konnten ja als generelle Qualitätsmerkmale von Erklärungen identifiziert werden. Ein Ansatzpunkt bietet sich hier, wenn mehrere Erklärungen für Eigenschaften des selben Plans nacheinander angefragt werden. Wurde in einer ersten Erklärung beispielsweise die Notwendigkeit eines Tasks im Laufe der Argumentation erklärt, müsste dies für eine weitere Erklärung nicht unbedingt nochmal gemacht werden, da der Nutzer dann bereits einen Grund für dessen Notwendigkeit kennt. Hierbei ergeben sich allerdings zwei Punkte, die bei einer möglichen Abkürzung einer Erklärung auf der Grundlage solchen episodischen Wissens in Betracht zu ziehen sind. Erstens kann eine weitere Erklärung nach wie vor zu einem besseren Verständnis des Plans beitragen, denn die Notwendigkeit eines darin enthaltenen Tasks kann ja durch verschiedene Beziehungen begründet werden. Zweitens sind die Ableitungen, die der Nutzer macht, schwer vorherzusehen: Wurde zum Beispiel die Notwendigkeit eines Tasks mit der Herstellung eines kausalen Links für einen anderen Task erklärt, so kann der Nutzer daraus auch den Schluss ziehen, dass der Produzent dieses kausalen Links natürlich vor dem Konsumenten angeordnet sein muss. Ob aber ein bestimmter Nutzer zu dieser Schlussfolgerung fähig ist oder nicht, ist nicht ohne weiteres festzustellen.

Landmarken von Plänen

Die Wissensrepräsentation enthält aktuell keine Informationen über den Plan, abgesehen von dessen faktischem Aufbau. Landmarken, also Tasks, die von besonderer Bedeutung für die Ausführbarkeit von Plänen sind, in den Erklärungsprozess einzubeziehen, könnte sich jedoch als äußerst hilfreich erweisen. Ein Aspekt ist dabei, dass diese Landmarken in Erklärungen einen besonders hohen Informationsgehalt aufweisen, da sie von kritischer Bedeutung für den Plan sind, und ihre Verwendung in der Argumentation entsprechend auf wichtige Zusammenhänge darin hinweisen würde.

Daneben kann die Kenntnis von Landmarken eines Plans auf Anwendungsseite dazu verwendet werden, eine Erklärung zu gliedern. Dies könnte als Alternative zu einer Verkürzung der Erklärung geschehen, so dass die Entfernung von Argumenten und der damit einhergehende Verlust von Informationen vermieden werden könnte, aber auch, um einem Nutzer Gelegenheit zu geben, Zwischenfragen zu stellen, und sich zum Beispiel einen bestimmten Teil der bereits gegebenen Erklärung detaillierter erläutern zu lassen.

Literaturverzeichnis

[ACBC+04] M. Ai-Chang, J. Bresina, L. Charest, A. Chase, J.C.-J. Hsu, A. Jonsson, B. Kanefsky, P. Morris, Kanna Rajan, J. Yglesias, B.G. Chafin, W.C. Dias, and P.F. Maldague. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems, IEEE*, 19(1):8 – 12, jan-feb 2004.

[ASF+94] James F. Allen, Lenhart K. Schubert, George Ferguson, Peter Heeman, Chung Hee Hwang, Tsuneaki Kato, Marc Light, Nathaniel G. Martin, Bradford W. Miller, Massimo Poesio, and David R. Traum. The trains project: A case study in building a conversational planning agent. *Journal of Experimental and Theoretical AI*, 7:7–48, 1994.

[BBG+11] Susanne Biundo, Pascal Bercher, Thomas Geier, Felix Müller, and Bernd Schattenberg. Advanced user assistance based on AI planning. *Cognitive Systems Research*, 12(3-4):219–236, April 2011. Special Issue on Complex Cognition.

[BBH+10] Julien Bidot, Susanne Biundo, Tobias Heinroth, Wolfgang Minker, Florian Nothdurft, and Bernd Schattenberg. Verbal explanations for hybrid planning. In Matthias Schumann, Lutz M. Kolbe, Michael H. Breitner, and Arne Frerichs, editors, *Proceedings of the Conference "Multikonferenz Wirtschaftsinformatik"(MKWI 2010),Teilkonferenz Planung/Scheduling und Konfigurieren/Entwerfen*, pages 2309–2320, Göttingen, Germany, 2 2010. Universitätsverlag Göttingen.

[BBS08] Julien Bidot, Susanne Biundo, and Bernd Schattenberg. Plan repair in hybrid planning. In Andreas Dengel, Karsten Berns, Thomas Breuel, Frank Bomarius, and Thomas R. Roth-Berghofer, editors, *KI 2008: Advances in Artificial Intelligence, Proceedings of the 31st German Conference on Artificial Intelligence*, volume 5243 of *Lecture Notes in Artificial Intelligence*, pages 169–176, Kaiserslautern, Germany, September 2008. Springer.

[BBS10] Susanne Biundo, Julien Bidot, and Bernd Schattenberg. Planning in the real world. *Informatik-Spektrum*, 2010. (under review).

[BCM+07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, New York, NY, USA, 2007.

[BS01] Susanne Biundo and Bernd Schattenberg. From abstract crisis to concrete relief – A preliminary report on combining state abstraction and HTN planning. In *Proceedings of the 6th European Conference on Planning (ECP-01)*, pages 157–168, 2001. Preprints.

[CFOG00] Luis A. Castillo, Juan Fernández-Olivares, and Antonio González. Automatic generation of control sequences for manufacturing systems based on partial order planning techniques. *AI in Engineering*, 14(1):15–30, 2000.

[EHN94] Kutluhan Erol, James Hendler, and Dana S. Nau. Umcp: A sound and complete procedure for hierarchical task network planning. 1994.

[ESB10] Mohamed Elkawkagy, Bernd Schattenberg, and Susanne Biundo. Landmarks in hierarchical planning. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 229–234. IOS Press, 2010.

[Fie01] Armin Fiedler. P.rex : An interactive proof explainer. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning*, volume 2083 of *Lecture Notes in Computer Science*, pages 416–420. Springer Berlin / Heidelberg, 2001.

[GKE+10] Moritz Göbeldecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. In Gerhard Lakemeyer, Hector J. Levesque, and Fiora Pirri, editors, *Cognitive Robotics*, number 10081 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[KPSR+09] Markus Krötzsch, Peter F. Patel-Schneider, Sebastian Rudolph, Pascal Hitzler, and Bijan Parsia. OWL 2 web ontology language primer. Technical report, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-primer-20091027/.

[ler08] Sascha Geßler. Ein Framework zur interaktiven hybriden Handlungsplanung. Master's thesis, Universität Ulm, 2008.

[MR91] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the ninth National conference on Artificial intelligence - Volume 2*, AAAI'91, pages 634–639. AAAI Press, 1991.

[PSMG09] Peter F. Patel-Schneider, Boris Motik, and Bernardo Cuenca Grau. OWL 2 web ontology language direct semantics. W3C recommendation, W3C, October 2009. http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/.

[SB07] Bernd Schattenberg and Susanne Biundo. A unifying framework for hybrid planning and scheduling. In Christian Freksa, Michael Kohlhase, and Kerstin Schill, editors, *KI 2006: Advances in Artificial Intelligence, Proceedings of the 29th German Conference on Artificial Intelligence*, volume 4314 of *Lecture Notes in Artificial Intelligence*, pages 361–373, Bremen, Germany, June 2007. Springer. ISBN 978-3-540-69911-8.

[SBGB09] Bernd Schattenberg, Julien Bidot, Sascha Geßler, and Susanne Biundo. A framework for interactive hybrid planning. In Bärbel Mertsching, Marcus Hund, and Zaheer Aziz, editors, *KI 2008: Advances in Artificial Intelligence, Proceedings of the 32nd Annual German Conference on AI*, volume 5803 of *Lecture Notes in Artificial Intelligence*, pages 17–24, Paderborn, Germany, 9 2009. Springer.

[See11] Bastian Seegebarth. Formale Aspekte der Erklärung hybrider Pläne. Master's thesis, Universität Ulm, 2011. In Bearbeitung.

[Sei07] Inessa Seifert. Region-based model of tour planning applied to interactive tour generation. In Julie Jacko, editor, *Human-Computer Interaction. HCI Intelligent Multimodal Interaction Environments*, volume 4552 of *Lecture Notes in Computer Science*, pages 499–507. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73110-8_54.

Abbildungsverzeichnis

2.1 Ein einfacher POCL-Plan für Configure_Contact 15
2.2 Funktionsweise einer Methode im HTN-Planen 17
2.3 Ein Planungsschritt des PANDA-Systems 23

4.1 Eine Implementierung des Tasks Call 34
4.2 Ein Plan zur Tätigung eines Anrufs 43
4.3 Der Aufbau einer Planerklärung 44
4.4 Eine Implementierung des Tasks configure_Contact 44
4.5 Eine Übersicht des Ansatzes zur nutzerorientierten Planerklärung . . 45

5.1 Eine Roherklärung für die Notwendigkeit von press_Home.People . . 58
5.2 Die Erklärung nach der Suche nach einem neuen Vorgänger für select_Contacts.ContactForContactable 59
5.3 Die Erklärung nach der Behebung des Argument-Defizits durch Eliminierung der Notwendigkeit des Arguments 60
5.4 Die Erklärung nach Behandlung des Verbindungs-Defizits 61

6.1 Legende für die Darstellung der Wissensbasis 64
6.2 Repräsentation von Roherklärungen, Argumenten und Defizitklassen 79
6.3 Eine Implementierung des Tasks send_EMail 80
6.4 Einige Implementierungen des Tasks press_Home 80
6.5 Repräsentation der Domäne 81

7.1 Ein Überblick über den Ablauf des Erklärungsprozesses 84
7.2 Überblick über die an der Behandlung von Defiziten beteiligten Klassen 88
7.3 Die Ausgabe eines Plans und einer Erklärung im ShellClient 92
7.4 Nachrichtenvermittlung und -verarbeitung im ShellClient 95

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.

Ulm, 30.06.2011

Christian Bauer