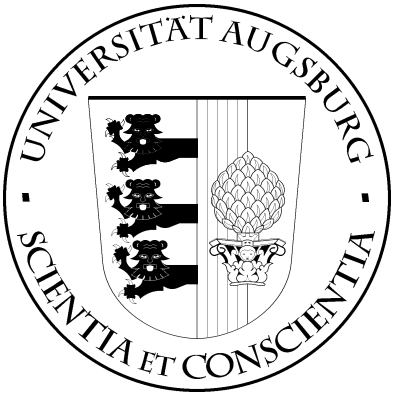
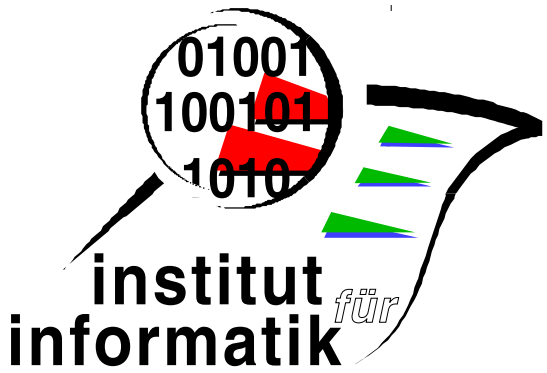
Universität Augsburg



Ontologie-basierte Modellierung und Synthese von Geschäftsprozessen

Florian Lautenbacher

Report 2005-16 September 2005



Institut für Informatik
D-86135 Augsburg

Kurzfassung

Die Modellierung von Geschäftsprozessen ist eine Aufgabe, die wichtige Entscheidungsträger von Firmen vornehmen und dabei ihr Fach- und Domänenwissen einbringen müssen, um ein Geschäftsfeld ausführlich und für andere verständlich darzustellen. Eine etwaige Änderung eines Teilprozesses bringt oft eine komplette Umstrukturierung der Geschäftsabläufe mit sich. Um das Geschäftsprozessmodell nicht manuell aktualisieren zu müssen, werden Methoden benötigt, die eine Unterstützung durch einen Computer ermöglichen.
Die Modellierung von Geschäftsprozessen muss daher um semantische Informationen erweitert werden, die für eine Maschine verständlich und auswertbar sind. Aufbauend auf den momentan entstehenden Standards des Semantic Web und Semantic Web Services-Umfeld können einzelne Prozesse semantisch annotiert werden. Wurden alle Prozesse durch semantische Informationen verfeinert, kann eine optimale Ausführungsreihenfolge aller Prozesse berechnet werden. Dieser sogenannte Synthesevorgang kann sowohl bei der Erstellung als auch bei einer Änderung eines Modells durchgeführt werden und erleichtert dadurch die Arbeit des Modellierers.

Die vorliegende Arbeit beschreibt einen Ansatz, wie Geschäftsprozessmodelle um semantische Informationen angereichert werden können. Dafür werden zuerst die am häufigsten verwendeten Modelle zur Beschreibung von Geschäftsprozessen erklärt. Anschliessend werden die verschiedenen Standards des Semantic Web- und Semantic Web Services-Umfeld evaluiert und es wird beschrieben welche Ontologien für eine semantische Prozessmodellierung benötigt werden. Die vorhandenen Konzepte aus dem Semantic Web-Bereich werden für die speziellen Belange der Ontologie-basierten Erweiterung von Geschäftsprozessmodellen aufgegriffen und erweitert. Um eine Synthese der Prozesse zu erreichen, wurden verschiedene Algorithmen entwickelt. Deren Aufbau und die Ergebnisse, die nach Ausführung der Syntheseverfahren im Rahmen diverser Beispiele erzielt wurden, werden detailliert beschrieben.
Um die Algorithmen testen zu können, entstand eine prototypische Implementierung. In diesem Prototyp kann ein Geschäftsprozess durch ein UML2-Aktivitätsdiagramm modelliert werden. Nach Erstellen des Modells ist es möglich eine semantische Anreicherung der einzelnen Prozesse vorzunehmen und eine Synthese durchzuführen. Sowohl der Aufbau und die Konzepte des Programmes, als auch das zugrundeliegende Rahmenwerk und benötigte Komponenten des Prototyps werden erklärt. Abschliessend wird dessen Installation und Bedienung beschrieben, bevor ein kurzer Ausblick über mögliche Erweiterungen der semantischen Modellierung und des Prototyps die Arbeit abrundet.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Seit vielen Jahren beschäftigt sich die Wirtschaftsinformatik mit der Beschreibung von Geschäftsprozessen. Diese werden entweder rein textuell annotiert oder graphisch mit Modellen dargestellt. Dafür haben sich im Laufe der Zeit verschiedene Standards herausgebildet: von Y-CIM über das ARIS-Haus bis hin zur mehr im Bereich der Informatik verwendeten Unified Modeling Language (UML).

Doch eine graphische Modellierung von Geschäftsprozessen allein ist nicht durch Maschinen auswertbar. Dies ist jedoch eine wichtige Grundlage, um den Ablauf von Geschäftsprozessen besser zu unterstützen und weiter optimieren zu können. Somit kann eine automatische Synthese von Geschäftsprozessen bei einer Modellierung (also die Bestimmung einer Ablaufreihenfolge), die von Computern auswertbar ist, die menschliche Arbeit bei der Erstellung eines Modells oder auch den Optimierungsaufwand bei einer geringen Änderung eines Modells drastisch reduzieren. Daher gehen jüngste Ansätze daran die Modellelemente mit zusätzlichen Metadaten zu annotieren.

Eine relativ neue Forschungsrichtung im Bereich des Internet ist das Semantic Web. Durch die hier entwickelten Standards soll erreicht werden, dass der Inhalt von Webseiten nicht mehr nur von Menschen sondern auch von Maschinen verstanden werden kann. Diese Entwicklungen im Semantic Web-Umfeld bieten hierfür die grundlegenden Standards, die auch leicht erweiterbar sind. Diese Standards, z.B. das Resource Description Framework (RDF) und die Web Ontology Language (OWL), können auch dazu verwendet werden, um computerverständliche Daten zu erstellen, damit Geschäftsprozessmodelle detailliert und einzelne Aktionen mit semantischen Informationen angereichert werden können (vgl. dazu auch [Nüt05]).

Sind alle Prozesse innerhalb eines Modells mit semantischen Beschreibungen versehen, kann ein Computer eine (semi-)automatische Synthese der Prozesse vornehmen, um auch bei Änderungen der Geschäftsprozesse eine optimale Reihenfolge gewährleisten zu können. Diese Synthese versucht eine optimale Abfolge aller Prozesse zu finden und dabei alle Parameter jedes Prozesses zu erfüllen.

1.2 Aufgabenstellung und Lösungsansatz

Die Modellierung von Geschäftsprozessen durch Diagramme, unterstützt durch eine textuelle Beschreibung wie einzelne Elemente zu verstehen sind und welche Aufgaben diese ausführen, ist bei

einer Nutzung allein durch den Menschen ausreichend. Soll aber ein Computer bei der Erstellung einer Ablaufreihenfolge oder Optimierung einer bestehenden Aktivität den Menschen unterstützen, muss diese bisher textuelle Beschreibung durch Konstrukte ersetzt werden, die durch den PC auswertbar sind.

Die in dieser Arbeit zu klärende Aufgabenstellung umfasst daher die Erweiterung eines Modells (hier: UML2-Aktivitätsdiagramm) zur Beschreibung von Geschäftsprozessen und dem Zusammenspiel dieser Prozesse. Sind die Prozesse erstellt, können diese mit semantischen Informationen versehen und danach kann eine Synthese gestartet werden. Die semantischen Objekte werden dabei aus einer Ontologie übernommen, die ebenfalls definiert werden muss. In der folgenden Synthese berechnet der PC die mögliche Verzahnung der Prozesse in verschiedenen Lösungen und bewertet diese. Die Lösung(-en) mit der besten Bewertung wird dem Benutzer zurückgegeben.

Im Rahmen dieser Arbeit werden zuerst die vorhandenen Semantic Web-Standards und die damit verbundenen Standards für Semantic Web Services evaluiert, um damit verbundene Beschreibungsmöglichkeiten von Prozessen zu finden. Darauf aufbauend wird mit den gefundenen Elementen ein Modell semantisch annotiert. Um eine Synthese vornehmen zu können, muss zuerst eine Möglichkeit gefunden werden mit Ontologien zu arbeiten und darin Schlussfolgerungen zu ziehen. Ausserdem werden verschiedene Syntheseverfahren beschrieben, die später in einer Fallstudie verglichen werden. Zuletzt werden die implementierungsspezifischen Details sowie die Benutzung eines Prototyps beschrieben, der im Rahmen dieser Arbeit entwickelt wurde.

Abbildung 1.1 beschreibt graphisch den groben Aufbau des Prototyps: in diesem Programm bzw. der Entwicklungsumgebung muss es möglich sein, die zugrundeliegende Ontologie zu editieren. Diese wird mit der Syntax und Semantik erstellt, wie sie in einem Schema definiert wurde. Ausserdem wird in der Entwicklungsumgebung ein Modell erstellt und mit semantischen Beschreibungen erweitert. Nach Fertigstellen des Aktivitätsdiagrammes wird von hier die Synthese gestartet. Diese baut auf einer Inferenz-Maschine auf, um das automatische Schlussfolgern zu ermöglichen. Die Synthese berechnet auf Basis unterschiedlicher Matrizen und darauf aufbauender Graphen-Algorithmen, die beste Zusammenstellung der Prozesse aus technischer Sicht. Dazu werden anfangs die Kombinationsmöglichkeiten von je zwei Prozessen betrachtet und diese numerisch bewertet. Diese Einträge werden in einer Matrix gespeichert und damit wird ein gerichteter und gewichteter Graph erstellt. Über diesem Graphen können Algorithmen den besten Ablauf der Prozesse unter Berücksichtigung der angegeben Bedingungen berechnen. Wenn eine Lösung alle Bedingungen erfüllt, wird sie aufgrund der verwendeten Kanten bewertet und aus der Summe aller Lösungen die beste Lösung ausgewählt und dem Nutzer angeboten.

1.3 Gliederung der Arbeit

Das folgende Kapitel 2 erklärt die Grundlagen der Modellierung von Geschäftsprozessen allgemein sowie an den Standards ARIS und UML2. Dabei wird hier besonders auf UML2-Aktivitätsdiagramme eingegangen, da sie sich am Besten für die Modellierung von Geschäftsprozessen eignen. Nachdem die Grundlagen des Semantic Web erläutert wurden, werden verschiedene Standardisierungsbemühungen für Semantic Web Services vorgestellt. Im Kapitel 3 wird darauf aufbauend eine semantische Beschreibung von Geschäftsprozessen vorgenommen, wobei zuerst die Möglichkeiten auf Basis der Semantic Web Service-Standards diskutiert werden, bevor die konkrete Umsetzung anhand eines Beispiels erläutert wird.

Die semantische Geschäftsprozessmodellierung ist die Grundlage, um eine automatische Synthese der Prozesse durchführen zu lassen, wie sie in Kapitel 4 genauer beschrieben wird. Dabei

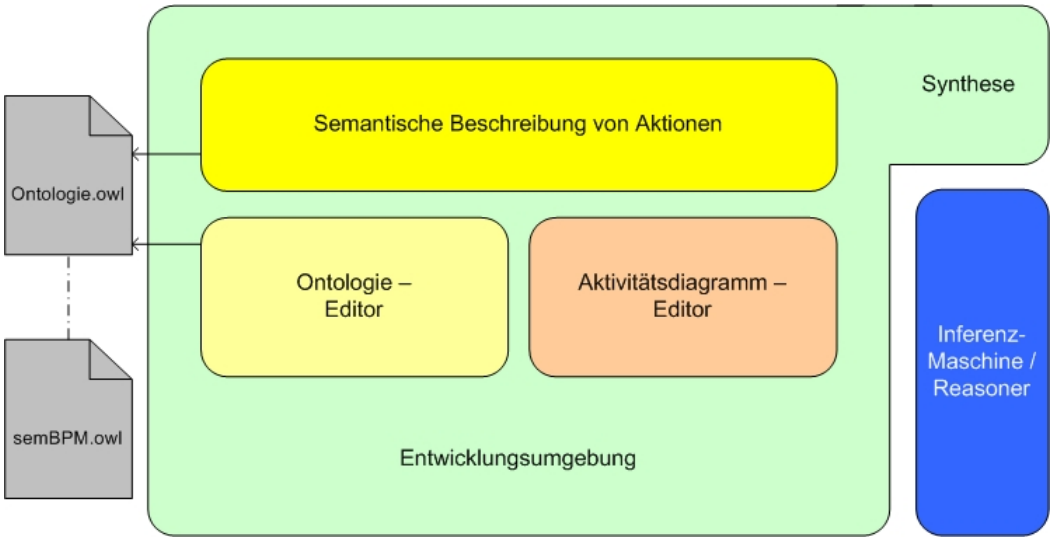


Abbildung 1.1: Konzeptioneller Aufbau des Prototypen

wird die Synthese mittels einer Inferenz-Maschine durchgeführt, mit deren Hilfe diverse Matrizen erstellt werden. Diese Matrizen sind die Grundlage für darauf aufsetzende Graphen-Algorithmen, die speziell an die Belange dieser Arbeit angepasst wurden.

Kapitel 5 widmet sich dem entwickelten Tool und stellt das zugrunde liegende Framework dieses Tools ebenso vor wie die Konzepte und den prinzipiellen Aufbau. Danach wird im Kapitel 6 eine Evaluation des Tools anhand verschiedener Beispiele vorgenommen und die Ergebnisse bewertet. Abschliessend zeigt Kapitel 7 einen Ausblick auf mögliche Erweiterungen des Tools.

Kapitel 2

Grundlagen

2.1 Modellierung von Geschäftsprozessen

2.1.1 Arten der Modellierung

Seit vielen Jahren beschäftigt sich die Informatik mit der Beschreibung von Prozessen. Begonnen wurde dies in den 70er Jahren durch Hoare und Milner, die sich mit der Frage beschäftigt haben, wie sich dynamische Systeme präzise (also in angemessener Syntax und formaler Semantik) so beschreiben lassen, dass man wichtige Eigenschaften wie z.B. Lebendigkeit berechnen kann. In der Wirtschaftsinformatik befasst man sich seit 1993 mit Geschäftsprozessen und deren Modellierung (meist auf Grundlage von Petri-Netzen). Bei einem Prozess handelt es sich dabei im wirtschaftlichen Sinne um eine

> „wiederholbare Folge von physischen oder informatorischen Tätigkeiten mit klar definiertem Input und Output [...]. Beispiele eines Prozesses sind beispielsweise „Brief schreiben“, „Termin vereinbaren“, „Akten ablegen“, u.v.m.“ [Sei02]

Ein Geschäftsprozess hingegen ist ein

> „Prozess mit hoher Wertschöpfung für den Kunden. In der Regel sind pro Unternehmen nur sehr wenige Prozesse von dieser wettbewerbskritischen Bedeutung vorzufinden - Beispiele: Auftragsbearbeitung, Produktentwicklung, Serviceleistungen“ [Sei02].

Die für die Geschäftsprozessbeschreibung in der Betriebswirtschaftslehre überwiegend verwendete natürliche Sprache besitzt aber Nachteile in ihrer fehlenden Eindeutigkeit, schwer nachvollziehbaren Vollständigkeit des dargestellten Sachverhalts und etwaiger Widersprüche. Verbale Beschreibungen sind aus diesem Grund für die Spezifikation von Informationssystemen oder Geschäftsprozessen nur bedingt geeignet. Die für Entscheidungs- und Planungsprobleme in der Betriebswirtschaftslehre verwendete mathematische Sprache ist zwar exakter und verifizierbar, aber nicht für alle Problemstellungen geeignet [Sch98]. Daher wurden im Laufe der Zeit immer mehr unterschiedliche Modellierungssprachen entwickelt: vom Handels-H-Modell über das Y-CIM-Modell, dem SCOR-Modell oder dem SAP R/3 Referenzmodell (eine ausführliche Übersicht bietet beispielsweise [FL04]). Mittlerweile sind unterschiedliche Modellierungsarten gängig: die am

häufigsten verwendeten Ansätze des Business Engineering sind ARIS (Architektur integrierter Informationssysteme) sowie ein Standard aus der Software-Entwicklungsbranche: die UML. Die von der Object Management Group (OMG) spezifizierte Unified Modeling Language (vgl. [OMG05]) nimmt in der neuen Version 2 nicht nur im Software-Engineering sondern auch in vielen anderen Bereichen [1] einen immer grösser werdenden Stellenwert ein.

2.1.2 ARIS

ARIS ist ein Architekturansatz, der verschiedene Modellierungssprachen zusammenführt [Sch98] [Sei02]. Es bildet die fachlichen Anforderungen auf verschiedenen Ebenen bis zur Implementierung von Informationssystemen ab. ARIS untergliedert die Aufgabenstellung in drei Ebenen: Fachkonzept, DV-Konzept und Implementierung. Es unterscheidet in der Modellierung unterschiedliche Sichten (Organisation, Funktion, Daten, Leistung, Steuerung), wobei der Schwerpunkt auf der Modellierung des Fachkonzeptes liegt. Hierfür werden vorrangig Funktionshierarchiebäume (Funktionssicht), Entity-Relationship-Modelle (Datensicht) und erweiterte ereignisgesteuerte Prozessketten (eEPKs auf Steuerungssicht) verwendet. Parallel zum Modellierungsansatz entwickelte *IDS Scheer* die Software ARIS Toolset. Der ARIS-Ansatz lässt sich daher als Modellierungssprache, Referenzmodell und Werkzeug vorwiegend auf Ebene der Geschäftsprozesse charakterisieren. ARIS hat sich zwischenzeitlich als Standard vor allem für grosse Unternehmen durchgesetzt [sB05].

2.1.3 UML2

Die Modellierungssprache Unified Modeling Language wurde ursprünglich für die Modellierung, Dokumentation, Spezifikation und Visualisierung komplexer Softwaresysteme erstellt. Sie umfasst die wichtigsten Aspekte verschiedener Ansätze um möglichst vielen Problemstellungen gerecht zu werden und von vielen anerkannt zu werden. Dabei handelt es sich bei UML nicht um eine Software-Methodologie, da dies eine Modellierungssprache zusammengenommen mit einem Software-Prozess wäre, vielmehr kann die UML in verschiedenen Methodologien angewendet und an unterschiedliche Bereiche angepasst werden ([Bau04]).
Ihre Gründung begann 1995, als sich James Rumbaugh und Grady Booch mit ihren Ansätzen Object Modeling Technique (OMT) und Object-Oriented Design (OOD) zusammentaten und zusammen die Unified Method gründeten. Durch die Hinzunahme von Object Oriented Software Engineering (OOSE), das von Ivar Jacobson 1992 erstellt wurde, entstand 1996 die erste Version der Unified Modeling Language. Diese Drei (später auch die 'drei Amigos' genannt) waren fortan für die Weiterentwicklung der UML verantwortlich, bevor 1999 die Object Management Group (OMG) diese Aufgabe ab der damaligen Version UML 1.3 übernahm. [WO04]
Im Laufe der Zeit wurden mehrere Zusätze in UML integriert, wie beispielsweise die Object Constraint Language (OCL), das auf der Extensible Markup Language (XML) basierende XML Metadata Interchange Format (XMI) oder auch die Meta Object Facility (MOF). Ausserdem finden sich mittlerweile auch andere Ansätze in der UML wie die Eingliederung von Zustandsdiagrammen oder die Darstellungsmöglichkeiten von Petri-Netzen.
Die neueste Version - UML 2.0 oder einfach UML2 - wurde teilweise bereits veröffentlicht, komplett aber erst im Laufe des Jahres 2005, und enthält gegenüber den Vorgängern einige Veränderungen. Die meisten Änderungen wurden dabei vom Konsortium U2 übernommen, das bestehend aus Alcatel, Computer Associates, Hewlett-Packard, Oracle, Rational Software und vielen anderen mehr,

[1] wie beispielsweise der Entwicklung von Unterhaltungselektronik durch Zustandsautomaten oder im Telekommunikationsbereich bei der Entwicklung von Kommunikationsprotokollen unter Zuhilfenahme von Timing-Diagrammen

den am besten ausgereiften Änderungsvorschlag der OMG übergab. Aus den vorliegenden Ideen entstanden im Wesentlichen zwei sich ergänzende und aufeinander verweisende UML2-Dokumente: die Infrastructure und Superstructure. In der Infrastructure sind grundlegende Sprachkonstrukte und die Basisarchitektur festgehalten, wohingegen die Superstructure aufbauend auf dieser Architektur Diagrammnotation und Semantik enthält.
Die UML2 besteht nunmehr aus 13 Diagrammtypen, sechs für die statische Struktur eines Systems und sieben Diagrammtypen für die Modellierung von dynamischen Aspekten. In der Version 2 der UML wurden einige Diagrammarten neu eingefügt (wie das Timing-Diagramm, das Interaktionsübersichtsdiagramm sowie das Kompositionsstrukturdiagramm), andere umbenannt (das ehemalige Kollaborationsdiagramm heisst jetzt Kommunikationsdiagramm) oder verbessert (wie z.B. das Aktivitätsdiagramm oder das Sequenz-Diagramm). Eine Zusammenfassung aller Änderungen und gute Dokumentation findet sich beispielsweise in [JRH+04]. Die offizielle Spezifikation der UML ist ein komplexes, über tausend Seiten umfassendes Werk. Die UML ist in Erfüllungsebenen (Compliance Levels) eingeteilt, um Werkzeugherstellern auch eine schrittweise oder selektive Implementierung von UML2 zu ermöglichen (Foundation, Basic, Intermediate und Complete).
Eine für uns wichtige Untergliederung von UML2 ist die 4-Schichten-Architektur der UML (siehe Abbildung 2.1). Modellebene 0 (M0) enthält die zu modellierende Wirklichkeit (in der angegebenen Grafik: Professor Bauer hält die Vorlesung Compilerbau). Auf Ebene M1 wird dies abstrakter beschrieben, nämlich, dass ein Professor beliebige Vorlesungen halten kann, wobei hier nicht festgelegt ist, welcher Professor und welche Vorlesung. Dies kann nur in den sogenannten Instanzen auf Ebene M0 geschehen. Eine weitere Ebene darüber (M2) enthält die Modellierungselemente, die auf Ebene M1 verwendet werden dürfen. So wird hier festgelegt, dass ein Professor und eine Vorlesung als Klasse modelliert werden soll und dazwischen Assoziationen existieren dürfen. Auf der obersten Ebene (M3) wird beschrieben, welche Elemente im UML-Metamodell auf Ebene M2 verwendet werden dürfen. Hier ist festgelegt, dass es Elemente wie Klassen oder Assoziationen gibt. Dadurch wird eine Möglichkeit geschaffen auch verschiedene Arten der Modellierung (z.B. UML oder ARIS) aufeinander abzubilden. Die M3-Ebene oder auch Meta-Metamodell-Ebene genannt, ist selbstbeschreibend, weshalb es keiner weiteren Ebene bedarf, um die Elemente dieser Ebene zu beschreiben [WO04].

Ein Metamodell (Ebene M2) beschreibt eine Abstraktionsebene über dem Modell aus welchen Elementen dieses Modell bestehen darf. Dabei wird ein Metamodell mittels eines UML-Klassenmodells beschrieben. Es ist auch möglich ein Metamodell eines Metamodells zu erzeugen, dieses wird dann Meta-Metamodell genannt (Ebene M3). Im UML Meta-Metamodell (Meta-Object Facility, MOF) werden die Grundkonstrukte eines UML-Metamodells beschrieben, welches dann gewöhnlich verfügbare UML-Modellierungs-Elemente wie z.B. 'Klasse' oder 'Assoziation' enthält. Im UML-Modell, welches sich auf dem Metamodell abstützt und z.B. Klassen oder Assoziationen verwendet, kann eine gewöhnliche Modellierung vorgenommen werden. Auf dieser Ebene können die oben angedeuteten Diagrammarten für einen speziellen Problemfall erstellt werden. Hier könnte also ein Aktivitätsdiagramm oder ein Klassendiagramm zu einem Geschäftsprozess und dessen Implementierung zu finden sein. In der untersten Stufe finden sich die UML-Instanzen, z.B. Herr Bauer als Instanz der Klasse Professor.

Die UML ermöglicht neben den Metamodellen die genauere Beschreibung von nicht modellierbaren Sachverhalten mittels rein textueller Beschreibung oder unter Zuhilfenahme einer formalen Sprache wie der Object Constraint Language (OCL).
Die oben erwähnte Meta-Object Facility MOF definiert dabei den Sprachschatz der Modell-Elemente als abstraktes Modell um den Austausch verschiedener Modelle zwischen unterschiedlichen Tools zu ermöglichen. Dabei definiert MOF nicht nur das Meta-Modell von UML, sondern auch von anderen Standards wie CWM (Common Warehouse Metamodel). Ausserdem definiert

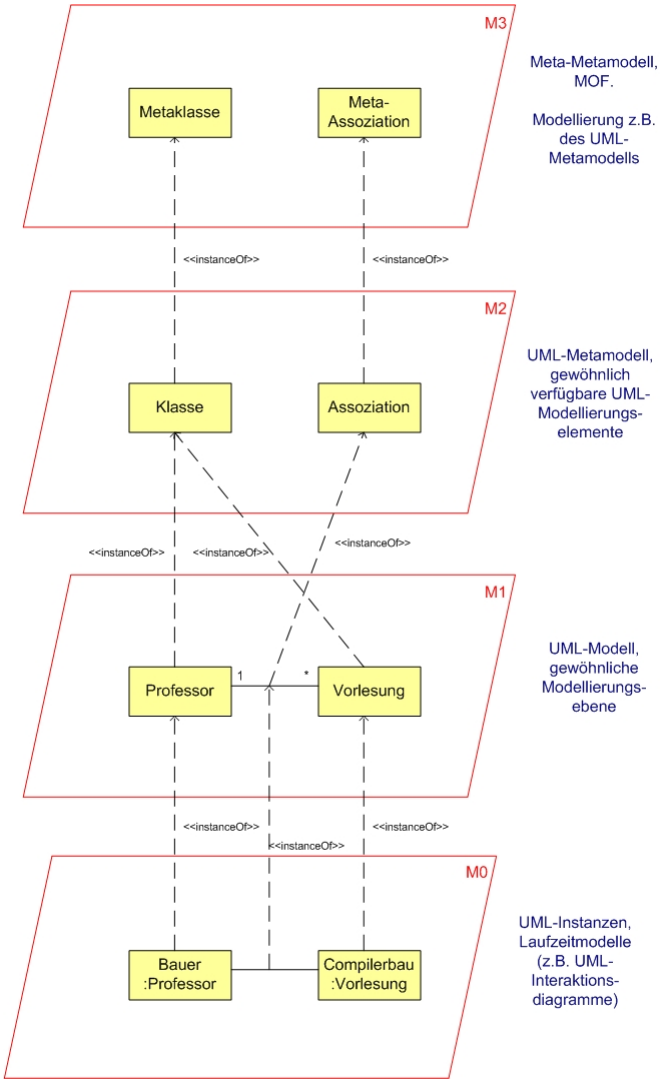


Abbildung 2.1: Die Schichtenarchitektur von UML2

die MOF Erweiterungsmechanismen, die benützt werden können um Metamodelle entweder zusammen mit UML-Profilen oder als Alternative dazu anzureichern [OMG03a]. Um den Austausch verschiedener Modelle zu erreichen, greift MOF auf XMI, den XML (Extended Markup Language) Metadata Interchange-Standard zurück. Dieses ist ein OMG-standardisiertes Format zum Modell und Metamodell-Austausch, welches auf XML zum Datentransport aufsetzt (mehr dazu unter [OMG03b]).
Bei allen Diagrammen (auch den sogenannten dynamischen Diagrammen) wird die Struktur oder der Ablauf starr dargestellt. Um Dynamik ausdrücken zu können, werden (wie beispielsweise Token im UML2-Aktivitätsdiagramm) eigene Konzepte eingeführt, die den konditionalen Ablauf darstellen sollen. Diese Token werden aber nur textuell beschrieben und können in den Diagrammen selbst nicht dargestellt werden.

2.1.4 Aktivitätsdiagramme

In UML2 finden sich verschiedene Diagrammtypen, die die Modellierung von dynamischen Sachverhalten ermöglichen. Das im Rahmen der Geschäftsprozessmodellierung am weitesten Genutzte ist sicherlich das Aktivitätsdiagramm [OWS+04]. Dieses Diagramm wird verwendet um die Zusammenhänge zwischen verschiedenen Aktionen zu verdeutlichen. Abbildung 2.2 zeigt ein kurzes Beispiel für ein Aktivitätsdiagramm: Ein Student muss sowohl Vorlesungen hören als auch Übungsblätter bearbeiten und im Anschluss daran eine Klausur über den Vorlesungsstoff schreiben. Dies wiederholt sich solange, bis alle notwendigen Prüfungen bestanden wurden. Dann kann, nach Abschluss der Diplomarbeit, das Studium beendet werden.

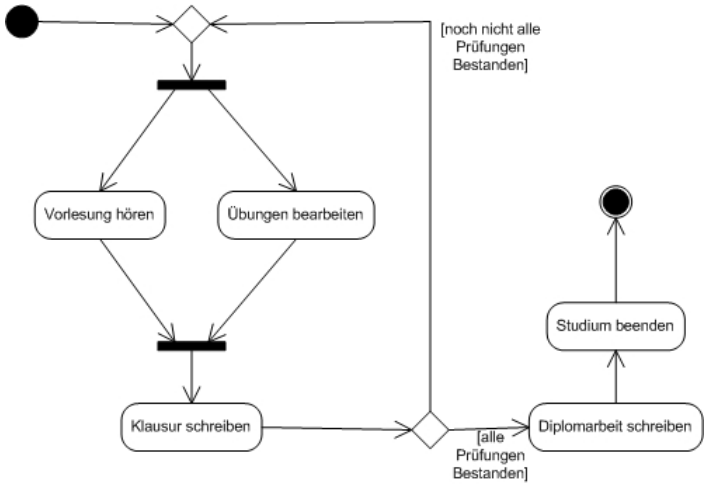
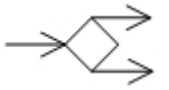
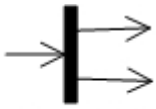


Abbildung 2.2: Ein sehr vereinfachter Vorgang des Studierens

Ein Aktivitätsdiagramm zeigt dabei einen möglichen Ablauf, wobei eine Aktivität die Gesamtheit aller Abläufe darstellt. In diesem Aktivitätsdiagramm befinden sich Aktionen, die je einen einzelnen Schritt in diesem Ablauf darstellen. Die Aktionen sind mit Kanten verbunden und werden durch Kontrollelemente gesteuert. Kontrollelemente ermöglichen die Parallelisierung und Synchronisation von Abläufen, bedingte Verzweigung und Zusammenführung, sowie Parametrisierung und Lenkung des Ablaufes.
Aktivitätsdiagramme gehen auf Struktogramme (Nassi-Shneiderman-Diagramme), Petri-Netze und Datenflussdiagramme zurück und wurden seit der letzen Version (1.5) der UML stark überarbeitet. Um den Ablauf einer Aktivität zu verstehen (nebenläufige Abläufe können sehr kompliziert werden), wurde bei Aktivitätsdiagrammen das logische Konzept der Token hinzugefügt (ähnlich wie bei Petri-Netzen). Dabei handelt es sich um einen Punkt (auch Marke genannt), an dem sich der Ablauf gerade befindet. Dieser Token wandert durch das Aktivitätsdiagramm und kann sich auch aufteilen. So können in einer Aktivität beliebig viele Token gleichzeitig unterwegs sein, um beispielsweise parallele Abläufe abzubilden. So gehen nach einem Parallelisierungs-Knoten (der die Parallelität einleitet, mehr dazu später) zu jeder Folgeaktion ein Token, so dass nach der Aufteilung die einzelnen „Teilabläufe“ unabhängig voneinander abgearbeitet werden können. Bei einer Verzweigung hingegen (für Alternativabläufe wie Ja/Nein-Entscheidungen etc.) liegt nur an einem der Folgeaktionen ein Token an, nämlich genau bei jenem, für welchen die Bedingung zutrifft.

Folgende Notationselemente stehen in einem Aktivitätsdiagramm zur Verfügung:

	Eine Aktion wird durch ein Rechteck mit abgerundeten Ecken dargestellt und enthält den Namen der Aktion. Ist diese Aktion an einem Objektfluss beteiligt, kann dies durch Pins (Rechtecke an den Kanten) dargestellt werden.
	Kanten sind gerichtete Übergänge zwischen zwei Knoten innerhalb einer Aktivität und können zum besseren Verständnis oder um Bedingungen auszudrücken (z.B. bei Entscheidungsknoten) mit einem Namen versehen werden.
	Ein Startknoten markiert den Startpunkt eines Ablaufs innerhalb einer Aktivität. In jeder Aktivität kann nur ein Startknoten existieren. Ein Startknoten kann beliebig viele wegführende Kanten besitzen.
	Ein Endknoten beendet die gesamte Aktivität, sobald er von einem Token erreicht wurde. Es können mehrere Endknoten für Aktivitäten existieren; in diesem Fall wird die Aktivität beendet, sobald *einer* der Knoten einen Token erhält.
	Ein Entscheidungsknoten spaltet eine Kante in mehrere Alternativen auf. Ein Token passiert nur eine ausgehende Kante und wird an diesem Verzweigungsknoten *nicht* dupliziert. Welche Kante gewählt wird, wird durch an die Kante annotierte Bedingungen festgelegt. Werden mehrere alternative Stränge wieder verbunden, wird der sogenannte **Verbindungsknoten** gewählt, der das selbe Aussehen wie ein Entscheidungsknoten hat.
	An einem Parallelisierungsknoten wird der eingehende Ablauf in mehrere parallele Abläufe aufgeteilt. Hier wird das eingehende Token dupliziert und wandert auf *jeder* ausgehenden Kante weiter. Werden parallele Kanten wieder zusammengeführt, benötigt man den **Synchronisationsknoten**, der das selbe Aussehen wie ein Parallelisierungsknoten besitzt.

Im Rahmen der Geschäftsprozessmodellierung werden die Konstrukte folgendermassen verwendet:

Aktion: Eine Aktion steht hierbei für einen Geschäftsteilprozess (z.B. die Bearbeitung von Daten) und stellt das zentrale Element eines Aktivitätsdiagrammes dar. Alle anderen Elemente

dienen nur dem Zweck, den Ablauf und die Kontrolle der aufeinander folgenden Aktionen zu steuern und deren Datenaustausch zu modellieren. Eine Aktion kann dabei mit Vor- und Nachbedingungen spezifiziert werden und auf Signale oder Ereignisse reagieren oder diese auslösen.

Aktivität: Die Gesamtheit aller Knoten und ihrer Verbindung wird mit Aktivität bezeichnet und stellt einen Geschäftsprozess dar. Es ist möglich Aktivitäten zu schachteln, so dass eine Aktion innerhalb einer Aktivität auf eine andere Aktivität verweist, was die Lesbarkeit eines Diagrammes stark erhöht. In einer Aktivität lassen sich sowohl Eingabe- als auch Ausgabe-Parameter festlegen, die in Form von Objekten realisiert sind.

Objektknoten: Ein Objektknoten innerhalb einer Aktivität repräsentiert Ausprägungen eines bestimmten Typs wie z.B. primitive Werte oder Objekte von Klassen. Damit lassen sich Ein- und Ausgabe einer Aktivität näher spezifizieren, als auch der Objektfluss zwischen zwei Aktionen. Objektknoten können explizit in einem eigenen Rechteck gezeichnet werden oder in Form von Pins an eine Aktion angehängt werden.

Kanten: Kanten sind Übergänge zwischen zwei Knoten (Aktionen, Objektknoten, etc.). Kanten sind dabei immer gerichtet und können mit einem Namen versehen sein. Es ist möglich Kanten mit Bedingungen zu belegen. Ein Übergang über diese Kante ist dann nur möglich, wenn die Bedingung erfüllt ist. Meist werden Bedingungen in Verbindung mit Verzweigungsknoten verwendet, um zu bestimmen, in welche Richtung der Ablauf fortgesetzt wird.

Entscheidungsknoten/Verbindungsknoten: Ein Entscheidungsknoten spaltet eine Kante in mehrere Alternativen auf. Bedingungen für diesen Verzweigungsknoten dürfen sich nicht überschneiden und es müssen alle Möglichkeiten berücksichtigt sein. Nach einer Verzweigung werden die Kanten über einen Verbindungsknoten wieder zusammengeführt. Unterscheidungen sind in jeder Art der Modellierung von Geschäftsprozessen denkbar.

Parallelisierungsknoten/Synchronisationsknoten: An einem Parallelisierungsknoten wird der eingehende Ablauf in mehrere parallele Abläufe aufgeteilt. Die duplizierten Knoten werden beim Auftreffen auf einen Synchronisationsknoten wieder zusammengeführt. Dabei werden die Knoten durch ein implizites UND getrennt und wieder zusammengefügt. Meist geschehen mehrere Geschäftsprozesse gleichzeitig, beispielsweise weil unterschiedliche Abteilungen an einem Prozess beteiligt sind (z.B. Produktion und Finanzbuchhaltung).

Startknoten/Endknoten: Ein Startknoten markiert den Startpunkt eines Ablaufes bei Aktivierung einer Aktivität, ein Endknoten beendet die gesamte Aktivität. Es dürfen beliebig viele Endknoten existieren, allerdings nur ein Startknoten.

Kontrollstrukturen: Desweiteren ist es möglich diverse Strukturelemente für die Modellierung von Algorithmen zu verwenden. So kann eine Auswahl mittels IF/THEN/ELSE ebenso dargestellt werden wie eine Wiederholung mittels WHILE/DO oder auch eine Mehrfachauswahl.

Eine genauere Beschreibung von Aktivitätsdiagrammen und deren Elemente findet sich unter anderem in [JRH+04].

2.2 Semantic Web Technologien

2.2.1 Grundlegendes

Die erste Definition von Semantic Web nahm Tim Berners-Lee in [BLHL01] vor. Dabei beschrieb er die Möglichkeit das World Wide Web so weiterzuentwickeln, dass es auch von Maschinen verstehbar ist. Dazu erstellte er einen „Semantic Web wedding cake“, der verschiedene zu erstellende Schichten definierte, von denen mittlerweile einige Schichten bereits als Standards vorliegen.
Um Informationen für Maschinen eindeutig interpretierbar darzustellen, kann man sich formallogischer Repräsentationsformalismen bedienen, wie sie im Bereich der künstlichen Intelligenz zur Wissensrepräsentation eingesetzt werden. Im Zusammenhang mit der semantischen Anreicherung von Information im Internet haben sich bisher v.a. sog. **Ontologien** bei der Beschreibung von Zusammenhängen durchgesetzt. Der Begriff der Ontologie kommt aus dem Griechischen und bedeutet „ons, ontos“ = sein und „logos“ = überlegen, also die „Lehre vom Sein“. Die für den Semantic Web Bereich am häufigsten zitierte Definition einer Ontologie von [Gru93] ist

> „An ontology is a formal explicit specification of a shared conceptualization.“

Man könnte eine Ontologie auch definieren als „eine hierarchisch strukturierte Menge von Ausdrücken zur Beschreibung einer Domäne, die als Grundgerüst für eine Wissensbasis dienen kann“ [SPKR96]. Eine etwas genauere Beschreibung liefert [UG96]: „Eine Ontologie kann verschiedene Formen haben, aber notwendigerweise beinhaltet sie ein Vokabular von Ausdrücken und eine Spezifikation ihrer Bedeutung. Dies beinhaltet ihre Definition und Hinweise, wie die Konzepte untereinander in Beziehung stehen, was zusammen der Domäne eine Struktur auferlegt und die möglichen Interpretationen der Ausdrücke begrenzt.“

Ontologien beschreiben demnach auf formale und explizite Weise Konzepte und deren semantische Beziehungen in einer bestimmmten Anwendungsdomäne. Ontologien können so von verschiedenen Akteuren als gemeinsame Wissensbasis zum Austausch von Informationen verwendet werden. In [NM01] werden eine Reihe von sich hieraus ergebenden Möglichkeiten bzw. Vorteilen aufgeführt:

1. Verwendung eines gemeinsamen Vokabulars in verteilten Ressourcen. Werden bei der Beschreibung von Wissen z.B. auf verschiedenen Webseiten Konzepte einer gemeinsamen Ontologie verwendet, können Softwareagenten diese Informationen extrahieren, zueinander in Beziehung bringen und verarbeiten, da die Semantik der Informationen eindeutig in der Ontologie festgelegt ist.

2. Wiederverwendung von Domänenwissen. Bei der Definition von neuen Ontologien können bereits bestehende Ontologien wiederverwendet werden. Dadurch wird ermöglicht, dass einmal getane Arbeit beim Erstellen einer Ontologie nicht wiederholt werden muss.

3. Trennung von Programmcode und Domänenwissen. Durch die Möglichkeit zur deklarativen Beschreibung kann ein großer Anteil des implizit in der Anwendung codierten Wissens extrahiert werden. Änderungen des Domänenwissens ziehen so nicht unmittelbar die Änderung des Programmcodes nach sich.

4. Analyse des Domänenwissens. Basierend auf der formalen Semantik des Repräsentationsformalismus kann in Ontologien beschriebenes Wissen auf bestimmte Eigenschaften wie z.B. Konsistenz untersucht werden. Dies ist besonders wichtig bei der Wiederverwendung und Erweiterung von Ontologien.

2.2.2 Semantic Web Standards

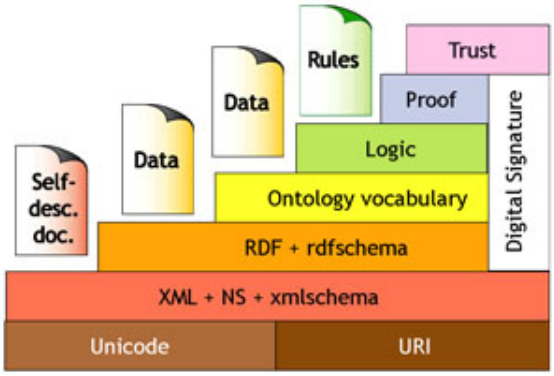


Abbildung 2.3: Die Schichtenarchitektur des Semantic Web [W3C04b]

Abbildung 2.3 zeigt den groben Aufbau der Semantic Web Schichten. Die unterste Stufe beschreibt den Zeichensatz (Unicode) und die Art wie Referenzen aufgelöst werden sollen. Dies geschieht mittels Unified Resource Identifiers (kurz: URI) bzw. dem Nachfolgestandard IRI (= Internationalized Resource Identifier [DS05]). Eine Spezialform stellen die aus dem Internet-Bereich bekannten Unified Resource Locators (URLs) wie z.B. *http://www.uni-augsburg.de* dar. Darauf aufbauend befindet sich die XML-Schicht. Hier setzt die im Nachfolgenden näher beschriebene RDF und RDF(S)-Schicht auf, die durch das Ontologie-Vokabular (speziell: OWL) erweitert wird. Darüber befinden sich die Stufen der Logic (in der sich aufkommende Standards wie die unten beschriebene SWRL befinden), sowie Proof und Trust mit z.B. Friend-Of-A-Friend (FOAF) als Standard. Parallel dazu muss mittels digitalen Signaturen gesichert werden, dass die angegebenen Daten nicht verändert wurden.

XML:

Die Grundlage aller Standards im Semantic Web-Umfeld nimmt die Extensible Markup Language (XML) ein. XML war der erste Sprachstandard im Internet, der versucht hat den Inhalt einer Webseite von der Präsentation des Inhaltes zu trennen. XML ist nicht notwendigerweise mit HTML verbunden, da beide für komplett unterschiedliche Zwecke entwickelt wurden. Davon abgesehen können sich beide ergänzen, um die Erfordernisse des Benutzers zu erfüllen. Um den Inhalt allein zu beschreiben, spezifiziert XML eine baumartige Struktur der Datei, um Zusammenhänge und Verbindungen darzustellen. XML definiert aber leider keinerlei Semantik, so dass der Ausdruck <BANK> sowohl ein Geldinstitut als auch eine Sitzgelegenheit meinen kann. Ausserdem ist es für einen Computer unmöglich zu folgern, dass beispielsweise „Ablaufdatum“ und „Verfallsdatum“ über dasselbe Konzept sprechen. Aus diesem Grund wurde nach einer weiteren Stufe gesucht, die folgende Anforderungen erfüllen sollte:

- Konzepte: Es muss eine genaue Spezifikation von Konzepten möglich sein. Unterschiedliche Konzepte mit gleichem Namen (vgl. <BANK>) müssen auseinander gehalten werden können.
- Verbreitetes Wissen: Im Internet befindet sich nicht eine globale Wissensbasis, sondern viele verschiedene Personen sowie Server mit unterschiedlichem Wissen und Fähigkeiten.

- Keine universelle Wahrheit: Es muss möglich sein Aussagen abzulehnen bzw. zurückzunehmen oder zu kombinieren, falls verschiedene Begriffe für eine Sache bestehen.
- Verschiedene Nutzergruppen: Da viele verschiedene Benutzer aus unterschiedlichen Fachrichtungen zusammentreffen muss der Standard sowohl leicht erweiterbar als auch einfach gehalten sein.

RDF:

Die oben genannten Anforderungen werden durch die auf XML aufsetzenden Standards RDF und OWL erfüllt. Bei dem vom World Wide Web Consortium (W3C) entwickelten RDF, dem Resource Description Framework [W3C04c] handelt es sich um einen Standard um Metadaten beschreiben zu können mit dem Ziel dem Web eine formale Semantik hinzuzufügen. So können Beziehungen zwischen Resourcen (alles was mittels eines Universal Resource Identifier, URI, genannt werden kann) in Form von Aussagen (Statements) bestehend aus Subjekt, Prädikat und Objekt beschrieben werden. RDF bietet nicht nur eine Möglichkeit den Typ zu spezifizieren (z.B. Statement), sondern auch Zusammengehörigkeiten zu beschreiben. Dafür existieren die drei RDF-Container rdf:Bag (ungeordnete Liste), rdf:Seq (eine geordnete Liste) oder rdf:Alt (für eine Liste von Alternativen). Ein einfaches Beispiel zeigt die Abbildung 2.4:

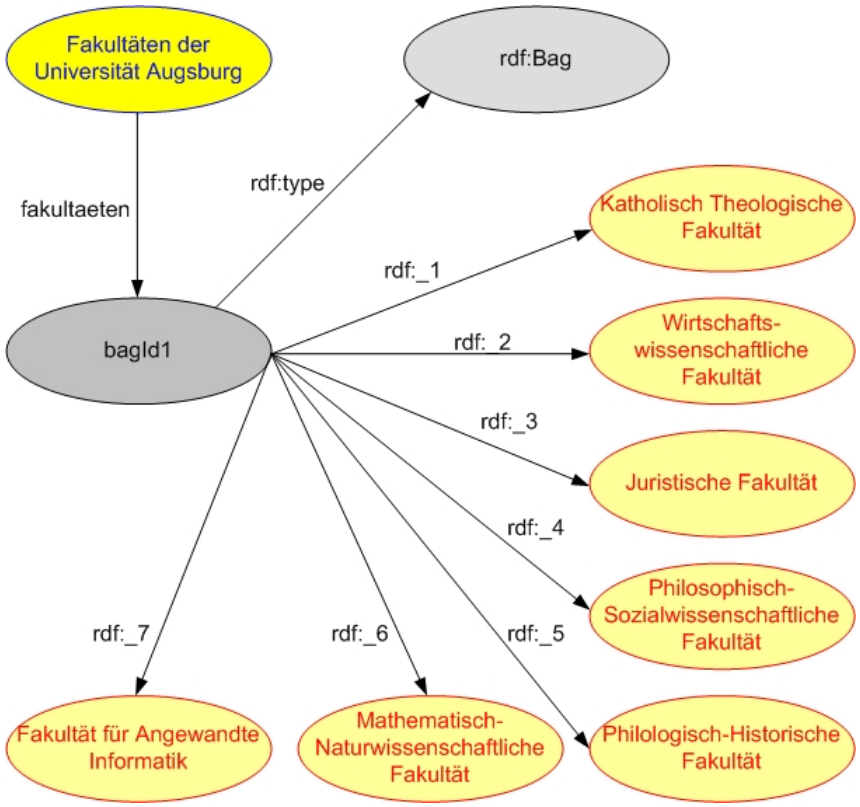


Abbildung 2.4: Die verschiedenen Fakultäten der Universität Augsburg

RDF Schema:

RDF Schema (RDFS oder auch RDF-S) definiert auf der Basis von RDF spezielle Typen von Ressourcen wie z.B. Klassen oder Properties, die es ermöglichen Vererbungsbeziehungen mittels rdfs:subClassOf, rdfs:subPropertyOf, usw. und einfache Beschränkungen für Properties (über welche Klassen sie sprechen) darzustellen (rdfs:domain, rdfs:range, usw.). Dadurch ist es möglich Gruppen von zusammengehörigen Resourcen und die Beziehungen zwischen diesen zu beschreiben. RDF Schema unterstützt RDF in derselben Weise wie DTDs und XML Schema dies für XML tun [SIC05].

OWL:

OWL setzt auf RDF, RDF Schema und XML auf. Bei OWL handelt es sich um die von der W3C spezifizierte Web Ontology Language (In Anerkennung an die Figur Owl aus „Winnie the Pooh“ wurde der Name von WOL zu OWL verändert [Wik05]). OWL ist der Nachfolger von DAML+OIL, einem Zusammenschluss des amerikanischen DARPA-Projekts DAML mit dem EU-Projekt OIL und wird seit 2004 vom W3C als Standard empfohlen [W3C04b].
Mit OWL ist es unter anderem möglich die Gleichheit (owl:equivalentClass oder owl:equivalentProperty) oder auch den Unterschied von Klassen oder Properties zu definieren. Ausserdem lässt sich die Kardinalität von Properties oder die Art (Symmetrisch, Funktional, Transitiv) der Properties genauer spezifizieren. Zusätzlich können Ontologien explizit beschrieben werden (Versionsstand, Import von anderen Ontologien, etc.).

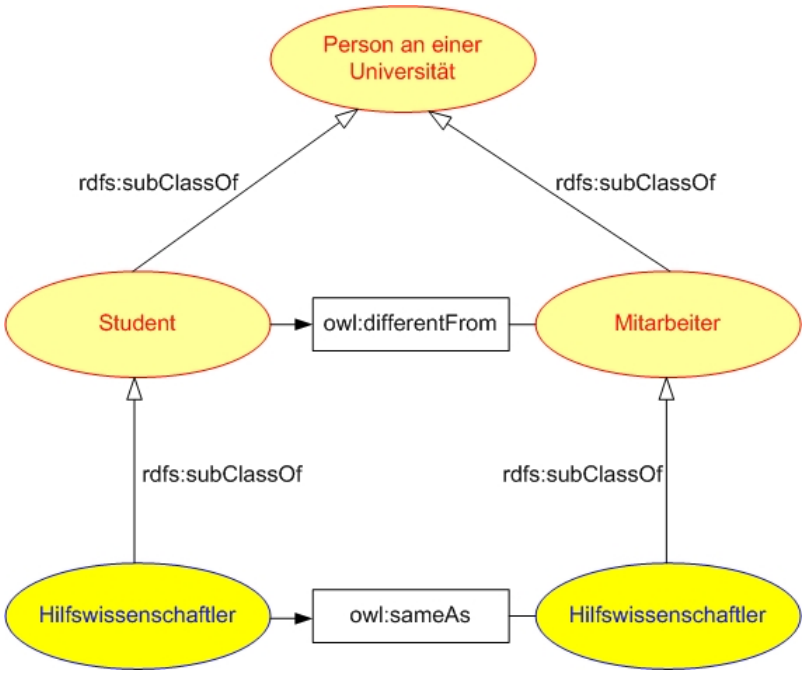


Abbildung 2.5: Ein kurzes Beispiel für OWL: die Position eines Hilfswissenschaftlers

OWL untergliedert sich in drei verschiedene Stufen:

OWL Lite: bietet eine Klassifikationshierarchie inkl. einfacher Beschränkungsmöglichkeiten und ist daher leichter von Tools und Werkzeugen zu implementieren.

OWL DL: ist eine Erweiterung von OWL Lite und bietet eine maximale Ausdrucksstärke, lässt aber gleichzeitig eine Beweisbarkeit mittels Inferenzmaschinen zu. Das Kürzel DL kommt dabei von der Beschreibungslogik (Description Logic), dem speziell finit entscheidbaren Bereich der Logik erster Stufe. In dieser Stufe findet eine Typseparation statt, hier können Klassen nicht gleichzeitig Instanzen und Eigenschaften sein.

OWL Full: bietet als Erweiterung von OWL DL eine maximale Ausdrucksstärke sowie die syntaktische Freiheit von RDF und RDFS, allerdings ohne eine Garantie der Auswertbarkeit mittels Inferenzmaschinen.

OWL Lite und OWL DL lassen sich beide der Beschreibungslogik zuordnen, wobei OWL Lite eine Untermenge ($\mathcal{SHIF}(\mathcal{D})$) von OWL DL ($\mathcal{SHOIN}(\mathcal{D})$) ist. $\mathcal{SHIF}(\mathcal{D})$ und $\mathcal{SHOIN}(\mathcal{D})$ beschreiben die Fähigkeiten der jeweiligen Sprache, wobei jeder Buchstabe für eine eigene Eigenschaft steht:

- $\mathcal{S}$ steht als Abkürzung für $\mathcal{ALCR}^+$. Dies setzt sich zusammen aus:
- $\mathcal{AL}$ beschreibt Ausdrücke, die nur eine kleine Menge von Operatoren benötigen: atomare Negation- oder Komplementbildung, Schnittmengenbildung, Wertebeschränkung von Rollen, eingeschränkte Existenzquantoren,
- $\mathcal{C}$ steht für die volle Negation eines Ausdrucks,
- $\mathcal{R}^+$ für transitive Rollen,
- $\mathcal{E}$ bedeutet volle Existenzquantoren ohne Einschränkungen,
- $\mathcal{H}$ Hierarchie von Rollen,
- $\mathcal{I}$ inverse Rollen,
- $\mathcal{N}$ offene Zahleinschränkungen (z.B. < 5),
- $\mathcal{F}$ funktionale Zahlbeschränkungen (z.B. 0 oder 1),
- $\mathcal{O}$ aufzählbare Klassen,
- $\mathcal{U}$ Vereinigungskonstruktor,
- ($\mathcal{D}$) Datentypen [HE04]

Dementsprechend besitzt OWL DL (im Gegensatz zu OWL Lite) zusätzlich die Funktionalität der aufzählbaren Klassen. Ausserdem können in OWL DL Zahlen nicht nur explizit mit festen Werten beschrieben werden, sondern es können auch Bereiche vorgegeben werden, in denen die Zahl liegen soll. OWL Lite ist entscheidbar in exponentieller Zeit (ExpTime), OWL DL in nicht-exponentieller Zeit (NExpTime), Konstrukte in OWL Full hingegen können unentscheidbar sein.

Logic:

Momentan werden zusätzliche Sprachen entwickelt um weitere Stufen der Semantic Web Vision zu verwirklichen. So wird zum Beispiel eine Regel-Sprache die Möglichkeit bieten bestimmte logische Beziehungen in einer Form auszudrücken, die von Maschinen verstanden werden können. Diese Sprache (auch als Rule Language bezeichnet) wird die Berücksichtigung von Entscheidungsregeln einer Firma ermöglichen und dadurch eine bessere Folgerung von Zusammenhängen. Erste Entwürfe einer Rules Language wie RuleML (Rule Markup Language) oder der ORL (Ontology Rules Language) wurden in einem bei der W3C zur Prüfung eingereichten Standard namens SWRL (Semantic Web Rules Language) [W3C04d] berücksichtigt. Als Konkurrent zeichnet sich hier der Standard WRL (Web Rules Language)[dB05] ab, der ebenfalls auf RuleML aufbaut und im Rahmen des WSMO (Web Services Modeling Ontology)- Projekts (siehe Kapitel 2.3.3) entwickelt wird. Eine Regel wird in SWRL in High-Level-Syntax repräsentiert und hat jeweils ein Antecedent und ein Consequent (eine Bedingung, welche erfüllt sein muss, sowie eine Schlussfolgerung). SWRL unterstützt Horn-Logik-ähnliche Regeln für OWL DL und OWL Lite. Um die Regeln von SWRL auswerten zu können, benötigt man wieder eine Inferenz-Maschine (siehe Kapitel 4.2), welche vermutlich in naher Zukunft angeboten werden wird. [Agg04] (aufgrund des frühen Entwicklungsstadiums - momentan existiert von SWRL die Version 0.7 vom Dezember 2004 und von WRL die Version 1.0 seit Juni 2005 - wird auf diese Standards im Folgenden nicht näher eingegangen).

Proof & Trust:

Jeder Mensch kann im Semantic Web Aussagen treffen. Eine Aussage könnte „ein Hilfswissenschaftler ist ein Mitarbeiter“ ebenso sein wie „ein Hilfswissenschaftler ist kein Mitarbeiter“. Nun wären zwei gegensätzliche Aussagen gegeben und ein auswertender Computer könnte über einen Hilfswissenschaftler alles schliessen oder gar keine Schlussfolgerungen mehr treffen. Aus diesem Grund muss es eine Grundlage geben, welchen Aussagen man eher vertraut als anderen. Um eine Lösung für dieses Problem zu finden, wurde der Standard FOAF (Friend-Of-A-Friend) [BM05] entwickelt. In diesem ist es möglich persönliche Daten über sich selbst zu speichern (Vorname, Nachname, Mail-Adresse, Homepage, etc.), ebenso welche Personen man kennt und welchen man traut. Dadurch entwickelt sich ein Netzwerk von Personen, denen man selbst traut sowie der Personen, denen die mir bekannten Personen trauen. Wird eine Aussage gefunden, wird in dem sich bildenden „Web of Trust“ gesucht, wie lange eine Kette von Menschen wäre, die von mir zu dieser Person führt [2]. Unterschreitet dieser Wert eine von mir gewählte Grenze, vertraue ich dieser Aussage, ansonsten traue ich ihr nicht [Dum02]. Zusätzlich muss es möglich sein, die von mir erstellten Aussagen zu signieren (mittels Hash-Werte), aber auch meine semantischen Informationen (bzw. Teile davon) zu verschlüsseln, wie dies beispielsweise in [Gie05] beschrieben ist.

2.3 Semantic Web Services

2.3.1 Web Service Standards

Web Services sind in sich geschlossene Einheiten, die auf Wunsch Dienste durch wohl definierte Schnittstellen zur Verfügung stellen [JUG03]. Die Definition von Sun [Sun03] stellt Web Services wie folgt dar:

[2] Gemäss der „Small World Hypothesis“ von Stanley Milgram (1967) kennt jeder Mensch über durchschnittlich sechs Menschen jeden anderen Menschen [Uni05].

> „An application that exists in a distributed environment, such as the Internet. A Web service accepts a request, performs its function based on the request, and returns a response. The request and the response can be part of the same operation, or they can occur separately, in which case the consumer does not need to wait for a response. Both the request and the response usually take the form of XML, a portable data-interchange format, and are delivered over a wire protocol such as HTTP.“

Zusammengefasst sind Web-Services abgeschlossene, beliebig komplexe und selbsterklärende Software-Komponenten - beschrieben mittels einer formalen Definition (im XML-Format) - die über das Internet (oder andere Middleware) veröffentlicht, gesucht und gefunden werden können und die mittels eines Standard-Protokolls (http, smtp) mit anderen Komponenten verbunden werden können. Dafür nutzt ein Web-Service gemäss dem SOA-Prinzip (Service Oriented Architecture) folgende Standards:

WSDL:

Die Web Service Description Language (aktuell in Version 1.1) ist eine XML-Grammatik zur Beschreibung von Web Services als eine Menge von Zugriffsendpunkten, die Nachrichten auf prozedur- oder dokumentorientierte Weise austauschen können. Die Operationen und Nachrichten werden abstrakt beschrieben und an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden, um einen Zugriffsendpunkt zu beschreiben.

SOAP:

Das Simple Object Access Protocol stellt die am häufigsten gebrauchte Möglichkeit dar, eine XML-basierte Nachrichtenschicht umzusetzen. Dabei handelt es sich um ein Kommunikationsprotokoll für den Dokumentenaustausch über ein Netzwerk. SOAP wurde maßgeblich von IBM und Microsoft entwickelt und dann dem W3C übergeben und liegt aktuell in der Version 1.2 vor. Die SOAP-Spezifikation beschreibt im Wesentlichen vier Komponenten für die Anatomie einer SOAP-Nachricht:

- Formatierungskonventionen für das Einkapseln von Daten und Routing-Informationen in der Form eines Envelopes
- Die Protokollbindung definiert eine Konvention zum Austausch von SOAP-Envelopes über ein Protokoll niedrigerer Ebene.
- Codierungsregeln definieren Konvertierungsregeln für die Datentypen verschiedener Anwendungen in eine XML-Tag basierte Darstellung.
- Der RPC-Mechanismus ermöglicht die Darstellung von entfernten Prozeduranrufen und deren Rückgabewerte.

UDDI:

Das Projekt Universal Description, Discovery and Integration (UDDI) ist eine Industrie-Initiative, deren Ziel es ist, die Interoperabilität und den Einsatz von Web Services zu erhöhen. Dazu stellt UDDI standardisierte Spezifikationen zur Beschreibung, dem Auffinden und der Integration von

Diensten bereit. UDDI liegt momentan in Version 3.0 vor.

Die oben genannten Standards legen einen Grundstein für die Kommunikation von einzelnen Web-Services miteinander. Sie erlauben aber noch keine Abhandlung von (meist komplexen) Geschäftsprozessen. Dafür müssen standardgemäss viele Web-Services miteinander in fest vorgelegtem Ablauf kommunizieren. Es muss daher möglich sein auf Rückgaben oder eine Fehlermeldung eines Service dynamisch zu reagieren. Aus diesem Grund haben sich im Bereich der Web Service Choreographie verschiedene Standards gebildet, die allesamt auf den oben genannten Standards aufsetzen. Der bekannteste unter ihnen ist WS-BPEL, die *Web Service Business Process Execution Language* (vormals BPEL4WS, also *Business Process Execution Language for Web Services* genannt).

Aufbauend auf den momentanen Standards für Web Services befinden sich die Semantic Web Services, die die Funktionen eines Web Services in maschinen-verständlicher Sprache beschreiben. Semantic Web Services bauen dabei gleichermassen auf den Standards des Semantic Web-Umfelds (RDF, OWL) wie auch des Web Service Bereichs (SOAP, WSDL, WS-BPEL, etc.) auf. Abbildung 2.6 zeigt das Stufen-Konzept von Semantic Web Services. Dabei stellt die linke Säule die Standards aus dem Bereich der Web-Services und die rechte Seite die Standards aus dem Semantic Web-Bereich dar. Semantic Web Services bauen gleichermassen auf beiden Säulen auf, wobei zwischen den auf derselben Höhe befindlichen Stufen der Säulen kein Zusammenhang besteht.

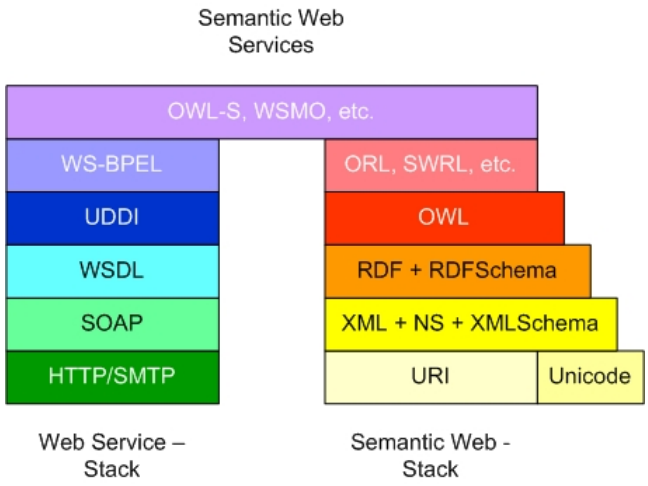


Abbildung 2.6: Der Semantic Web Service - Stack

Im Bereich der Semantic Web Services herrscht im Moment noch ein „Sprachen-Wirrwarr“, da sehr viele verschiedene Standards bei unterschiedlichen Gremien zur Prüfung eingegeben wurden. Die folgenden Abschnitte erläutern die verschiedenen Vorschläge:

2.3.2 OWL-S

Der bekannteste Standard im Semantic Web Services Umfeld ist OWL-S (genauer Titel: Semantic Markup for Web Services). Dies baut wie im Namen bereits ersichtlich auf der Semantic Web Sprache OWL (sowie auf SWRL) auf und befindet sich momentan in Version 1.1b bei der W3C zur Standardisierung [W3C04a]. OWL-S ist eine Weiterentwicklung des früheren DARPA-Standards DAML-S und basiert wie dieser auf einem Service der aus einem

- ServiceProfile („Was macht der Service?“)
- ServiceGrounding („Wie spricht man den Service an?“) und einem
- ServiceModel („Wie arbeitet der Service?“) besteht.

Zur genaueren Spezifizierung wie ein Service arbeitet, werden einzelne Services mittels Parameter spezifiziert: **Input** definiert die Eingaben (Objekte), die ein Service erhält, **Output** die Ausgaben eines Services, **Precondition** alle Vorbedingungen, die ein Service erfüllen muss, um überhaupt aufgerufen werden zu können sowie **Effect** um alle Effekte und Nachbedingungen, die eine Ausführung des Services mit sich bringt, definieren zu können. Diese Vier werden oft auch einfach mit **IOPE** abgekürzt.
In OWL-S werden drei Arten von Prozessen unterschieden: Atomare Prozesse, simple Prozesse sowie zusammengesetzte Prozesse (Composite Processes). Atomare Prozesse (oder Atomic Processes) können direkt aufgerufen und ausgeführt werden und greifen auf keine anderen Web Services zurück. Simple Prozesse hingegen dienen nur der Abstraktion und können nicht direkt aufgerufen werden. Sie können sowohl für einen atomaren Prozess als auch für einen Composite Process stehen. Ein zusammengesetzter Prozess kann ebenfalls direkt aufgerufen werden, benötigt aber andere Web Services für seine Ausführung.

ServiceModel: Ein Service wird durch das ServiceModel beschrieben. Das ServiceModell hat eine Unterklasse Process mit diversen weiteren Unterklassen (AtomicProcess, Simple Process, Composite Process wie oben beschrieben). Dabei beschreibt die Prozess-Klasse zusammen mit ihren Subklassen wie der Service funktioniert, bzw. welche Parameter vorliegen. Diese Klasse ist die Basis für eine Web Service Komposition.

ServiceGrounding: Alle Prozesse definieren im ServiceGrounding die Beziehung zu WSDL, um von anderen Web Services aufrufbar zu sein. Der Service unterstützt das ServiceGrounding mithilfe der Unterklasse Grounding um ein Mapping des abstrakten (OWL-S) ServiceModel zu einer konkreten Spezifikation zu unterstützen. Dabei handelt es sich um eine WSDL Beschreibung, wobei OWL Klassen zu WSDL Typen umgewandelt werden, Prozess Input/Output zu WSDL Input/Output, Message Parts sowie Atomare Prozesse direkt in WSDL-Operationen. Ein WebService kann mehrere Groundings haben, ist also nicht an eines speziell gebunden. [LRPF04]

ServiceProfile: Die oben genannten IOPEs finden sich in einem Service sowohl im ServiceProfile, als auch im ServiceModel, wobei die IOPEs des ServiceProfile eine Untermenge der IOPE´s des ServiceModel sein müssen. Ein ServiceProfile hat eine Unterklasse Profile, welches das Vokabular sowohl für funktionale Eigenschaften (IOPEs) als auch für nicht-funktionale Eigenschaften (wie z.B. Kontaktinformationen, Servicekategorien, etc.) enthält. Gedacht ist dieses nur für die automatische Suche von Web Services, nicht zur Komposition von Web Services [Mur04].

In OWL-S ebenfalls übernommen wurden die Kontrollstrukturen, die bereits aus Web Service Orchestration Standards wie WS-BPEL bekannt waren (Sequenz, Split, Join, Choice, If-Then-Else, etc.). Damit ist es möglich einen Ablauf von einzelnen Prozessen zu steuern: sollen diese hintereinander (Sequenz), parallel zu einander (Split, Join) oder alternativ (Choice, If-Then-Else) verwendet werden.

2.3.3 WSMO

Ein weiterer bekannter Standard von DERI International (Digital Enterprise Research Institute, http://deri.org) im Bereich der Semantic Web Services ist die Gruppe um WSMO. WSMO bedeutet Web Service Modeling Ontology (WSMO) und baut auf dem WSMF-Vorschlag von [FB02] auf. Sie benützt die WSML (Web Service Modeling Language), die unter anderem als Grundlage für die WRL (Web Rules Language) dient.
WSMO sieht sich selbst als Meta-Modell für Semantic Web Service-bezogene Aspekte und wurde durch MOF erstellt. WSMO besteht aus 4 verschiedenen Hauptelementen: Ontologien (für die Terminologie für die anderen Elemente), Ziele (für die Problembeschreibung, die von den Web Services gelöst werden soll), Web Service Beschreibungen (die versch. Aspekte von Web Services beschreiben) und Mediatoren (um Interoperabilität zwischen Systemen zu gewährleisten) [RLK04] (siehe dazu auch Abbildung 2.7).

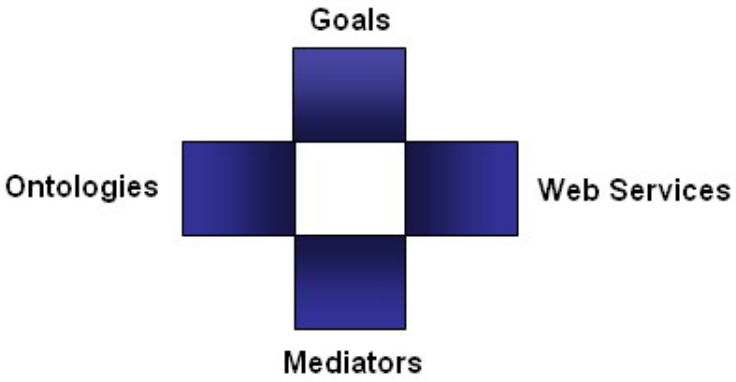


Abbildung 2.7: Die 4 Hauptelemente von WSMF bzw. WSMO [FB02]

WSMO definiert ein Konzept, welches in OWL-S nicht vorhanden ist: Mediatoren. Diese Mediatoren sind verantwortlich um die restlichen Hauptelemente zu verbinden. Dabei werden vier Arten von Mediatoren unterschieden: die ersten beiden in der Klasse der „Refiners“ ermöglichen eine neue Komponente auf der Basis von vorhandenen Komponenten (engl. to refine), um speziell die Wiederverwendung zu erleichtern. Bei den beiden Refiner-Mediatoren handelt es sich um den ggMediator (goal to goal), der zwei Ziele miteinander verbindet, sowie den ooMediator (ontology to ontology), der die Zusammenhänge zweier Ontologien definiert. Desweiteren gibt es noch zwei sogenannte „Bridges“, die die Verbindung von zwei Komponenten für die Interaktion zwischen beiden definiert: wgMediators (web service to goal), um Web Services mit Zielen zu matchen und wwMediators (web service to web service), um verschiedene Web Services miteinander zu verknüpfen.
WSMO baut nicht auf dem vorher genannten Web-Service-Stack auf, sondern nutzt die Sprache WSML (Web Service Modeling Language) und bietet eine Referenz-Implementierung mit dem Namen WSMX (Web Service Modeling eXecution). WSML gibt es in fünf verschiedenen Stufen [dBLPF05]. Den Zusammenhang zwischen den unterschiedlichen Stufen verdeutlicht Abbildung 2.8:

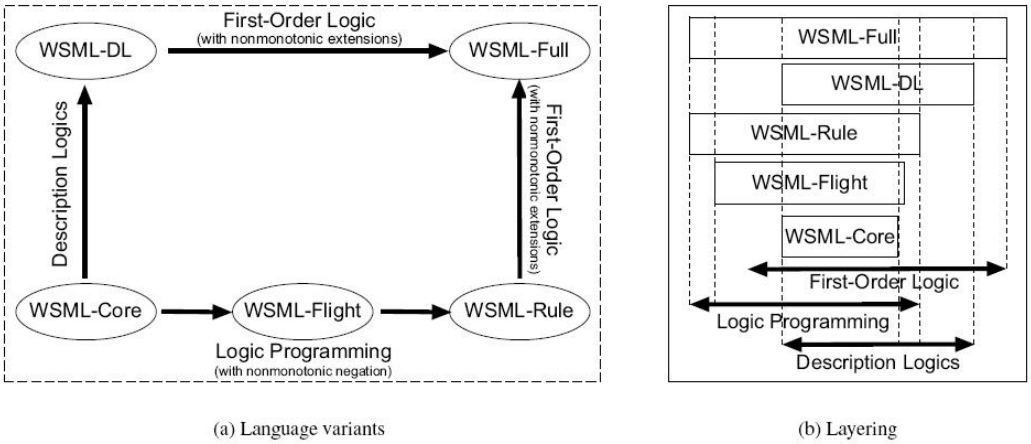
WSML-Core: besitzt die geringste Ausdrucksstärke aller WSML-Varianten. Die Hauptbestandteile sind Konzepte, Attribute, binäre Beziehungen und Instanzen, ebenso wie Konzept- und Beziehungshierarchien und Unterstützung von Datentypen.

WSML-DL: baut auf Description Logic ($\mathcal{SHIF}(\mathcal{D})$) auf, welches den grössten Teil von OWL-DL ausmacht. WSML-DL beinhaltet ausserdem eine Erweiterung der Datentypen.

WSML-Flight: erweitert WSML-Core und stellt eine mächtige Regelsprache zur Verfügung mit zusätzlichen Möglichkeiten wie Meta-Modellierung, Bedingungen und nicht-monotoner Negation. WSML-Flight basiert auf einer Logik-Programmierungsvariante von F-Logic und ist semantisch äquivalent mit Datalog.

WSML-Rule: erweitert WSML-Flight um zusätzliche Möglichkeiten der Logik-Programmierung wie der Benutzung von Funktionssymbolen oder unsicheren Regeln. In diesem Bereich findet sich auch die Regelsprache WRL.

WSML-Full: vereint WSML-DL und WSML-Rule unter dem Mantel der Prädikatenlogik erster Stufe.



(a) Language variants (b) Layering

Abbildung 2.8: WSML Varianten und Stufen [dBLPF05]

Die *Preconditions* und *Effects* von OWL-S werden in WSMO als *Assumptions* und *Effects* bezeichnet, dabei haben die Parameter weniger Beschränkungen als in OWL-S [LRPF04]. WSMO bietet im Gegensatz zu OWL-S folgende Eigenschaften: nicht-funktionale *Properties* (in OWL existiert nur die *FunctionalProperty*), eine Beschreibung von globalen Zielen ohne Rücksicht auf Preconditions oder Inputs nehmen zu müssen, das oben genannte Mediatoren-Konzept, eine Fehlermodellierung mittels Error-Interface sowie eine Definition von Pattern für den Nachrichtenaustausch.
WSMO nützt in voller Ausbaustufe Prädikatenstufe erster Ordnung (Full First Order Logic) mit einem minimalen Modell, was (vorhandene) Inferenzmaschinen bereits ausnutzen. Durch die Verwendung von First Order Logic (FOL) wird eine Korrektheit, Vollständigkeit und Erweiterbarkeit garantiert [Agg04].

2.3.4 SWSF

Ein weiterer alternativer Standard im Bereich der Semantic Web Services wird von der Semantic Web Services Initiative (SWSI) vorangetrieben. Er nennt sich das Semantic Web Services Framework (SWSF) und besteht aus zwei Teilen: der Semantic Web Services Language (SWSL) und der darauf aufbauenden Semantic Web Service Ontology (SWSO) [BBB+05]. SWSF ist derzeit dem

W3C (World Wide Web Consortium) zur Prüfung vorgelegt und könnte vom W3C standardisiert werden [GHM05].
Die SWSL besteht aus zwei Teilen, der SWSL-FOL (Sprache auf Basis der First-Order-Language, also Prädikatenlogik erster Ordnung) sowie der SWSL-Rules (einer weiteren Regelsprache). Dieselbe Unterteilung wie bei den Sprachen findet sich auch bei den Ontologien (SWSO). So wird hier in ROWS und FLOWS unterschieden. ROWS ist die *Rules Ontology for Web Services*. FLOWS, die *First-Order Logic Ontology for Web Services*, ist von den Konstrukten sehr stark an OWL-S angelehnt und besteht ebenso wie diese aus drei Teilbereichen: der *Service Descriptoren*, dem *Process Model* und dem *Grounding*.
Service Descriptoren liefern grundsätzliche Informationen über einen Web-Service und können nicht-funktionale Meta-Informationen enthalten. Ausserdem unterstützen sie eine automatische Suche nach Web-Services.
Das *Process Model* definiert (analog zu OWL-S) Eingaben und Ausgaben eines Prozesses sowie die Abfolge verschiedener Prozesse. Auch hier sind die von OWL-S (bzw. WS-BPEL) bereits bekannten Konstrukte zur Beschreibung der Ablaufreihenfolge wie Split, Sequence, Unordered, Choice, IfThenElse, Iterate oder RepeatUntil vorhanden. Allerdings wird das Process Model hier als Untermenge der PSL Ontologie definiert und mit Konzepten zur Service-Beschreibung erweitert. PSL, die Process Specification Language, ist ein internationaler Standard vom National Institute of Standards in den USA (ISO 18629).
Das *Grounding* von FLOWS bietet eine Möglichkeit auf WSDL aufzusetzen und ist gleich aufgebaut wie das *Grounding* eines OWL-S-Services.
FLOWS setzt, wie der Name bereits sagt, auf der Prädikatenlogik erster Stufe auf, die eine sehr gute Ausdruckskraft dank Variablen oder Quantoren bietet und damit mehr als die in OWL-S angebotenen Konzepte umfasst, dafür aber nur semi-entscheidbar ist (im Gegensatz zu OWL-DL). [SWS05]
Das Prozessmodell von PSL umfasst folgende Konzepte (ähnlich zu OWL-S oder WSMO):

- Prozess-Komposition
- Möglichkeit sowohl einfache Workflows wie wiederholte, komplexe Prozesse zu modellieren
- Nebenläufigkeit
- Inputs und Outputs
- Preconditions und Effects
- Invarianten
- Bedingungen und konditionale Prozesse
- Interaktionen mit externen Aktivitäten
- zeitliche Bedingungen (kommen in dieser Art in OWL-S oder WSMO nicht vor)
- explizite Repräsentation des Zustands und von Zustandsbedingungen (kommt in den bisherigen Standards ebenfalls nicht vor)

2.3.5 METEOR-S

Der letzte zu nennende Vorschlag in Richtung Semantic Web Services ist das METEOR-S-Projekt des LSDIS-Labs der Universität von Georgia, USA (siehe [LSD05]) in Zusammenarbeit mit IBM.

METEOR-S steht dabei für *Managing End-To-End OpeRations for Semantic Web Services* und versucht den Prozessgedanken mehr in den Vordergrund zu stellen. Das Hauptaugenmerk des Projektes ist die Anwendung von Semantik im kompletten Lebenszyklus von Semantic Web Prozessen, die komplexe Interaktionen zwischen Semantic Web Services repräsentieren. Die Aufgaben beim Erstellen von Semantic Web Prozessen wurden hierbei identifiziert als die Entwicklung, semantische Annotation, das Auffinden, sowie Zusammenfügen unterschiedlicher Web Services. Aus diesem Grund bietet METEOR-S auch verschiedene, im folgenden kurz beschriebene, Frameworks, um diese Aufgaben zu erfüllen. Ein weiteres Hauptziel der Forschung war die Entwicklung verschiedener Ontologien, die in diesem Bereich benötigt werden. Die gefundenen vier Arten von Semantik werden nachfolgend ebenfalls genauer beschrieben.
METEOR-S bietet ebenso wie die bisherigen Standards die Möglichkeit Inputs, Output, Preconditions und Effects (hier: Postconditions genannt) zu modellieren. Zusätzlich ist es hier noch möglich einzelne Operationen eines Prozesses oder auch mögliche Exceptions klar zu beschreiben. METEOR-S baut auf dem bekannten Standard OWL zur Beschreibung von Ontologien auf und wird momentan für die Regelsprache SWRL erweitert.
METEOR-S definiert vier verschiedene Arten von Semantik:

Data: Damit Web Services miteinander kommunizieren können, benötigen sie ein gemeinsames Vokabular. Die Daten- oder Informations-Semantik ist für die (semi-)formale Definition von Daten in Input- und Output-Nachrichten eines Webservices auf Basis von OWL verantwortlich sowie für eine Beschreibung von aufgetretenen Fehlern (Exceptions) und ermöglicht damit die Interoperabilität von Prozessen.

Functional: Die funktionale Semantik ermöglicht die formale Repräsentation von Möglichkeiten eines Web Services, um ein Auffinden und die Komposition von Web Services zu vereinfachen. Die funktionale Semantik baut dabei auf den Konzepten der Daten-Semantik auf und beschreibt damit die Funktionalität eines Web Services unter anderem durch Angaben, welche Vor- und Nachbedingungen bzw. Effekte in der Aussenwelt auftreten können.

QualityOfService: Die QoS-Semantik ermöglicht eine semiformale Repräsentation von qualitativen (Sicherheit, Transaktionsmodell - WS-Policy) und quantitativen (Kosten, Zeit - WS-Agreement) Maßen von Web Prozessen. Auch die Verfügbarkeit und Verlässlichkeit eines Services kann hier in einer Form beschrieben werden, dass sie auch andere Web Prozesse verstehen, die mit METEOR-S arbeiten.

Execution: Die Ausführungs-Semantik beschreibt die Ausführung oder den Fluss eines Web Services in einem Prozess oder von Operationen in einem Service und ermöglicht damit eine Analyse, Validierung und Ausführung von Prozessmodellen [VSC04]. Zur Modellierung der Ausführungs-Semantik könnten Zustandsmaschinen, Petri-Netze oder auch Aktivitätsdiagramme verwendet werden.

Das METEOR-S Projekt versucht die Forschung von vier verschiedenen wichtigen Punkten im Lifecycle eines Semantic Web Services voranzutreiben: Annotation, Auffindung, Komposition und Ausführung. METEOR-S baut dafür auf verschiedene Architekturen auf, von denen (ähnlich wie bei SWSF) die meisten noch in der Entstehungsphase sind ([CMSP04]):

WSDL-S: Ein Standard zur Beschreibung von Web Services, der diese um semantische Informationen anreichert (setzt direkt auf WSDL auf). Dabei werden Input und Output Message-Parts von WSDL direkt mit Ontologien zusammengebracht. Dies können extra dafür erstellte

Ontologien oder bereits vorhandenes Vokabular (z.B. RosettaNet oder ebXML) sein. Ausserdem werden auch die Operationen und damit verbundene Vor- und Nachbedingungen der Operationen direkt auf Ontologien gemappt.

MWSDI: Die METEOR-S Web Service Discovery Infrastructure ermöglicht die Zusammenarbeit verschiedener Registry-Server (momentan: UDDI), um bei Anfragen an einen Server auch die Ergebnisse zu erhalten, die nur auf einem anderen Server gespeichert sind. Bisher sind in jedem UDDI-Server dieselben Web Services redundant gespeichert, was aber bei einer steigenden Anzahl von Services zu einem Skalierungsproblem führen wird. Daher arbeitet dieser Ansatz auf der Basis eines Peer-to-Peer-Frameworks (JXTA als P2P-Framework), um semantisch annotierte Services nur auf einem Server bereitzustellen, auf welchen Anfragen von anderen Servern weitergeleitet werden können.

MWSAF: Das METEOR-S Web Service Annotation Framework ist ein graphisches Tool um vorhandene Web Services mit semantischen Web Service Beschreibung semi-automatisch anzureichern. Es ermöglicht das gleichzeitige Parsen von Ontologien und Web Service-Beschreibungen. MWSAF war bisher unter dem Namen *Semantic Annotation of Web Services* (SAWS) bekannt.

MWSCF: Das METEOR-S Web Service Composition Framework erweitert die bisher vorhandenen Web Service Kompositionstechniken, indem (semantische) Templates verwendet werden um die semantischen Anforderungen des Prozesses abzudecken. Ein Semantic Process Template beschreibt dabei die benötigte Funktionalität eines bestimmten Prozesses, die durch ausführbare Services später instanziiert werden.

MWSDPM: Der METEOR-S Web Service Dynamic Process Manager erlaubt das Binden von Web Services sowohl zur Erstellungszeit als auch zur Laufzeit basierend auf Geschäfts- und Prozessbedingungen.

Durch die direkte Anbindung der Ontologien in die Beschreibung des Web Services mittels WSDL-S benötigt dieser Ansatz im Gegensatz zu den vorher Genannten kein explizites Grounding.

2.3.6 Vergleich und Auswahl

OWL-S bietet sicherlich den am bekanntesten und am weitesten fortentwickelten Standard. Es existiert bereits eine Vielzahl von Tools oder APIs, die auf OWL-S aufbauen. Dank der Standardisierung durch das W3C ist OWL-S sicherlich am zukunftsträchtigsten. WSMO bietet als einziger Standard das Konzept der Mediatoren, um die explizit modellierten Ziele mit den Web Services zu vereinen oder den Zusammenhang zwischen verschiedenen Ontologien darzustellen. Mit WSMX existiert auch hier bereits eine Referenzimplementierung, aber mittlerweile setzen auch Andere, wie z.B. das Internet Reasoning System IRS-III, auf WSMO auf. Dank First Order Logic sind WSMO oder SWSF besser erweiterbar und sie bieten eine bessere Ausdrucksstärke, schränken aber die automatische Beweisbarkeit ein.
SWSF und METEOR-S sind beide noch in der Entstehung und bisher erst teilweise veröffentlicht oder auch nur angedeutet. Desweiteren existieren im Moment noch keine Referenzimplementierungen, weshalb nicht auf diesen Standards aufgesetzt wird. Wohl aber definieren sie einige Konzepte, die für den Bereich der Prozessmodellierung oder der Prozesssynthese durchaus interessant sind.
Aufgrund der Bekanntheit wird OWL-S als Hauptstandard verwendet, um die einzelnen Prozesse zu beschreiben. Wie in WSMO üblich, wird eine Trennung von Ontologie und der Web Service Beschreibung vorgenommen. Die Web Service Beschreibung wird aber mittels OWL erfolgen. Um

globale Zustände beschreiben zu können, wird das Variablen-Konzept von SWSF aufgegriffen. Ebenso sollen die zeitlichen Zusammenhänge zwischen zwei Prozesen dargestellt werden wie im Vorschlag von SWSF spezifiziert. Desweiteren werden drei der vier von METEOR-S definierten Semantik-Arten (Data, Functional und Execution) aufgegriffen. Die QoS-Semantik ist für den Bereich der Modellsynthese zur Zeit (im Rahmen dieser Arbeit nicht relevant, könnte aber jederzeit hinzugefügt werden. Die Ausführungssemantik (Execution Semantic) umfasst dabei die zeitlichen Constraints und wird in der Datensemantik integriert. Diese beinhaltet desweiteren die Ontologie und Variablen, die benötigt werden, um die Prozesse (welche in der Functional Semantics spezifiziert sind) zu beschreiben. Abbildung 2.9 zeigt eine grobe Übersicht der vorhandenen Semantic Web Service Standards und deren Zusammenarbeit mit anderen Standards.

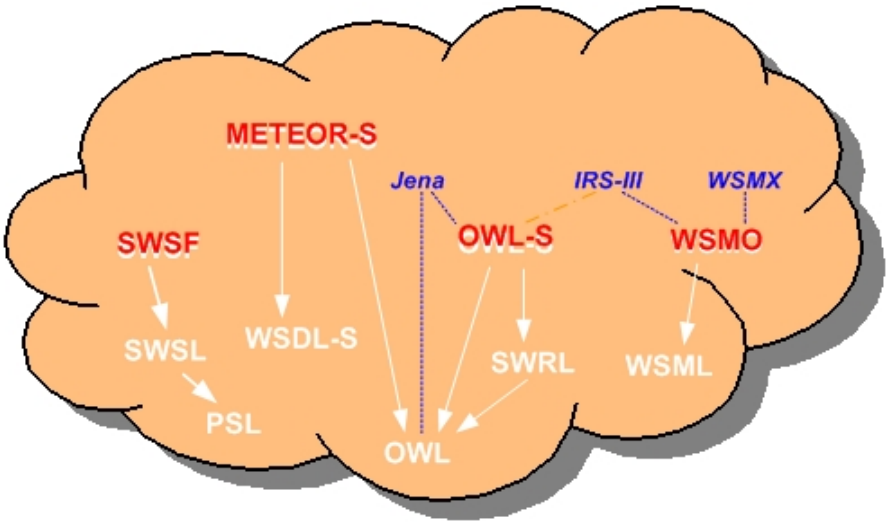


Abbildung 2.9: Die Semantic Web Service Standards im Überblick

Kapitel 3

Ontologie-basierte Erweiterungen der Geschäftsprozessmodellierung

3.1 Möglichkeiten der semantischen Beschreibung

3.1.1 Übersicht

Um Prozesse im Rahmen der Geschäftsprozessmodellierung Ontologie-basiert zu erweitern, werden die im vorigen Kapitel dargestellten Konzepte aufgegriffen und verwendet.
Interessant für eine semantische Beschreibung von Prozessen ist sicherlich, was ein Prozess genau macht, welche Auswirkungen er auf einen globalen Zustand hat oder wann er überhaupt aufgerufen werden kann. Ausserdem muss klar definiert sein, welche Begrifflichkeiten zur Beschreibung eingesetzt werden können, nach welchem Standard diese definiert sein müssen und wo diese definiert werden. Diese Erweiterungen sollen sowohl zur Beschreibung einer einzelnen Aktion als auch für die semantische Beschreibung einer kompletten Aktivität möglich sein, um diese auch im Zusammenhang mit anderen Aktivitäten nutzen zu können.
Eine Aufteilung der Ontologien wie im Standardisierungsvorschlag METEOR-S vorgesehen, erscheint zur Reduzierung der Komplexität sehr sinnvoll zu sein. Daher wird im Weiteren zwischen der Datensemantik, funktionalen Semantik und Ausführungssemantik unterschieden. Die im Rahmen des METEOR-S-Projektes ebenfalls verwendete QoS-Semantik bringt für die Synthese von Geschäftsprozessen keinen Nutzen, könnte aber jederzeit nachträglich eingebaut und verwendet werden.
Die vier Arten der Semantik können unterschieden werden in globale und lokale Beschreibungen. Die globale Beschreibung umfasst die Data Semantics und die Execution Semantics und beschreibt die für alle Prozesse geltende Ontologie sowie wie verschiedene Prozesse logisch zusammenhängen können. Im Gegensatz dazu beschreibt die lokale Semantik (wie in der Functional Semantics und der QoS Semantics), wie ein einzelner Prozess aufgebaut ist, welche Parameter er hat und welche Bedingungen gelten müssen, damit er überhaupt ausgeführt werden kann, sowie welche Bedingungen er während der Laufzeit oder auch danach erfüllt. Diese Unterscheidung zwischen globaler und lokaler Semantik wird im Nachfolgenden auch eine entscheidende Rolle spielen.
Im Folgenden werden die einzelnen Semantiken genauer erklärt.

3.1.2 Daten-Semantik

Im Rahmen der Datensemantik werden alle Konstrukte definiert, die für die genauere Beschreibung eines Prozesses benötigt werden. Hier werden sowohl alle Parameter, die später für die Beschreibung von Input und Output verwendet werden, sowie ihre Zusammenhänge spezifiziert, als auch die Variablen, welche für die Ausdrücke benötigt werden, um Voraussetzungen und Nachbedingungen eines Prozesses zu definieren. Variablen können einen Wert zugewiesen bekommen und es kann überprüft werden, ob eine Variable noch undefiniert ist. Diese Variablen dienen der Beschreibung eines globalen Kontextes, in dem ein Prozess steht. Jeder Prozess kann Auswirkungen auf andere haben, dadurch dass er entweder Objekte erstellt oder zerstört, Daten in eine Datenbank schreibt, etc. Ein Konzept der Datensemantik könnte z.B. ein „Professor“ sein oder ein „Student“, sowie der Zusammenhang „Professor“ „lehrt“ „Student“ (siehe Abbildung 3.1).

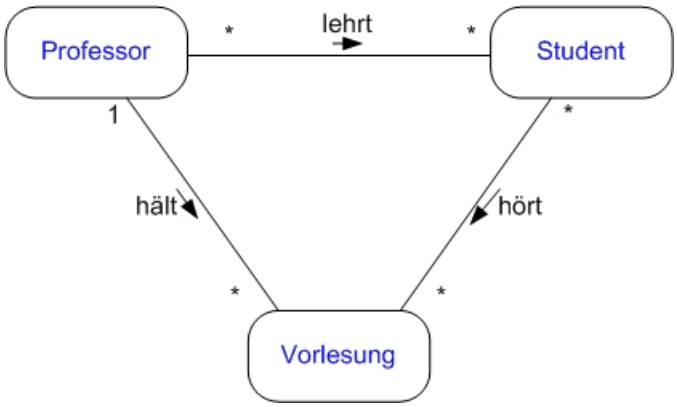


Abbildung 3.1: Konzepte im Rahmen der Datensemantik

3.1.3 Funktionale Semantik

Aufbauend auf der Datensemantik können für die funktionale Semantik Konzepte herausgegriffen werden, um einen Prozess genauer zu beschreiben. Die funktionale Semantik teilt sich dabei wie gewohnt in Input, Output, Preconditions und Effects (IOPEs) auf. In Input und Output steht jeweils eine Liste von Konzepten der Datensemantik. Ein Prozess kann einen oder mehrere Inputs benötigen oder auch gar keinen (ebenso wie Outputs) und dabei als Input auch die Outputs von mehreren anderen Prozessen als Voraussetzung haben. Beispielsweise würde der Prozess „Vorlesung halten“ einen „Professor“ als Input sowie mindestens einen „Student“ benötigen.
Preconditions und Effects können Variablen beschreiben (z.B. Variable X hat den Wert „true“) bzw. auch abfragen („Ist der Wert von Y grösser als 10?“). Eine Precondition des Prozesses „Vorlesung halten“ wäre zum Beispiel „Beamer funktionstüchtig“ oder „Verbindungskabel vorhanden“ (siehe Abbildung 3.2).

3.1.4 Ausführungssemantik

War bisher Fachwissen für die Beschreibung der Konzepte ausreichend, benötigt man zum Modellieren der Ausführungssemantik Domänenwissen. Die Ausführungssemantik spezifiziert die Bedingungen die ein Ablauf von Prozessen einhalten muss und benötigt daher eine Auflistung aller

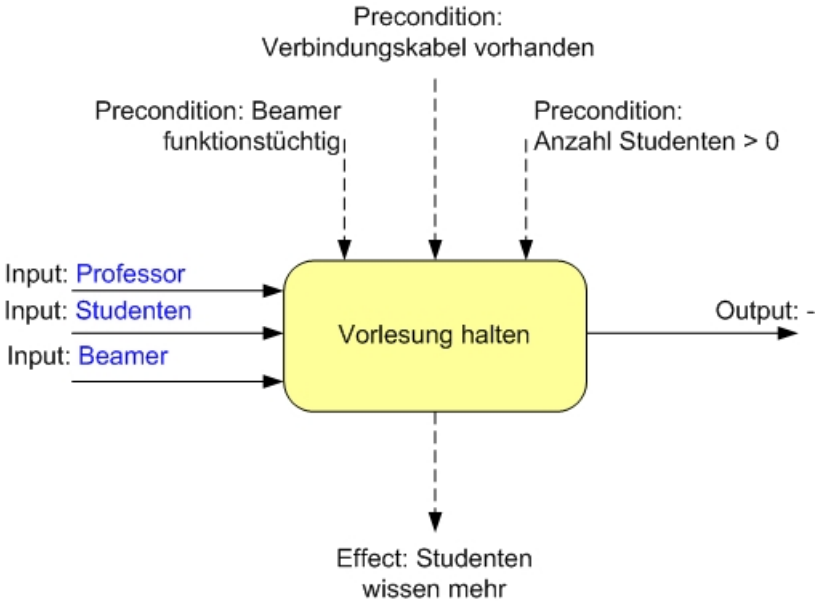


Abbildung 3.2: Funktionale Semantik eines Prozesses „Vorlesung halten“

Prozesse. So kann hier definiert werden, dass Prozess A direkt vor Prozess B kommt, vor Prozess C irgendwann Prozess A gekommen sein muss oder die beiden Prozesse D und E parallel ablaufen müssen. Diese Bedingungen werden dann bei der Berechnung einer optimalen Ablaufreihenfolge berücksichtigt (siehe Abbildung 3.3).

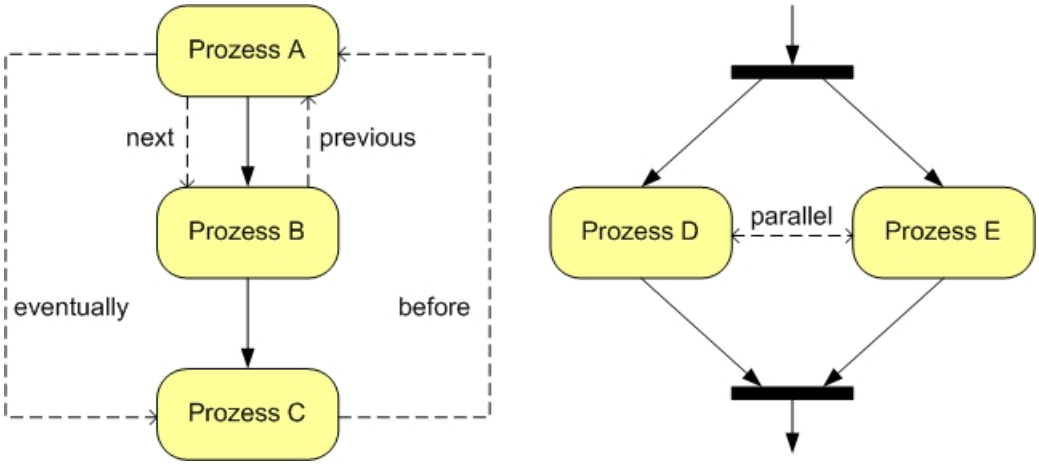


Abbildung 3.3: Die Konstruke der Ausführungssemantik

Abbildung 3.4 stellt den konzeptionellen Aufbau dar: In einem UML2-Aktivitätsdiagramm werden alle Prozesse (hier beispielhaft Prozess A und Prozess B) mit Input, Output, Preconditions und Effects spezifiziert, wobei auf die Ontologie und die Variablen der Datensemantik aufgebaut wird. Die Ausführungssemantik spezifiziert parallel dazu Bedingungen über den späteren Ablauf der Prozesse.

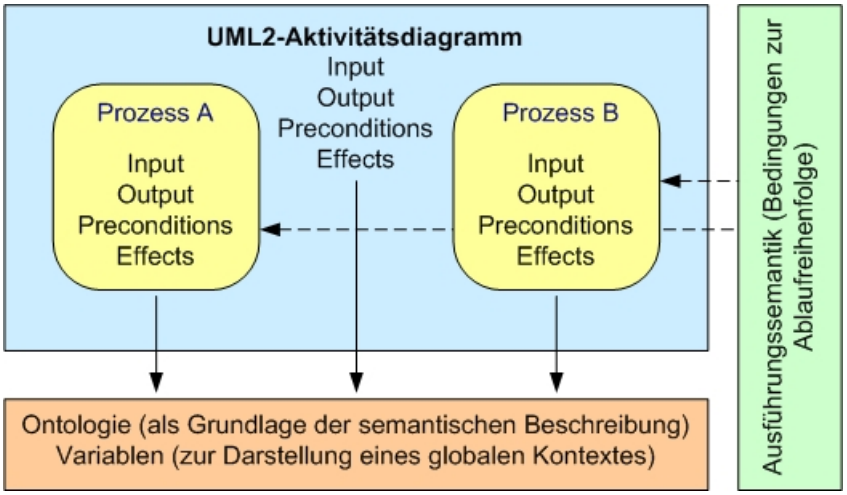


Abbildung 3.4: Die semantische Beschreibung von Prozessen

3.1.5 Erweiterungsmöglichkeiten und Vorgehensmodell

In dem bisherigen Ansatz wird nicht berücksichtigt, dass Inputs beispielsweise auch optional sein können, also ein Prozess zur Bearbeitung eines Auftrags gewisse Vorgaben benötigen würde, beim Fehlen dieser aber auf Standardwerte zurückgreifen kann. Auch ist es nicht möglich, Objekte entweder nur als Input oder nur als Output zu beschreiben oder auf eine dynamische Änderung der Prozesse zu reagieren. Sollte sich also ein Prozess verändern (weil er entweder einen Befehl dazu erhalten hat oder sich selbst umorganisiert), müsste er komplett umgeschrieben werden.
Momentan wird nicht zwischen Zielen und unerwünschten Nebenwirkungen unterschieden, sondern beide Arten werden im Bereich der Effects modelliert. Ausserdem wäre noch denkbar eine Modellierung von Invarianten eines Prozesses hinzuzufügen, also Vorgaben, die während der Ausführung eines Prozesses immer eingehalten werden müssen, damit der Prozess zum Ende gelangen kann (z.B. während einer Überweisung darf das zugrundeliegende Konto nicht aufgelöst werden).
Eine typische Vorgehensweise beim semantischen Modellieren wäre die Definition von Variablen auf Basis vordefinierter Datentypen für die Modellierung von Preconditions und Effects. Parallel dazu kann die Definition der Ontologie vorgenommen werden. Nun kann die Definition der Prozesse begonnen werden und die Zuweisung von Input und Output sowie Preconditions und Effects. Sind alle Prozesse definiert, kann zuletzt die Beschreibung der Ausführungssemantik vorgenommen werden. Abbildung 3.5 fasst obiges graphisch nochmals zusammen.

3.2 Konkrete Umsetzung

3.2.1 Erstellung von Ontologien

Um die Basis für die semantische Geschäftsprozessmodellierung zu schaffen, muss wie oben beschrieben die Datensemantik erstellt werden. In dieser werden einzelne Konzepte sowie deren Zusammenhang beschrieben. Damit diese für die weitere Synthese automatisch erkannt werden können, wurde hierfür eine Basisontologie geschaffen, die hier näher erklärt werden soll. In dieser zugrundeliegenden Datei namens SemBPM.owl (Semantic-based Business Process Modeling) wer-

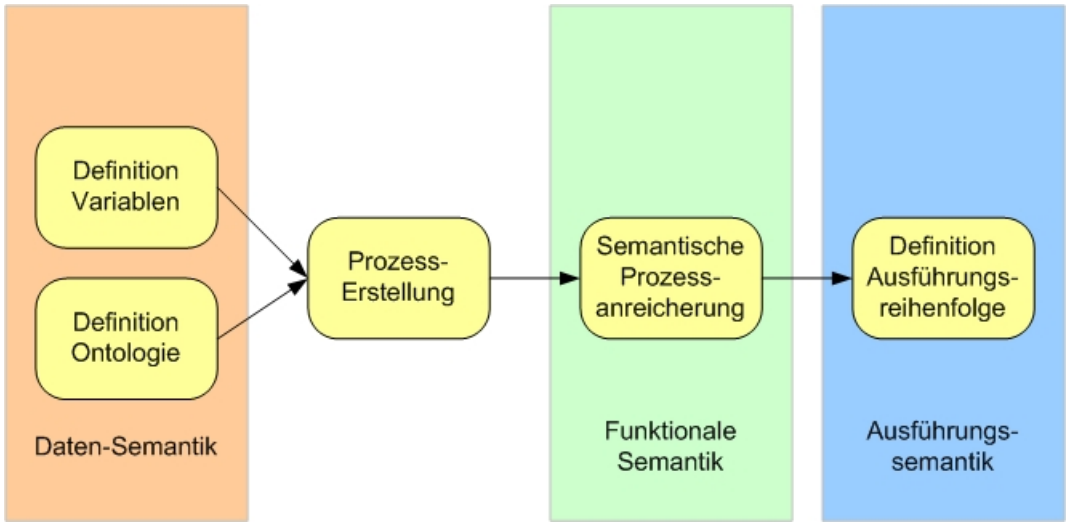


Abbildung 3.5: Vorgehensmodell der semantischen Modellierung

den folgende Konzepte (aus der Datensemantik ebenso wie aus der Ausführungssemantik - also der globalen Semantik) definiert:

Object: Für die Definition von Konzepten der Ontologie werden Object-Konstrukte verwendet. Jedes Konzept muss als Object definiert werden. So beispielsweise „Konto“, „Bank“, „Zinsen“, etc. Um die Zusammenhänge zwischen zwei Objekten darzulegen werden „SemObjectProperties“ verwendet wie z.B. „hat_Zins“ in „Konto“ „hat_Zins“ „Zins“ ersichtlich.

SemDataProperty/Variables: Für die Arbeit mit Variablen existiert das Konstrukt „globalVariable“ als Unterklasse von „Variable“. Variablen können mittels „SemDataProperty“ belegt und ausgelesen werden. Beispiele für diese SemDataProperties wären „setValue“, um die Variable mit einem beliebigen Wert (boolean, integer, String) zu belegen oder auch „hasValue“, um zu überprüfen ob eine Variable bereits belegt wurde oder ob sie genau einen Wert hat. Ausserdem sind noch die Vergleichsoperatoren „lessThan“ und „biggerThan“ vorgegeben, um zwei numerische Werte vergleichen zu können.

SemExecutionProperty/Process: Die Ausführungssemantik benötigt Prozesse, über die sie reden kann. Diese Prozesse können in unterschiedlichem Zusammenhang zueinander stehen (ausgedrückt als SemExecutionProperties): „next“ gibt an, dass auf einen Prozess direkt ein anderer folgt; „previous“ spezifiziert einen direkten Vorgängerprozess; „eventually“ und „before“ betrachten die nicht-direkte Abfolge zweier Prozesse (Prozess A kommt irgendwann vor Prozess B) und „parallel“ setzt zwei Prozesse in die parallele Abfolge.

Abbildung 3.6 zeigt das Meta-Modell und die Zusammenhänge der soeben eingeführten Konstrukte. ObjectProperty, DatatypeProperty und AtomicProcess sind dabei bestehende Konstrukte aus den Standards OWL bzw. OWL-S und dienen als Grundlage für die neu erstellten Objekte zur semantischen Beschreibung von Geschäftsprozessen.

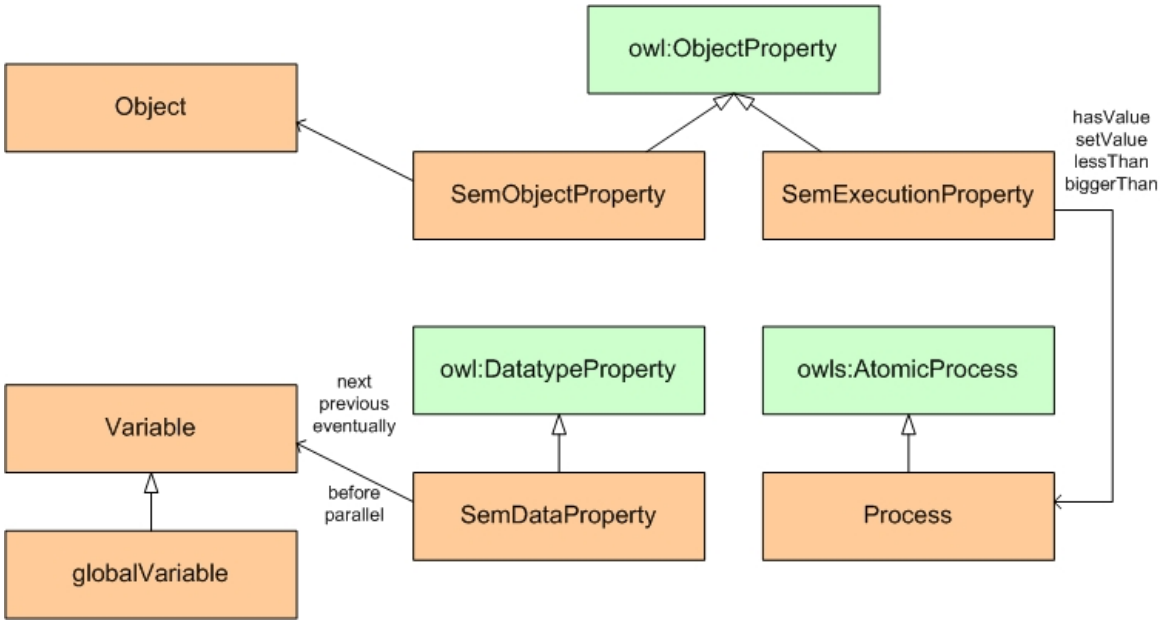


Abbildung 3.6: Metamodell von SemBPM

3.2.2 Einführung eines Fallbeispiels

Um die konkrete Umsetzung der Semantik-Arten verständlicher darlegen zu können, wird im Folgenden als Fallbeispiel das Buchen einer Dienstreise eingeführt. Dieser Geschäftsprozess umfasst erfahrungsgemäß das Buchen eines Hotels, eines Hin- und Rückfluges sowie (optional) eines Mietwagens während des Aufenthalts am Veranstaltungsort (meist übernehmen diese Aufgaben mittlerweile die Reisebüros oder diverse Anbieter im Internet).
Vereinfachend könnte man folgende Prozesse zusammenfassen:

Geschäftsreise planen: Beim Planen einer Geschäftsreise muss berücksichtigt werden, an welchem Ort die zu besuchende Konferenz stattfindet und an welchen Tagen man anwesend sein möchte.

Flug planen: Hier muss entschieden werden von welchem Flughafen man starten möchte und welches der zum Hotel nächstgelegene Zielflughafen ist. Eventuell könnten hier auch Präferenzen für gewisse Fluganbieter ausgedrückt werden. Um die Flughäfen planen zu können, benötigt man natürlich das Land, in welchem die Konferenz stattfindet sowie das Anfangs- und Enddatum der Reise.

Flug buchen: Bei bekanntem Abflughafen und Zielflughafen sowie Anfangs- und Enddatum ist es möglich (evtl. bei einer bestimmten Fluglinie) einen Flug zu buchen. Nach Ausführung dieses Prozesses soll der globale Zustand wissen, dass der Flug gebucht ist.

Hotel planen: Ausgehend von dem Ort der Konferenz kann in passender Nähe ein Hotel mit einem gewissen Standard (z.B. mindestens 4 Sterne) gebucht werden.

Hotel buchen: Wenn ein passendes Hotel gefunden wurde, kann dieses in dem passenden Zeitraum gebucht werden. Nach Ausführung dieses Prozesses soll der globale Zustand so geändert

werden, dass die Hotelbuchung bekannt ist. Wenn beides (Flug und Hotel) erfolgt ist, kann man sich zuletzt noch um den Mietwagen kümmern:

Mietwagen buchen: Für die Tage im Ausland kann bei einem beliebigen Autovermieter (wieder je nach Präferenzen) für alle oder nur bestimmte Tage ein Leihwagen gebucht werden. Dies soll im globalen Zustand ebenfalls ausgedrückt werden.

Geschäftsreise durchführen: Hat man alle Buchungen vorgenommen, kann man den Vorgang der Reisebuchung abschliessen und sich auf den zu haltenden Vortrag am Zielort vorbereiten und schliesslich die Geschäftsreise durchführen.

3.2.3 Die Datensemantik anhand eines Beispiels

Auf Basis der SemBPM.owl können für die oben beschriebenen Prozesse die Bestandteile der Datensemantik spezifiziert werden. So wird es Objekte geben zum „Land_der_Konferenz“, zum „Anfangsdatum“ und „Enddatum“ (abgeleitet von einer gemeinsamen Oberklasse „Datum“), „Abflughafen“ und „Zielflughafen“ (ebenfalls Unterklassen, hier von „Flughafen“) sowie „Autovermietung“ und „Auto“ oder „Hotelanbieter“ und „Hotel“. Ausserdem kann hier ausgedrückt werden, dass eine Autovermietung ein Auto vermietet, das Hotel in dem Urlaubsland liegt oder der Hotelanbieter ein Hotel anbietet. Ein beispielhafter Auszug aus der Ontologie sähe folgendermassen aus (komplette Datei siehe Anhang D):

```
<semBPM:Object rdf:ID='Autovermietung'/>

<semBPM:Object rdf:ID='Auto'/>

<semBPM:SemDataProperty rdf:ID='vermietet'>
   <rdfs:domain rdf:resource='#Autovermietung'/>
   <rdfs:range rdf:resource='#Auto'/>
</semBPM:SemDataProperty>
```

Denkbar wären hier noch einige Erweiterungen: Anzahl der Sterne eines Hotels, Grösse und PS eines Autos, Zusätze wie Klimaanlage, etc. Für das recht kurze einführende Beispiel soll der momentane Umfang der Beschreibungen ausreichen. Eine Definition von Variablen in der Ontologie könnte beispielsweise durch

```
<semBPM:globalVariable rdf:ID='Flug_gebucht'/>
```

erfolgen. Dies würde eine neue Variable anlegen (Name: Flug_gebucht), welche später für die funktionale Semantik verwendet werden kann.

3.2.4 Funktionale Semantik für das Beispiel

Alle vorhandenen Prozesse müssen mit semantischen Informationen belegt werden: der Prozess „Flug buchen“ beispielsweise bekäme als Input das Land der Konferenz sowie die vorher festgelegten Anfangs- und Enddaten der Reise und würde nach seiner Ausführung den Anfangsflughafen und Zielflughafen zurückgeben. Der Prozess „Flug buchen“ würde genau diese Informationen als Input benötigen und als Effect im globalen Zustand die Reise als gebucht erfassen.
Ein Ausschnitt der funktionalen Semantik zur Beschreibung des Prozesses „Flug buchen“ sähe folgendermassen aus:

```
<semBPM:Process rdf:ID='Flug_buchen'>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Abflughafen'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Zielflughafen'/>
   </owls:hasInput>
   <owls:hasResult>
      <owls:Result>
         <owls:hasEffect>
            <owls:Effect>
               <semBPM:globalVariable semBPM:variable='&dataSem;#Flug_gebucht'
                  semBPM:hasValue='True'/>
            </owls:Effect>
         </owls:hasEffect>
      </owls:Result>
   </owls:hasResult>
</owls:AtomicProcess>
```

Die funktionale Semantik stellt die „lokale Semantik“ dar, d.h. sie wird für jeden Prozess eigens definiert und findet sich nicht in derselben Datei wie die Datensemantik. Abbildung 3.7 zeigt eine detaillierte Zusammenfassung aller Prozesse im Rahmen der Geschäftsreise-Buchung sowie ihrer funktionalen Semantik.

3.2.5 Die Ausführungssemantik im Beispiel

Im oben dargestellten Beispiel könnte festgelegt werden, dass die Prozesse „Flug buchen“ und „Hotel buchen“ parallel ablaufen müssen, da beispielsweise die Buchung eines Hotels ohne den Flug ins jeweilige Land sicher keinen Sinn machen würde. Auch wäre vorstellbar zu definieren, dass das Buchen eines Mietwagens erst nach der Flugbuchung geschehen darf. Man könnte aber natürlich ebenso festlegen, dass das Buchen des Mietwagens parallel dazu passieren soll. Hier sind dem Modellierer des Geschäftsprozessmodells keinerlei Grenzen gesetzt. Konkret könnte hier also ausgewählt werden, dass der Prozess `Flug_buchen parallel` zum Prozess `Hotel_buchen` stattfinden soll.

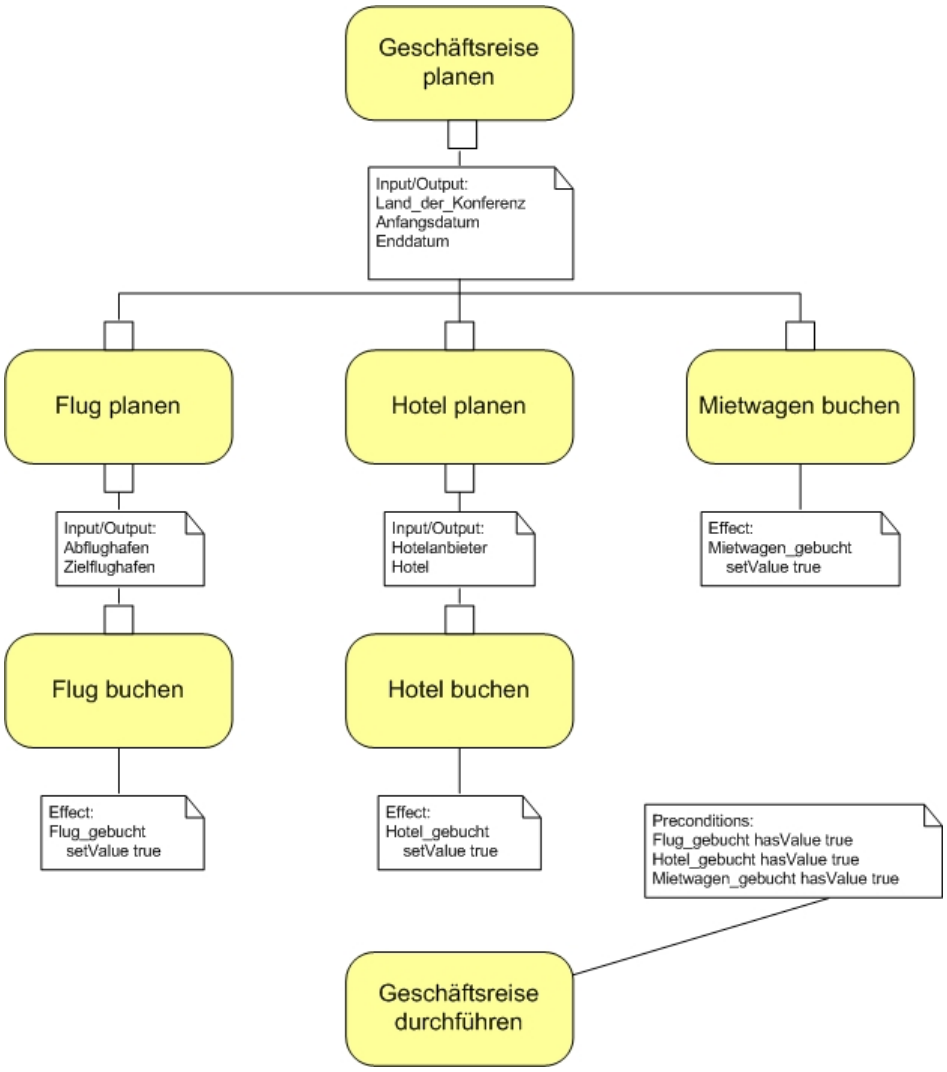


Abbildung 3.7: Funktionale Semantik am Beispiel des Buchens einer Geschäftsreise

Kapitel 4

Synthese

4.1 Übersicht

Für eine automatische Synthese von Prozessen innerhalb eines Geschäftsprozessmodells wird auf die im vorherigen Kapitel erläuterte semantische Beschreibung zurückgegriffen. Diese wird von mehreren Prozessen verglichen, um so ein „Matching“ (also ein Zusammensetzen bzw. Hintereinanderausführung) der Prozesse herauszufinden. Dafür wird die Daten-Semantik und die funktionale Semantik in eine sogenannte Inferenz-Maschine (inference machine oder auch reasoner) geladen und darin Anfragen gestellt, inwiefern die jeweiligen Parameter (IOPEs) zueinander passen. Aufbauend auf den Ergebnissen wie zwei Prozesse matchen, werden Matrizen erstellt, die den Grad der Kongruenz oder der Parallelisierbarkeit zweier Prozesse speichern. Im Folgenden wird ein Algorithmus auf diese Matrizen angewandt, um die möglichst beste Synthese von Prozessen zu berechnen.

Im Rahmen der Synthese wird zuerst die nachfolgend beschriebene Synthese-Matrix erstellt. Um die Werte für diese Matrix zu berechnen, werden innerhalb der Inferenzmaschine die Elemente der funktionalen Semantik und ihrer Zusammenhänge in der Datensemantik überprüft. Hier wird beispielsweise getestet, ob der Input eines Prozesses gleich dem Output eines anderen Prozesses ist oder eine Teilmenge. Bei der Erstellung der Matrixeinträge werden ausserdem die Bedingungen der Ausführungssemantik berücksichtigt (z.B. ein Prozess soll direkt auf einen anderen folgen).
Nach Erstellung der Synthese-Matrix wird eine weitere Matrix mit dem Namen IdentityMatrix erstellt, die festhält, wie sehr sich zwei Prozesse ähneln und ob sie sich daher gut parallelisieren lassen oder nicht. Dabei wird auch beim Erstellen dieser Matrix die Inferenzmaschine benützt, um die IOPEs der funktionalen Semantik zu erhalten und auf Basis der Datensemantik zu vergleichen. Wurden beide Matrizen erstellt, startet das Syntheseverfahren. In diesem wird unter Zuhilfenahme der Matrizen ein Algorithmus (standardmässig werden die beiden Algorithmen „modified Prim“ und „RandomWalk“ zur Auswahl geboten) ausgeführt, der basierend auf einem gewichteten, gerichteten Graphen, der durch die Synthese-Matrix gebildet werden kann, versucht den besten Ablauf zwischen den Prozessen zu finden. Wurde eine oder mehrere Lösungen gefunden, wird die Beste an den Benutzer zurückgegeben (eine Übersicht bietet Abbildung 4.1).

Das nachfolgende Kapitel beschreibt eine State-of-the-art-Analyse derzeit verfügbarer Inferenz-Maschinen und wählt eine für die Implementierung aus. Das darauf folgende Kapitel beschreibt die verwendeten Matrizen genauer, um anschliessend das Syntheseverfahren und die damit verbundenen Algorithmen an sich zu erklären.

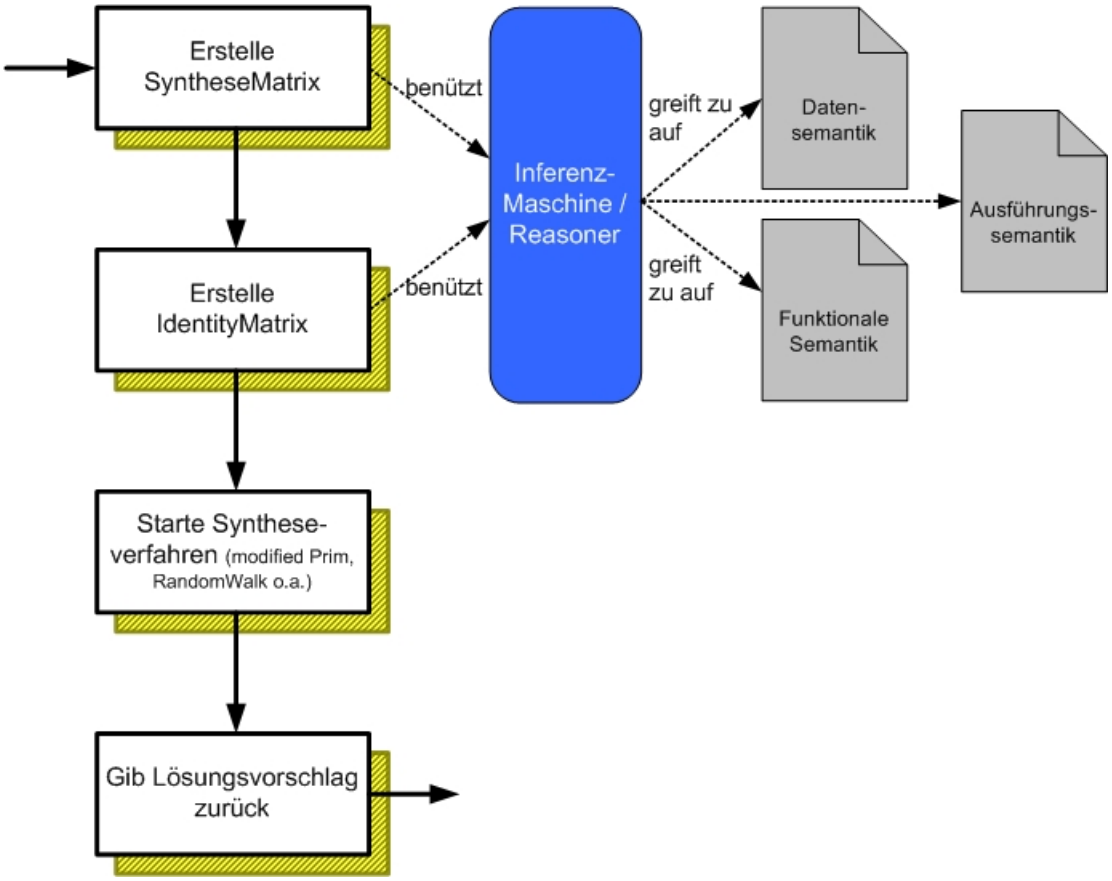


Abbildung 4.1: Schematischer Ablauf des Syntheseprozesses

4.2 Arten von Inferenz-Maschinen

4.2.1 Was ist eine Inferenzmaschine?

Eine Inferenzmaschine (teilweise auch Reasoner oder Theorem-Beweiser genannt) ist ein Software-Tool, um neue Fakten oder Assoziationen aus vorhandenen Informationen zu erzeugen. Man unterteilt diese in drei Schritte: Akquisition (Wissenserhebung und -erfassung), Repräsentation (Darstellung und Speicherung des Wissens in der Wissensbasis) und Inferenz (Verarbeitung des Wissens zur Lösung bestimmter Probleme) [SIC05].
Oft wird behauptet, dass eine Inferenz-Maschine die menschlichen Fähigkeiten des Schlussfolgerns nachahmt. Indem man aus den Informationen und Beziehungen ein Modell erstellt, ermöglicht man den Reasonern logische Schlussfolgerungen auf Basis dieses Modells zu ziehen. Ein typisches Beispiel eines Reasonings ist es Modelle von Menschen und ihren Beziehungen zu anderen Personen zu erstellen, um neues Wissen zu erhalten. Durch eine genaue Betrachtung eines solchen Netzwerkes können Inferenzmaschinen Beziehungen erkennen, die vorher nicht explizit definiert wurden [SIC05]. So könnte beispielsweise aus dem Wissen, dass A Vater von B und C Bruder von A ist, geschlussfolgert werden, dass C Onkel von B ist. Vorausgesetzt natürlich, dass die Regeln wie eine Onkel-Beziehung dargestellt wird, dem System bekannt sind.

Die Anfänge von Inferenzmaschinen gehen bis in das Jahr 1943 zurück: die damals definier-

ten *Post Production Rules* definierten eine symbolische Logik, wonach eine Menge von Symbolen mittels Antecedent und Consequent (also Vor- und Nachbedingung) in eine anderen Menge von Symbolen übergeleitet werden können. Diese Regeln wurden im Markov-Algorithmus (1954) mit Kontrollstrukturen versehen, um eine leichtere Handhabung zu ermöglichen. In dem 1957 von Newel und Simon gestarteten Projekt „General Problem Solver“ (GPS) zeigten diese, dass die meisten menschlichen Probleme durch einfache IF-THEN-Produktionsregeln gelöst werden können. Dabei basiert eine Regel auf einer geringen Menge von Wissen (sogenanntem „Chunk“). Zusätzlich wird ein einfacher Prozesser („Cognitive processor“, also eine Inferenzmaschine) benötigt, die heutzutage die Basis von modernen regelbasierten Expertensystemen darstellt. Die hohe Ineffizienz des Markov-Algorithmus (jede Regel muss immer wieder getestet werden, dadurch existieren bereits bei einer geringen Zahl von Regeln sehr hohe Laufzeiten) wurde 1979 von Charles L. Forgy durch den Rete-Algorithmus für schnelles Pattern Matching behoben, der auch heute noch in vielen Inferenz-Maschinen benützt wird.

Forward Chaining vs. Backward Chaining

Bei Inferenzmaschinen werden naturgemäss zwei Arten des Schlussfolgerns unterschieden: Auf der Basis einer Regel, die in der Form

```
wenn A, dann B
```

repräsentiert wird, kann eine einfache Schlussfolgerung gezogen werden. Häufig will man jedoch komplexere Schlussfolgerungen aus mehreren gegebenen Regeln ziehen. Eine Möglichkeit hierzu besteht in der Verkettung von Regeln. Die Vorwärtsverkettung (engl. forward chaining) geht dabei transitiv vor, d.h. aus einem Faktum wird anhand einer Regel und einer Inferenzmethode (z.B. modus ponens) eine Schlussfolgerung gezogen, die wiederum als Prämisse und mittels einer weiteren Regel für eine weitere Schlussfolgerung verwendet wird usw. Da von einem meist fallspezifischen Faktum ausgegangen wird, bezeichnet man diese Inferenzstrategie auch als datengetriebene Inferenz. Ebenso wie die Vorwärtsverkettung basiert die Rückwärtsverkettung (engl. backward chaining) auf einer transitiven Verknüpfung von Regeln. Man geht dabei jedoch vom Zielobjekt aus und prüft nur die Regeln, die das Ziel in der Schlussfolgerung haben. Falls der Wert eines Objektes in der Prämisse einer solchen Regel unbekannt ist, wird versucht, diesen aus anderen Regeln herzuleiten. Gelingt dies nicht, so wird der Wert schließlich vom Benutzer erfragt. Man nennt dieses Verfahren auch zielorientierte Inferenz.
Desweiteren unterscheidet man zwei Arten von Wissen: ABox und TBox. Unter TBox wird das intensionale, also unveränderliche Wissen in Form einer Terminologie bezeichnet (terminologisches Wissen), welches sich auch beim Inferenzschritt zur Erhebung neuen Wissens nicht verändert. ABox hingegen beinhaltet das extensionale Wissen spezifisch für die Instanzen einer Domain (auch Individuals genannt - assertions on individuals, daher: assertorisches Wissen). Dieses Wissen kann sich im Laufe der Zeit verändern (meist vergrössern).
In OWL findet auch diese Unterscheidung in ABox und TBox-Wissen statt: OWL Facts wie Typ-Zusicherungen, Eigenschaftszusicherungen, Gleichheit oder Ungleichheit von Individuen sind ABox-Zuweisungen, OWL Axiome hingegen (wie Unterklassen, Untereigenschaften, die Domain oder der Bereich einer Property) hingegen sind TBox-Wissen. [SP04]

4.2.2 Einteilung von Inferenzmaschinen und Übersicht

Einteilung nach Art der Logik

Die **Aussagenlogik** (auch veraltet Urteilslogik genannt) ist ein Bereich der Logik, der sich mit der logischen Bewertung von Aussagen befasst. Dabei werden Elementaraussagen durch Junktoren ($\vee$, $\wedge$) verknüpft. Die Struktur der Elementaraussagen wird nicht näher untersucht. Aussagenlogik besteht aus Ausdrücken, die durch Terme und Formeln gebildet werden. Terme sind induktiv definiert als:
Sind S und T Terme, so sind auch S $\vee$ T, S $\wedge$ T, $\neg$ S Terme. Formeln werden definiert als S $\rightarrow$ T, S $\leftrightarrow$ T, wenn S und T Terme sind. Darauf aufbauend können Wahrheitswerte und Wahrheitstabellen auf Grundlage von boolschen Werten erstellt werden (1=true, 0=false).
In der Aussagenlogik werden Symbole (Terme und Formeln) verwendet, um Fakten über die Welt darzustellen. Die Tatsache „Professor gibt Vorlesung“ könnte z.B. durch das Symbol P dargestellt werden, oder auch durch aussagekräftigere Symbole wie beispielsweise *ProfessorGibtVorlesung*. Einfache Fakten wie diese werden als *atomare Sätze* bezeichnet. Es können komplexere Aussagen (oder Sätze) gebildet werden, indem atomare Sätze mit den logischen Bindegliedern $\vee$ (oder), $\wedge$ (und), $\neg$ (nicht), $\rightarrow$ (Folgerung) und $\leftrightarrow$ (Äquivalenz) kombiniert werden. Beim Vorliegen eines weiteren Satzes Q (z.B. gleichbedeutend mit „Professor bereitet Vorlesung vor“), könnten wir folgende Fakten haben:

- $P \vee Q$: „Der Professor gibt eine Vorlesung oder bereitet sie vor“.
- $P \wedge Q$: „Der Professor gibt eine Vorlesung und bereitet eine Vorlesung vor“.
- $\neg Q$: „Der Professor bereitet keine Vorlesung vor“.
- $Q \rightarrow P$: „Wenn der Professor eine Vorlesung vorbereitet, dann gibt er sie auch“.
- $Q \leftrightarrow P$: „Wenn der Professor eine Vorlesung vorbereitet, hält er sie auch und umgekehrt“.

Die Aussagenlogik stellt eine formale Wissensrepräsentationssprache zur Verfügung. Aussagenlogische Kalküle sind korrekt und vollständig und Aussagenlogik ist entscheidbar. Da aber die Aussagenlogik keine große Ausdrucksmächtigkeit hat und eine Wissensrepräsentation nur durch Angabe von Fakten (aussagenlogische Variablen) und Beziehungen zwischen diesen (durch logische Konnektoren) möglich ist, hilft die darauf aufbauende Prädikatenlogik weiter. In der Aussagenlogik ist es beispielsweise nicht möglich Aussagen zu treffen wie „Ein Professor bereitet alle Vorlesungen vor, die er gibt“. Die Prädikatenlogik ermöglicht dagegen solche allgemeinen Aussagen.

Die **Prädikatenlogik** ist der wohl am längsten und gründlichsten untersuchte Formalismus zur Repräsentation von Wissen. Prädikatenlogik erster Stufe (PL1 oder auch **FOL** nach „first order logic“) ist eine Erweiterung der Aussagenlogik. Die konzeptuelle Grundlage ist die Annahme, dass die Welt aus Objekten besteht, die bestimmte Eigenschaften besitzen. Objekte können miteinander über Relationen in Beziehung stehen. Diese werden mittels Prädikat- und Funktionssymbolen ausgedrückt, die über die ihnen zugeordnete Interpretation ihre Bedeutung bekommen. Mit Hilfe von Quantoren ($\forall$,$\exists$) können Eigenschaften ganzer Objektmengen beschrieben werden. Weitere logische Konnektoren ($\neg$, $\vee$, $\wedge$, $\rightarrow$) dienen (ähnlich zur Aussagenlogik) zur induktiven Konstruktion von komplexen Formeln. Die Semantik der Symbole ist durch die Sprachdefinition genau festgelegt. Es existieren korrekte und vollständige Inferenzmechanismen für die PL1. Der wohl bekannteste ist der *Resolutionskalkül*. Mit ihm kann indirekt geprüft werden, ob eine Formel

aus einer Wissensbasis abgeleitet werden kann. Auf diese Weise lassen sich mit Inferenzmaschinen korrekte und vollständige Antworten auf sehr umfangreiche Anfragen ableiten. Die meisten Strategien zum Schlussfolgern können jedoch im Allgemeinen nur mit exponentieller Komplexität realisiert werden. So können Theorembeweiser schon bei kleinen trickreichen Problemen versagen, die nur wenige Formeln umfassen. Zudem ist die Prädikatenlogik erster Stufe in seiner klassischen Form semi-entscheidbar, d.h. falls ein Theorem nicht bewiesen werden kann, stoppt der Algorithmus nicht. Desweiteren darf die Wissensbasis keine Widersprüche enthalten, da sonst prinzipiell jede Formel aus der Wissensbasis abgeleitet werden kann (siehe dazu auch [SS98], [Pla94]).

Die Prädikatenlogik zweiter Ordnung (oder oft auch Higher-Order Logic, **HOL**, genannt) erweitert die First-Order-Logic um einige Punkte: Zum einen können Quantoren nicht mehr nur über Variablen, sondern auch über Prädikate gelegt werden. Andererseits können Prädikate in HOL andere Prädikate als Argument erhalten. Ein Higher-Order-Prädikat der Ordnung n hat also ein oder mehr Prädikate der Ordnung n-1 als Argument (wobei $n > 1$). HOL besitzt eine bessere Ausdrucksstärke, aber aufgrund der vielfältigeren Prädikate ist es für viele Anwendungen schwerfälliger und daher meist ungeeignet.

Die **Beschreibungslogik** (auch Description Logic, **DL** genannt) stellt eine Kompromisslösung dar zwischen der Ausdrucksmächtigkeit der Sprache und der Möglichkeit v.a. entscheidbar und effizient Inferenz zu betreiben. Die Beschreibungslogik (manchmal auch als terminologische Logik bezeichnet) erlaubt die Spezifikation von terminologischen Hierarchien und verwendet eine eingeschränkte Menge von Formeln der Prädikatenlogik erster Stufe. Dabei repräsentieren Begriffsymbole, Objekte und Rollensymbole binäre Relationen, über die zwei Objekte miteinander in Beziehung gesetzt werden können. Zusätzlich gibt es noch sogenannte Begriffskonstrukturen ($\sqcap$, $\sqcup$), mit welchen man Begriffe und Rollen zu komplexeren Formeln, sog. Begriffsausdrücken zusammenfügen kann. Insbesondere sind die Terme im Unterschied zur FOL variablenfrei. Ein Beispiel einer komplexeren Formel wäre *„Professor hält Vorlesung“* (wobei *Professor* und *Vorlesung* Objekte sind, die mittels der Relation *hält* verbunden sind) und *„Student besucht Vorlesung“* (ebenfalls mit *Student* und *Vorlesung* als Objekt). Dadurch kann ein Zusammenhang zwischen Professor und Student gefunden werden. Im Semantic Web-Bereich bauen die meisten Sprachstandards auf der Beschreibungslogik auf (wie beispielsweise im Namen OWL DL für *Description Logic* erkannt werden kann).

verwendete Programmiersprachen bei der Implementierung von Inferenzmaschinen

Eine weitere mögliche Unterteilung der vorhandenen Inferenz-Maschinen ist nach Art der verwendeten Programmiersprachen. Die Inferenzmaschinen wurden funktional, prozedural oder objektorientiert programmiert. Für die Integration der Inferenzmaschine in ein bestehendes Programm ist es daher meist am sinnvollsten eine Inferenzmaschine zu wählen, die mittels des selben Programmierparadigmas erstellt wurde. Natürlich ist es auch möglich bei diversen Inferenzmaschinen (die entsprechende Interfaces bieten) auch mit anderen Programmiersprachen zu arbeiten, als der, die für die Erstellung des Inferencers verwendet wurde. Viele Inferenzmaschinen lassen sich eigenständig auf einem System installieren und mittels wohl definierter Schnittstellen ansprechen, andere hingegen müssen durch direkte Aufrufe im Programmcode initialisiert werden und erlauben meist eine leichtere Integration in ein bestehendes System. Abbildung 4.2 zeigt eine Übersicht über verfügbare Inferenzmaschinen sowie die dort verwendeten Logikarten bzw. Programmiersprachen.

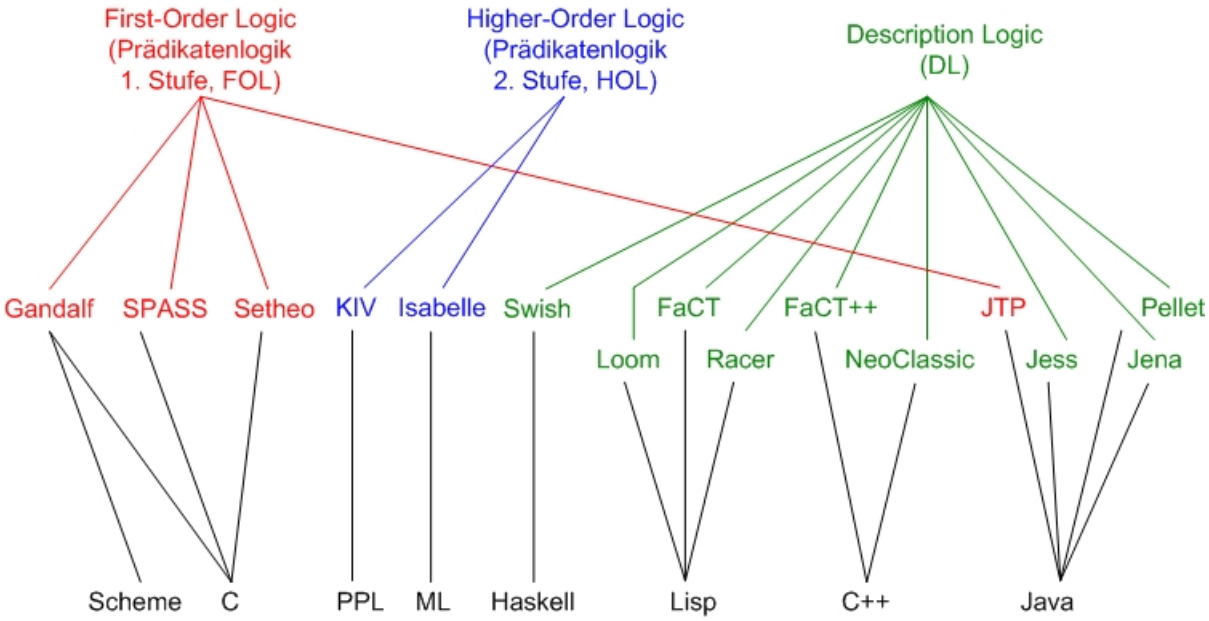


Abbildung 4.2: Eine Übersicht von Inferenz-Maschinen

4.2.3 Kurzbeschreibung von ausgewählten Inferenzmaschinen

KIV

Das System KIV (Karlsruhe Interactive Verifier) wurde an der Universität Ulm und zuletzt an der Universität Augsburg entwickelt. Das KIV-System ist ein System zur Spezifikation und Verifikation von Software Systemen. Hierbei handelt es sich um einen interaktiven Beweiser, der auf Prädikatenlogik zweiter Ordnung (HOL) basierend auf dem Sequenzenkalkül im Gentzen-Stil aufbaut. Nebenläufige Programme werden durch Temporallogik ebenso unterstützt wie dynamische Logik-Regeln. Zur algebraischen Spezifikation setzt KIV auf abstrakten Zustandsmaschinen (abstract state machines, ASM) auf. Beweise in KIV können allein durch die in Java geschriebene Oberfläche interaktiv durchgeführt werden. Der Beweiser selbst wurde in PPL implementiert, einer ML-ähnlichen funktionalen Programmiersprache. KIV bietet momentan keine Möglichkeit mit Konstrukten des Semantic Web umzugehen und auf Basis der Beschreibungslogik Schlüsse zu ziehen (siehe [HBB+05] sowie [BRSS98]).

Setheo

Setheo steht für SEquential THEOrem prover und wurde von der TU München entwickelt. Es basiert auf der Prädikatenlogik erster Stufe in Verbindung mit dem Tableaux-Kalkül und wurde sowohl in C als auch in Prolog entwickelt. Es wird momentan in einem DFG-Projekt (Deutsche Forschungsgemeinschaft) names E-SETHEO weiterentwickelt und ersetzt dabei das bisher eingesetzte Programm SPASS des Max-Planck-Instituts Saarbrücken durch eine eigene Inferenzmaschine namens E (Equational Theorem Prover), welches ebenfalls auf C basiert. Siehe dazu auch [LGM+02] und [Sch00].

Loom

Das Projekt Loom wurde 1986 an der University of Southern California unter Sponsoring der DARPA begonnen. Die Inferenzmaschine wurde mit Lisp auf Basis zweier Sprachen implementiert: auf Basis von Prädikatenlogik erster Stufe mithilfe von KIF (Knowledge Interchange Format) und basierend auf der Beschreibungslogik im KRSS/DARPA-Projekt. Es nutzt dabei gleichermaßen forward chaining als auch backward chaining: Vorwärtsverkettung um durch das Modell zu gehen und neues Wissen zu generieren und Rückwärtsverkettung, um Anfragen im Query-System zu verarbeiten.
Loom wurde im Projekt PowerLoom weiterentwickelt und baut auf der eigens dafür entwickelten Programmiersprache STELLA (a Strongly-Typed Lisp-like LAnguage) auf, die sowohl in Lisp, als auch in C++ oder Java übersetzt werden kann. PowerLoom ist momentan in Version 3.0.2 verfügbar und wird speziell für die Sprachen RDF und OWL weiterentwickelt.

FaCT, FaCT++

Die Inferenz-Maschine FaCT (kommt von Fast Classification of Terminologies) wurde ursprünglich auf Basis von Common Lisp entwickelt und unter dem Namen FaCT++ für C++ weiterentwickelt. Der FaCT++ DL Reasoner ist unter http://wonderweb.semanticweb.org verfügbar und verwendet (wie der Name bereits andeutet) Description Logic. Aber er bietet auch ein Interface, um mit First-Order Logic zu arbeiten und kann folgende Aufgaben erfüllen:

- Überprüfung der Konsistenz einer Ontologie
- Überprüfung der Erfüllbarkeit eines einzelnen Konzepts oder einer Gruppe von Konzepten
- Überprüfung der Beziehung zwischen zwei Konzepten
- Klassifizieren einer ganzen Ontologie und erstellen einer Taxonomie [TH01]

FaCT++ wird momentan für den Semantic Web-Bereich weiterentwickelt und kann ein Reasoning auf Basis von OWL durchführen. Dafür wird aber eine laufender FaCT-Prozess auf dem Rechner benötigt, dem man dann Fakten und Regeln übergeben kann.

Jena

Bei Jena handelt es sich um ein Open-Source Java-Framework vom Hewlett-Packard Labs Semantic Web Programm um semantische Informationen einzulesen und zu verarbeiten (siehe auch http://jena.sourceforge.net). Mit Hilfe von Jena können Programme sowohl RDF als auch OWL-Klassen und Properties einlesen und verarbeiten. Jena besitzt eine RDF- und OWL-Inferenzmaschine und liegt unter anderem als PlugIn für die Entwicklungsumgebung Eclipse vor. Ein Reasoning auf Basis von SWRL ist in Jena momentan noch nicht möglich, Jena kann aber benützt werden, um andere Inferenzmaschinen, die auf Basis von Jena (momentan in Version 2.2) arbeiten, anzubinden wie beispielsweise Jess, Racer oder Pellet. Der integrierte Reasoner in Jena basiert auf einer hybriden Architektur und verwendet für das forward chaining den Rete-Algorithmus und für das backward chaining einen Algorithmus auf Basis von Logic Programming (LP). Die Arbeit mit Jena ist aufgrund der sehr ausführlichen API (application programming interface) und vielen Beispielen auf der Website sehr schnell erlernbar.

Java Theorem Prover

Der Java Theorem Prover JTP wurde, wie der Name bereits sagt, in Java implementiert und besitzt eine hybride Reasoning-Architektur. Er unterstützt backward chaining-Reasoner um Anfragen auszuführen und den Beweis für eine Antwort zu ermitteln, sowie forward chaining-Reasoner um neue Informationen der Wissensbasis hinzuzufügen und Schlussfolgerungen zu ziehen. Der JTP ermöglicht es Unterklassen von Reasonern einzubinden, um mittels eines Dispatchers Anfragen an verschiedene Reasoner zu verteilen. Der JTP baut auf dem Jena-Framework auf und kann sowohl DAML+OIL als auch OWL-Konstrukte verarbeiten. Der Java Theorem Prover setzt als einzige Inferenz-Maschine unter Java auf der First-Order Logic auf (weitere Informationen unter [FFJ03], [FFJ04]).

Jess

Die Java Expert System Shell JESS (aktuell: Version 7.0a6) wird von den Sandia National Laboratories in Livermore, California, USA entwickelt (siehe: http://herzberg.ca.sandia.gov/jess). Es wurde in der Programmiersprache Java implementiert und liefert eine sehr detaillierte API mit. Es implementiert den Rete-Algorithmus und erweitert diesen um Optimierungen, backward chaining und einem Konstrukt namens „defquery“ um direkte Anfragen an die Wissensbasis zu stellen. Jess baut ebenfalls auf Jena auf und es existieren bereits Übersetzer von OWL-S nach JESS. JESS ist ein eigenständiges Programm, welches von anderen Systemen mittels Interface angesprochen werden kann. Dort erwartet JESS die Regeln, Axiome und Instanzen einer Ontologie als Fakten. Es ist also keine direkte Verarbeitung von OWL-Konstrukten aus Jena heraus möglich.

Pellet

Eine weitere Inferenzmaschine auf Basis von Java ist Pellet, welches in der MINDSWAP Research Group der University of Maryland, USA entwickelt wird. Pellet basiert ebenfalls auf Jena, kann direkt die dort eingegebenen Fakten übernehmen und Anfragen auf deren Basis bearbeiten. Es ermöglicht eine „Reparatur“ von OWL-Full auf OWL-DL, falls eine Anfrage in OWL-Full nicht zu beantworten wäre. Pellet basiert auf Tableaux-Algorithmen für ausdrucksstarke Beschreibungslogik [HST00] und steht unter der Lizenz des MIT, der Quelltext kann also einfach eingesehen werden. Pellet wird im Moment für SWRL weiterentwickelt und stellt bereits für ein Reasoning auf OWL oder RDF-Basis eine ausführliche Java API zur Verfügung.
[PS04] bietet einen guten Überblick über Pellet und um einen Artikel über Semantic Search Engines zu zitieren [Lin05]: „Pellet tends to beat other reasoners on functionality“.

4.2.4 Auswahl einer Inferenzmaschine

Da im Semantic Web Bereich beinahe ausschliesslich mit Beschreibungslogik gearbeitet wird (OWL DL!), sind für unsere Zwecke primär die Inferenzmaschinen auf dieser Basis interessant. Da der zu implementierende Prototyp für die Modellierung und Synthese von Geschäftsprozessen in Java implementiert werden soll, scheint es am sinnvollsten zu sein, sich die dort verwendeten Programme genauer anzusehen.
Der Java Theorem Prover bietet zwar eine OWL-Unterstützung, basiert aber auf First-Order logic und ist nur umständlich anzusprechen, weshalb dieser nicht geeignet erscheint. Jena ermöglicht eine einfache Bearbeitung von OWL-Fakten und kann auch für den Standard OWL-S erweitert

werden, weshalb es als Framework verwendet wird. Darauf aufbauend gibt es verschiedene Alternativen: den integrierten Reasoner, der allerdings noch keine Unterstützung von Regelsprachen wie SWRL bietet. JESS bietet eine Möglichkeit OWL in JESS-Fakten zu übersetzen, aber eine einfache Übernahme der Daten aus Jena ist nicht möglich. Es wäre ein zusätzliches Programm zu installieren, welches von dem Tool für die Modellierung angesprochen werden muss. Dies könnte einerseits zu Problemen mit Firewalls führen, und führt andererseits eine weitere mögliche Fehlerquelle dem System hinzu. Pellet besitzt eine gute API und wird momentan für neue Standards wie SWRL weiterentwickelt. Es ist einfach auf Jena aufzusetzen und kann die Daten direkt von Jena übernehmen. Es steht unter einer quelloffenen Lizenz und kann einfach angesprochen werden.
Aus Gründen der Einfachheit wird für die Entwicklung das Jena Semantic Web Framework hergenommen und auch für das Reasoning verwendet, da die Fähigkeiten des internen Reasoners für die Zwecke dieser Arbeit ausreichend sind. Sollte in einer späteren Weiterentwicklung des Prototypen eine Unterstützung von SWRL notwendig werden, kann mit nur wenigen Änderungen der Pellet-Reasoner auf das Jena-Framework aufgesetzt werden (Die Implementierung des Prototypen erfordert nur das Ändern einer einzelnen Klasse um das Reasoning mittels Pellet zu betreiben. Dadurch wurde eine lose Kopplung der Systemkomponenten erreicht).

4.3 Matrizen

Mittels der oben angesprochenen Inferenzmaschinen werden die Ontologien (bzw. alle Werte der Datensemantik) eingelesen und dann Anfragen an die Inferenzmaschine gestellt. Die Anfragen sollen feststellen, wie gut zwei Prozesse zusammenpassen: können diese Prozesse parallel ausgeführt werden, muss ein Prozess nach einem anderen Prozess ausgeführt werden oder kann ein Prozess nur alternativ zu einem anderen auftreten? Auf Basis dieser Ergebnisse wird eine Bewertung vorgenommen, wie gut zwei Prozesse zusammenpassen und diese in einer Matrix eingetragen. Durch die Benutzung von Matrizen wird eine Entkopplung des Inferenzvorganges von der eigentlichen Synthese erreicht. So könnten einmal berechnete Synthese- und Identity-Matrizen für zahlreiche Synthesevorgänge genutzt werden, um Rechenzeit zu sparen (die Berechnung der Matrizen benötigt die meiste Laufzeit). Ausserdem muss bei einer geringen Änderung im Diagramm nicht mehr alles neu berechnet werden, sondern nur die Zeile und Spalte der Matrix, in der der veränderte Prozess steht.

4.3.1 Synthese-Matrix

Konkret wird die Kongruenz zweier Prozesse auf folgende Eigenschaften überprüft: ist der Input von Prozess A derselbe oder eine Untermenge des Outputs von Prozess B? Ausserdem wird überprüft ob die Preconditions von Prozess A nach Ausführung des Prozesses B (und Eintreten aller Effects von B) erfüllt sind, ob eine Teilmenge der Preconditions erfüllt sind oder sogar eine Obermenge. Das Ergebnis wird als numerischer Wert in einer Matrix (der sogenannten Synthese-Matrix) gespeichert. Bei der Synthese-Matrix handelt es sich um eine $n \times n$-Matrix, wobei n die Anzahl der Prozesse im vorliegenden Modell ist.
Die Einträge innerhalb der Matrix werden beim Vergleich zweier Prozesse durch die Inferenz-Maschine ermittelt. Die einzelnen Werte wurden absichtlich nicht laufend durchnummeriert, um bei einer Addition der Grundwerte sehr schnell erkennen zu können, welche Grundwerte verwendet wurden. Die ersten beiden Grundwerte (2 und 4) betrachen den logischen Zusammenhang zweier Prozesse anhand ihrer Inputs und Outputs. Passen die Outputs des ersten Prozesses mit den Inputs des Zweiten zusammen, wird bei Kongruenz der Eigenschaften der Wert 4 gewählt, andernfalls

(die Inputs des zweiten Prozessen entsprechen nur einer Teilmenge der Outputs von Prozess eins) der Wert 2. Analog dazu werden die beiden nächsten Werte (5 und 6) durch den Vergleich der Preconditions mit den Effects ermittelt. Im Detail bezeichnen die Werte jeweils folgenden Zustand:

0: In der Synthese-Matrix an Stelle [x][x] handelt es sich entweder um denselben Prozess und daher ist das Matching eines Prozesses mit sich selbst nicht relevant oder es wurde kein Matching zwischen den beiden Prozessen gefunden.

2: Der Input des zweiten Prozesses ist durch den Output des ersten Prozesses erfüllt (später kurz auch als IOsim = Input-Output-similar bezeichnet)

4: Der Input des zweiten Prozesses entspricht genau dem Output des ersten Prozesses (später als IOsame betitelt, Input und Output sind „the same“).

5: Die Preconditions des zweiten Prozesses werden durch die Effects des ersten Prozesses erfüllt, es handelt sich also beispielsweise um eine Teilmenge der Effects (später auch als PEsim bezeichnet)

6: Die Preconditions und Effects passen genau zusammen (PEsame)

Darauf aufbauend ergeben sich durch Addition der vier Grundwerte folgende Werte:

7: Sowohl ein IOsim- als auch PEsim-Zustand liegt vor (mathematisch ausgedrückt: I<O, P<E - IOPEsim)

8: Es liegt ein IOsim- und PEsame-Zustand vor.

9: Hier liegt ein IOsame- und ein PEsim-Zustand vor.

10: Sicherlich der beste Zustand: sowohl Input und Output, als auch Precondition und Effect passen perfekt zueinander (IOPEsame).

Die Werte von 7 und mehr lassen sich durch Addition der jeweils darunterliegenden Werte berechnen. Dies wurde wegen implementierungs-technischen Vorteilen so gewählt. Hiermit kann nämlich durch Betrachten der Werte innerhalb der Synthese-Matrix direkt festgestellt werden, wie sich diese zueinander verhalten (z.B. PEsim = 5). Bei einer laufenden Durchnummerierung hätte man dies nicht sofort erkennen können (1+4 oder 2+3).
Sollte bereits eine Verbindung zwischen zwei Prozessen im Diagramm fest vorgegeben sein, wird ein Wert von 15 in der Matrix gesetzt. Auch werden Vorgaben der Ausführungssemantik hier berücksichtigt und ebenfalls der Wert 15 gesetzt, wenn auf Prozess A beispielsweise direkt Prozess B folgen soll.

Abbildung 4.3 zeigt zwei Prozesse A und B. Die Effects des Prozesses A entsprechen direkt den Preconditions von B, hingegen die Inputs und Outputs der beiden Prozesse sind nicht gleich. Die Inputs von Prozess B sind nur eine Teilmenge der Outputs von Prozess A. Daher wären die hier beschriebenen Prozesse im Zustand PEsame und IOsim, würden also in der Synthese-Matrix in Zeile (Nummer von A) und Spalte (Nummer von B) den Wert 8 erhalten.

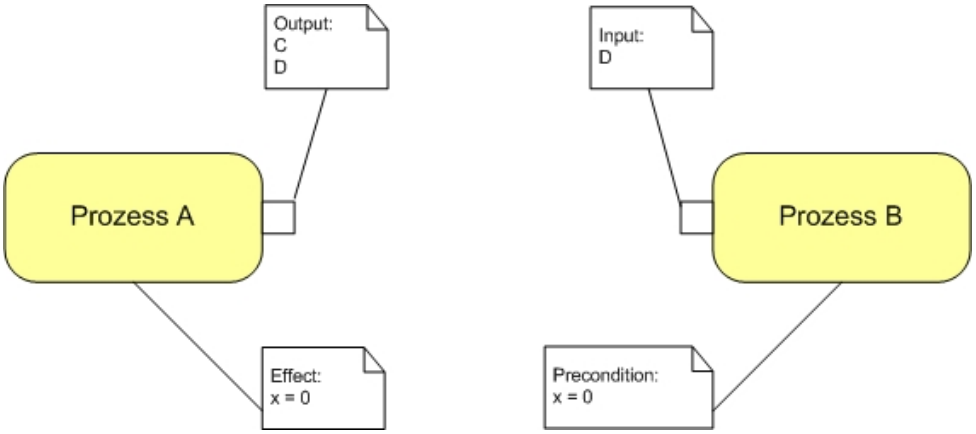


Abbildung 4.3: Erstellung der Synthese-Matrix

4.3.2 Identity-Matrix

Gleichzeitig muss überprüft werden, ob zwei Prozesse sich dafür eignen parallel ausgeführt zu werden. Parallelisierbar sind zwei Prozesse, wenn sie sich in den Parametern stark gleichen. So sind beispielsweise in der Fallstudie des vorherigen Kapitels der Prozess „Flug buchen“ mit dem „Hotel buchen“ vermutlich stärker parallelisierbar, als die Prozesse „Geschäftsreise planen“ und „Geschäftsreise durchführen“. Sowohl die Flugbuchung als auch die Hotelbuchung benötigen die Anfangs- und Enddaten des Urlaubs sowie das Urlaubsland und speichern nach erfolgreicher Ausführung ihre Ergebnisse in einer globalen Variablen. Dafür werden mittels des Reasoners die Inputs, Outputs, Preconditions und Effects zweier Prozesse verglichen und wieder mit einem numerischen Wert benotet. Ähnlich wie bei der Synthese-Matrix ist der Wert innerhalb der Matrix sehr hoch, wenn sich zwei Prozesse sehr stark parallelisieren lassen. Dabei bezeichnen folgende Werte die Parallelisierbarkeit in der zu erstellenden Identity-Matrix:

0: Findet man in der Diagonalen der Matrix, um anzudeuten, dass es sich hier um denselben Prozess handelt, der natürlich alle Parameter mit sich selbst gleich hat oder um zu zeigen, dass diese beiden Prozesse nicht parallelisierbar sind.

1: Sind die Inputs zweier Prozesse gleich, ist der Wert 1.

2: Die Outputs zweier Prozesse sind gleich.

4: Die Preconditions zweier Werte sind gleich.

8: Die Effects sind gleich.

15: Alle Parameter sind identisch also sowohl Inputs, Outputs, Preconditions und Effects passen ideal zueinander. Dieser Prozess sollte parallel ausgeführt werden.

16: Ist die Parallelität bereits in der Ausführungssemantik fest vorgegeben, erhält die Matrix an dieser Stelle den Wert 16.

Ausserdem sind noch verschiedene andere Werte zwischen 8 und 15 denkbar, die sich ergeben, wenn zwei oder drei Parametergruppen identisch sind. Auch hier wurde absichtlich keine laufende Durchnummerierung gewählt, um bei Betrachtung eines Wertes direkt die betroffenen Komponenten sehen zu können, die in beiden Prozessen gleich sind. So wäre aus der Zahl 11 klar ersichtlich, dass in beiden Prozessen sowohl Inputs, Outputs als auch Effects identisch wären.

4.4 Syntheseverfahren

Sind mittels der Inferenzmaschinen die oben eingeführten zwei Matrizen Synthese-Matrix und Identity-Matrix erstellt, werden diese Matrizen als gerichtete, gewichtete Graphen interpretiert. Die Knoten in diesem Graphen sind die vorliegenden Prozesse und die Kanten sind dann gegeben, wenn ein Eintrag in der Synthese-Matrix > 0 ist (im Moment wird die Identity-Matrix noch nicht benötigt).
Am Beispiel der folgenden Synthese-Matrix (Abbildung 4.4) würde der danebenliegende Graph entstehen und ein korrekter Algorithmus müsste als besten Weg 1-2-3 erkennen: Beim Erstellen der Synthese-Matrix wurde also eine sehr gute Aufeinanderfolge von Prozess 3 auf Prozess 2 festgestellt (Wert 10 in Zeile 2 / Spalte 3). Beinahe gleich gut folgt Prozess 2 auf Prozess 1 (Wert 7 in Zeile 1 / Spalte 2). Hingegen eher schlecht passen der Prozess 3 und der Wert 1 (Möglichkeit, dass 1 auf 3 folgt nur bei 2). Die übrigen Kombinationen der Prozesse scheinen nicht möglich zu sein (jeweils Wert 0). Wird die Matrix als Graph interpretiert, hat dieser Graph zwischen den drei Prozessen drei Kanten, nämlich genau diejenigen, welche einen Wert grösser als 0 in der Matrix haben. Von diesen Kanten werden die Besten genommen, bis alle Prozesse besucht sind (je nach Art des Algorithmus). Daher würde als Lösung der Weg 1-2-3 erkannt werden.

	1	2	3
1	0	7	0
2	0	0	10
3	2	0	0

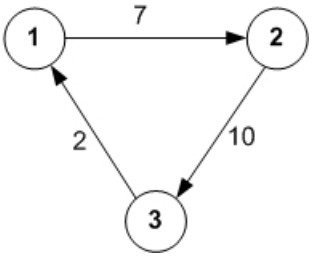


Abbildung 4.4: Beispiel einer Synthese-Matrix und dem dazugehörigen Graphen

Auf diesem Graphen können verschiedene Algorithmen angewendet werden. Im Rahmen dieser Arbeit wurden verschiedene Algorithmen entwickelt und auf ihre praktische Anwendbarkeit getestet und evaluiert. Das System wurde so gestaltet, dass für die zukünftige Arbeit auch relativ einfach neue Algorithmen getestet und eingebaut werden können. Für die Beschreibung der Algorithmen bedarf es erst einiger Definitionen:

Definition 4.1. *Eine **Kante** liegt dann zwischen zwei Knoten vor, wenn der Wert in der Synthesematrix > 0 ist. Der **Arbeitsraum** ist der komplette Graph mit allen Kanten, der durch die Synthese-Matrix und Identity-Matrix erzeugt wird. Eine **Lösungsinstanz** ist ein Weg durch den Arbeitsraum, der alle Knoten durchläuft und sich eventuell zwischendurch aufteilt. Die **Lösungsmenge** ist die Menge aller möglichen Lösungsinstanzen. Für jede Lösungsinstanz kann aufgrund der verwendeten Kanten und der Art ihrer Verwendung (hängt davon ob, ob die Kanten in einem Einzelstrang oder in einer Parallelität durchlaufen wird) durch den Kantenwert eine **Bewertung** errechnet werden. Eine **optimale Lösung** ist eine Lösungsinstanz mit maximaler Bewertung.*

Bevor ein Algorithmus gestartet wird, wird überprüft ob die Prozesse im Diagramm auch eine Synthese ermöglichen. Daher wird überprüft, ob im Diagramm Kreise vorkommen, ob auf einen Parallelisierungsknoten immer irgendwann ein Synchronisationsknoten folgt und ob die existierenden Kanten im Graphen der Ausführungssemantik widersprechen. Existieren bereits Kanten im Graphen kann der User auswählen, ob er diese für die Synthese übernehmen möchte (sinnvoll, wenn nur wenige Kanten im Diagramm existieren) oder ob er eine vollautomatische Synthese

ohne Berücksichtigung der bestehenden Kanten wünscht. Diese Überprüfung wird mittels eines *Recursive Descent-Parsers* durchgeführt, der die Grammatik aller Teile des Diagramms testet. Entsprechen alle Teile der Grammatik wird (je nach ausgewählten Präferenzen) ein Algorithmus gestartet. Momentan existieren zwei unterschiedliche Algorithmen, die im Weiteren genauer erklärt und im folgenden Kapitel evaluiert werden.

4.4.1 Modified Prim

Der erste Algorithmus, der im Rahmen dieser Arbeit verwendet wird, ist ein leicht veränderter Prim-Algorithmus [1] und hat im schlechtesten Fall eine Komplexität von $O(n \cdot n!)$. Der Algorithmus von Prim erstellt einen minimalen Spannbaum und wählt dabei immer die Kante mit dem geringsten Wert aus und fügt diese dem bereits vorhandenen Graphen hinzu. Dieser Algorithmus wurde als Grundlage hergenommen und an die Erfordernisse von Aktivitätsdiagrammen angepasst. Dafür wird nicht mehr nur ein einzelner Weg durch den Pfad gesucht, sondern auch erlaubt, dass mehrere parallele Wege durch den Graphen möglich sind. Ausserdem wird nicht mit der kleinsten Kante begonnen, sondern zuerst die größte Kante der Lösungsmenge hinzugefügt. Als nächstes wird die zweitbeste Kante genommen (mit dem nächstgrössten Kantenwert) und geprüft, ob diese der Lösungsmenge hinzugefügt werden kann. Dabei darf diese Kante nicht gegen folgende Regeln verstossen:

- Die neue Kante darf keinen Kreis erzeugen, es darf also durch die bisher vorhandenen Kanten (die ja gerichtet sind) beim Hinzufügen der neuen Kante nicht möglich sein vom Endpunkt der neuen Kante zum Startpunkt der Kante zu gelangen.
- Sollte die Kante eine Parallelität erzeugen, muss mindestens ein Prozess in jedem parallelen Strang vorhanden sein. Ein optionaler Prozess ist nicht erlaubt. Davor wird überprüft ob die dadurch entstehenden parallelen Prozesse gemäß der Identity-Matrix überhaupt parallelisierbar sind.
- Die neue Kante darf nicht gegen die Bedingungen der Ausführungssemantik verstossen. Es darf also beispielsweise keine Kante von A nach B hinzugefügt werden, wenn in der Ausführungssemantik festgelegt war, dass A nach B kommen muss.

Dieses Vorgehen wird iterativ durchgeführt, bis alle Prozesse einmal besucht wurden. Alle Prozesse haben also mindestens eine eingehende und eine ausgehende Kante. Ausnahmen sind zwei Prozesse: der Start- und der Endprozess, die natürlich keine eingehenden (Startprozess) bzw. ausgehenden (Endprozess) Kanten haben dürfen. Wurden alle Prozesse besucht werden die Kanten überprüft und evtl. aufgeteilt. Führen mehrere Kanten aus einem Prozess heraus, wird ein Parallelisierungsknoten oder ein Entscheidungsknoten eingefügt abhängig von den Preconditions der folgenden Prozesse. Nachdem auch alle Synchronisationsknoten und Verbindungsknoten hinzugefügt wurden, wird der Graph als mögliche Lösungsinstanz zurückgegeben. Um auch Lösungsinstanzen aus der Lösungsmenge erkennen zu können, die anfangs suboptimal erscheinen, weil eine Kante einen geringen Wert hat, aber im Zusammenspiel mit allen anderen Kanten vielleicht doch die optimale Lösung darstellen, wird der oben skizzierte Algorithmus von Threads ausgeführt, von denen jeder bereits eine Kante vorgegeben hat. Es werden also soviele Threads gestartet, wie es Kanten in

[1] Robert Prim, ein Mathematiker und Informatiker hat 1957 einen Algorithmus zur Erzeugung von minimalen Spannbäumen (minimal spanning tree, MST) erfunden, der später von Edsger Dijkstra auf Graphen angewandt wurde.

dem Graphen gibt und jeder Thread enthält je eine Kante als Anfangsbestand seiner Lösungsmenge. Hat jeder Thread eine Lösung gefunden, in der er alle Prozesse besucht hat, wird überprüft, ob dabei auch die Bedingungen der Ausführungssemantik eingehalten wurden, sowie (mittels der Identity-Matrix) ob in der Lösung vorkommende Parallelitäten vorkommen dürfen.
Auf dieser Basis wird die Bewertung der Lösung vorgenommen: Dazu werden alle benützten Kanten betrachtet und addiert. Als letztes meldet jeder Thread seine Lösungsinstanz sowie die Bewertung seiner Lösung und ermöglicht damit eine Sortierung und Rückgabe der optimalen Lösung an den Benutzer.

4.4.2 RandomWalk

Der zweite Algorithmus, der im Rahmen dieser Arbeit getestet werden soll, ist ein modifizierter RandomWalk (worst-case Komplexität: $O(n^2)$). RandomWalk wurde im Rahmen der Mathematik zur Optimierung von mathematischen Problemen entworfen und versucht durch einen beliebigen Startpunkt ein lokales (oder natürlich besser noch: globales) Maximum zu erreichen, indem die vorhandene Lösung schrittweise verändert wird. Dafür wird initial im Arbeitsraum eine zufällige Lösungsinstanz gesucht und von diesem Graphen beliebige Kanten entfernt und nach neuen gesucht, die eventuell eine bessere Bewertung besitzen. Ist nach n Schritten eine bessere Lösungsinstanz erreicht, wird diese als neue Ausgangsbasis übernommen und wiederum eine neue Veränderung gesucht. Um eine Veränderung vorzunehmen, werden Kanten bevorzugt, die eine hohe Bewertung besitzen. Auf diesem Weg wird versucht die optimale Lösung iterativ anzunähern.
RandomWalk merkt sich immer die bisher optimale Lösung und gibt am Ende auch nur diese zurück (im Gegensatz zu modified Prim, der die x-besten Lösungen zurückgeben kann - je nach Einstellung der Präferenzen).

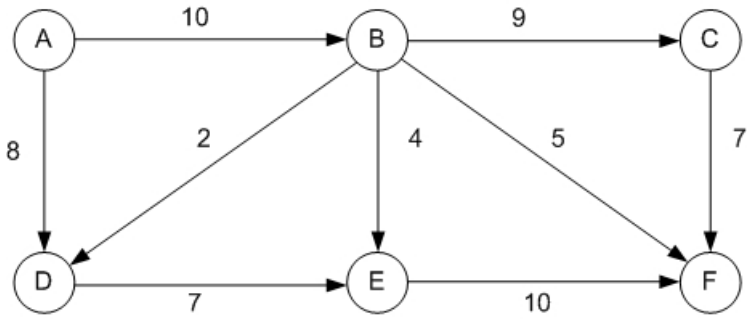


Abbildung 4.5: Beispiel des RandomWalk-Algorithmus

In Beispiel 4.5 könnte eine initale Lösungsinstanz von A zu B gehen und sich dann aufteilen in zwei Wege (über C bzw. über D und E) um zu F zu gelangen. Die Bewertung dieser Lösung wäre 45 (10+2+7+10+9+7). Durch das Entfernen der Kanten von B nach C und von C nach F könnte der Algorithmus die Kante von B nach F mit dem Wert 5 hinzufügen. Damit wären aber nicht alle Prozesse besucht und daher würde diese Lösung verworfen. Alternativ könnte auch die Kante von B nach D entfernt werden und dafür die Kante von A nach D verwendet werden. Diese Lösungsinstanz hätte eine Bewertung von 51 (10+9+7+8+7+10) und wäre damit besser als die vorherige. Nachdem auch durch weitere Veränderungen keine bessere Lösung gefunden werden kann, wird diese Lösung von RandomWalk zurückgeliefert.

Wurde mit einem der beiden Algorithmen eine möglichst optimale Lösung gefunden, wird diese dem Benutzer angezeigt und der Benutzer kann wählen, ob er diese übernimmt oder ob er weitere

Veränderungen am vorliegenden Graphen vornehmen möchte, um danach eine neue Synthese zu starten. Beim Algorithmus „modified Prim“ kann zusätzlich noch die Anzahl der x besten Lösungen angezeigt werden.

Abbildung 4.6 zeigt abschliessend ein Kommunikationsdiagramm, um die Zusammenhänge bei der Synthese zu verdeutlichen.

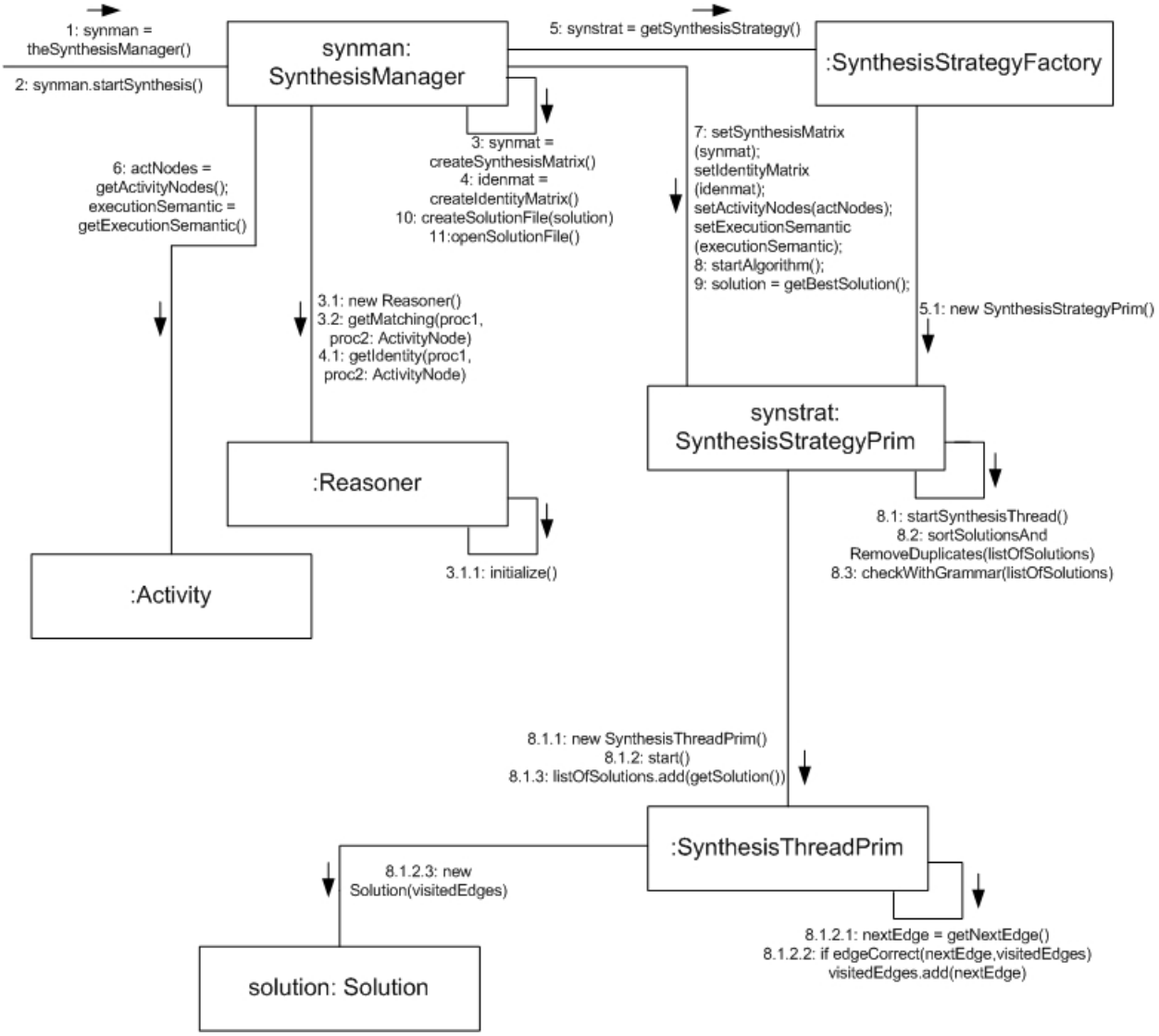


Abbildung 4.6: Kommunikationsdiagramm zur Synthese

Kapitel 5

Tool-Entwicklung

5.1 Eclipse

Um eine Ontologie-basierte Modellierung von Geschäftsprozessen vornehmen zu können, musste entschieden werden, ob ein eigenes Tool entwickelt werden sollte, oder ein bereits vorhandenes Tool um diese Funkionalitäten erweitert werden sollte. Das zu erweiternde Tool musste auf jeden Fall Diagramme darstellen können. Da es grösstenteils um UML-Diagramme geht, wurden UML-Tools genauer betrachtet. [Jec05] bietet beispielsweise eine ausführliche Übersicht über UML-Tools. Hier finden sich Tools wie Poseidon for UML, ARIS UML Modeller, EclipseUML oder auch der Innovator von MID wieder. Einige Tools unterstützen hier nicht nur eine Erstellung von graphischen Diagrammen sondern auch gleichzeitig eine Weiterverarbeitung dieser Diagramme in Programmcode. Am besten geeignet wäre ein Tool, mit welchem man einfach UML-Diagramme (aber im Bedarfsfall auch andere, wie z.B. ARIS) erstellen und bearbeiten kann und diese dort gewonnenen Informationen weiter verarbeiten kann.
Ein Tool, welches sich vor allem im Bereich der Software-Entwicklung immer mehr durchsetzt, ist Eclipse. Mit diesem Tool ist es möglich Diagramme jeglicher Art einfach zu bearbeiten und dennoch flexibel auch Veränderungen an normalen Textdateien zu ermöglichen. Durch seine offene Plugin-Struktur ist die Erweiterung von Eclipse sehr einfach. Eclipse ist sowohl eine Entwicklungsumgebung, eine Plattform für neue Programme, ein Modell-Desginer und kann auch an alle anderen Bedürfnisse eines Projektes angepasst werden. Eclipse ermöglicht die spätere automatische Quellcodegenerierung und Weiterverarbeitung aller erzeugter Dateien. Auch existieren mittlerweile diverse andere Plugins für die Arbeit mit Prozessen und Web-Services, weshalb hier sehr einfach eine Schnittstelle zu anderen Tools geschaffen werden kann.

5.1.1 Aufbau von Eclipse

Abbildung 5.1 zeigt die Struktur von Eclipse und dem erstellten PlugIn zur Ontologie-basierten Modellierung von Geschäftsprozessen. Die Eclipse-Plattform ist

> „an IDE for anything, but nothing in particular.“ [Ecl05]

Eclipse ist in erster Linie für die Entwicklung von Java-Applikationen gedacht, aber nicht ausschließlich dafür. Es ist vielmehr eine generische Plattform, die mittels Plugins um neue Funktionalität erweitert werden kann und Entwicklungswerkzeuge für verschiedenste Inhalte (C#, Java,

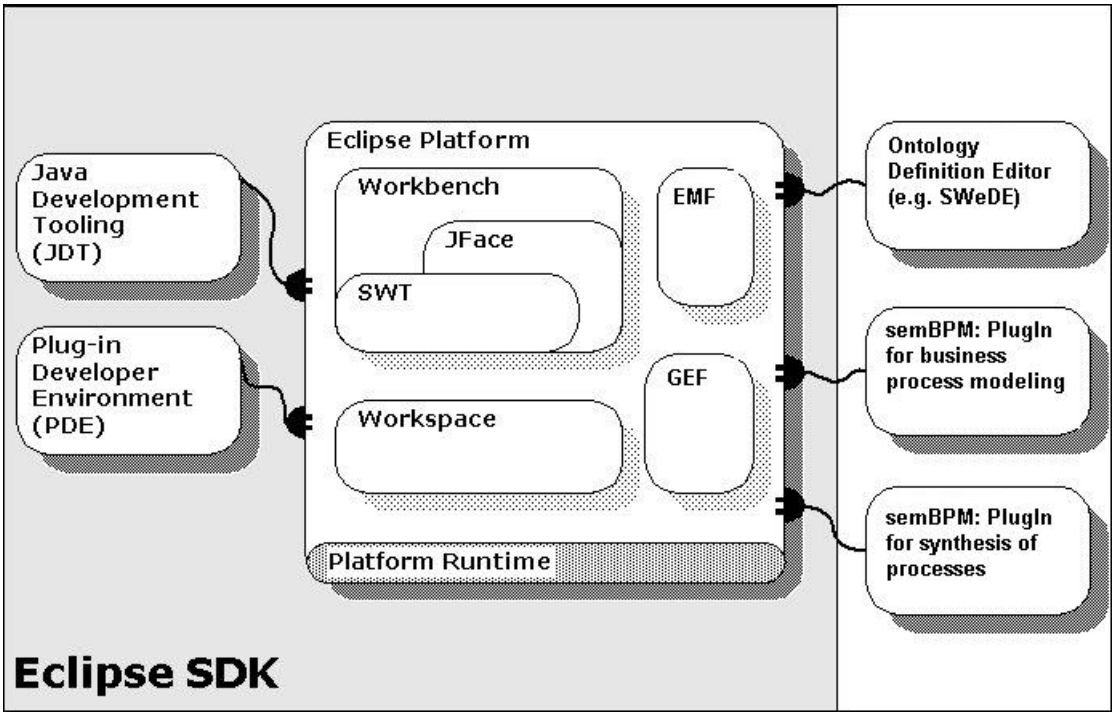


Abbildung 5.1: Die Eclipse-Architektur als Grundlage (basierend auf [Ecl05])

XML etc.) integriert. Die Plattform bietet dabei eine Menge von Grundservices an, die von Entwicklern genutzt werden können, um eigene PlugIns zu entwickeln. Diese PlugIns können in die Entwicklungsumgebung integriert werden und auf deren Basis wieder neue PlugIns geschrieben werden. Eclipse unterstützt diese Vorgehensweise durch eine spezielle Art von Projekten (PDE, Plugin Development Environment) und ein leicht zu erlernendes Konzept wie Plugins implementiert werden müssen.

Eclipse arbeitet in einem Workspace, darin sind alle benötigten Dateien gespeichert und es kann eine oder mehrere Workbenches gestartet werden. Eine Workbench ist eine Instanz von Eclipse. Die Runtime-Workbench beispielsweise wird benötigt, um neu zu entwickelnde Plugins zu testen ohne (bei einem Fehler) die normale Entwicklungsumgebung zum Absturz zu bringen. Alle Fenster innerhalb der Workbench sind mit der graphischen Bibliothek SWT (Standard Widget Toolkit) und dem darauf aufbauenden JFace erstellt. Diese Bibliotheken müssen daher auch für die Entwicklung eigener Oberflächen verwendet werden. Um graphische Modelle erstellen zu können, wird meist EMF (Eclipse Modeling Framework) zusammen mit GEF (Graphical Editing Framework) benutzt.
Zur Erleichterung seiner Arbeit stehen dem Entwickler diverse Zusätze zur Verfügung wie das oben genannte PDE um Plugins zu erstellen oder auch JDT (Java Development Tooling) speziell für Java Entwickler. In diese Plattform können jederzeit weitere benötigte Plugins installiert werden.

5.1.2 Verwendete PlugIns

Zur Erstellung und Bearbeitung von Ontologien können verschiedene Möglichkeiten in Betracht gezogen werden: Editieren innerhalb eines einfachen Text-Editors. Dieser bietet kein Syntax-Highlighting und keine Hilfe bei der Fehlersuche an. Das Plugin SWeDE von BBN Technologies

[Tec05] bietet diese Möglichkeiten an und vereinfacht dadurch die Arbeit mit Ontologien um einiges. Desweiteren gibt es von IBM das Ontology Definition Toolkit (ODT) um Ontologien zu bearbeiten, allerdings ist das Plugin sehr komplex und für die meisten Ontologien, die im Rahmen dieser Arbeit benötigt werden, zu umfangreich. Daneben gibt es natürlich noch Tools ausserhalb von Eclipse wie z.B. Protégé, welches als das Standard-Tool im Semantic Web-Bereich gilt [Inf05]. Im Moment arbeitet ein Standardisierungsgremium der OMG (Object Management Group) an ODM, dem Ontology Definition Metamodel. Dieses wird es ermöglichen eine Ontologie auch in einem graphischen Modell zu erstellen und zu bearbeiten sowie Modelltransformationen zwischen verschiedenen Meta-Modellen (UML, ER-Diagramm, TopicMaps, etc.) und der Ontologie vorzunehmen [DIS05].

Da das hier zu implementierende Plugin einen graphischen Editor zur Erstellung von Aktivitätsdiagrammen ermöglichen soll, wurde auf das Graphical Editing Framework (GEF) aufgesetzt. GEF erlaubt Entwicklern einen umfangreichen graphischen Editor auf Basis eines Modells zu entwickeln. Das Draw2d-Plugin bietet dafür die Grundlage um das Layout oder Rendering von graphischen Elementen zu steuern. GEF funktioniert gemäss dem Model-View-Controller-Pattern (MVC), welches das Modell und die Darstellung explizit unterscheidet. Das Modell weiss dabei nichts über seine Darstellung (die View), weshalb es einen Controller benötigt, der die beiden Teile aufeinander abbildet. Ein Controller wird in Eclipse als EditPart bezeichnet und übernimmt die Kommunikation zwischen dem Modell und dem View (hier Figure genannt). Das Verhalten eines EditParts wird durch sogenannte EditPolicies festgelegt, wobei hier auf bereits vordefinierten EditPolicies aufgesetzt werden kann. Eine EditPolicy ist ein Rahmenwerk, um das Verhalten von Modellelementen im Diagramm festzulegen und auf Aktionen des Benutzers reagieren zu können. Im Rahmen einer EditPolicy wird ein Command erzeugt und auf einem CommandStack ausgeführt. Dieser CommandStack ermöglicht das Rückgängigmachen von Aktionen ebenso wie eine Wiederholung einer Aktion (für weitere Informationen über GEF siehe [Ecl05]).

5.2 Semantic-based Business Process Modeling Plugin

Abbildung 5.2 zeigt den prinzipiellen Aufbau des Plugins zur Semantischen Geschäftsprozessmodellierung (SemBPM). SemBPM baut einerseits auf den von Eclipse zur Verfügung gestellten Standardbibliotheken wie SWT und JFace auf, welche in GEF und PDE ebenfalls verwendet werden, benötigt andererseits aber auch eine Inferenzmaschine (in diesem Fall Jena) zur automatischen Synthese und Arbeit mit den semantischen Informationen. Jena liegt hierbei auch als Plugin vor (in Version 2.0), da dies für die Erstellung eines eigenen Plugins auf Basis von Jena Voraussetzung ist.

Das Plugin besteht aus mehreren Komponenten: es benötigt eine zugrundeliegende Ontologie, die bei Erstellung eines neuen Projektes festgelegt werden muss. Diese Ontologie nützt die Konzepte, die bereits in Kapitel 3.2.1 beschrieben wurden. Das Plugin selbst besteht aus drei Komponenten: Activity ermöglicht die Erstellung und Bearbeitung eines Aktivitätsdiagrammes, Synthesis führt eine Synthese auf dem Aktivitätsdiagramm durch und arbeitet mit den Ontologien und mittels der Preferences können Vorgaben für Aktivitätsdiagramme oder die Synthese festgelegt werden (siehe Abbildung 5.3). Desweiteren nutzt das Plugin diverse Funktionalitäten von Eclipse, die ebenfalls als Komponenten modelliert werden könnten aber zur Vereinfachung vernachlässigt wurden.
Zu weiteren Informationen über die Installation, Bedienung oder Struktur des Programmes siehe Anhang B und E.

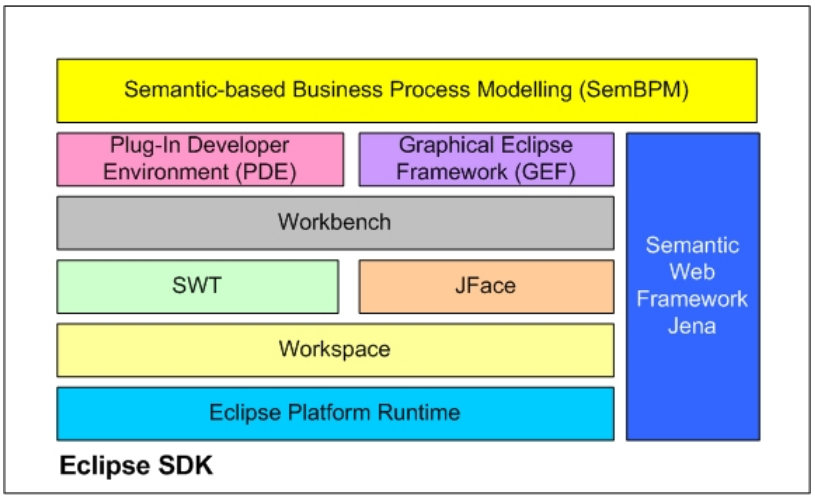


Abbildung 5.2: Schichtenaufbau des Plugins SemBPM

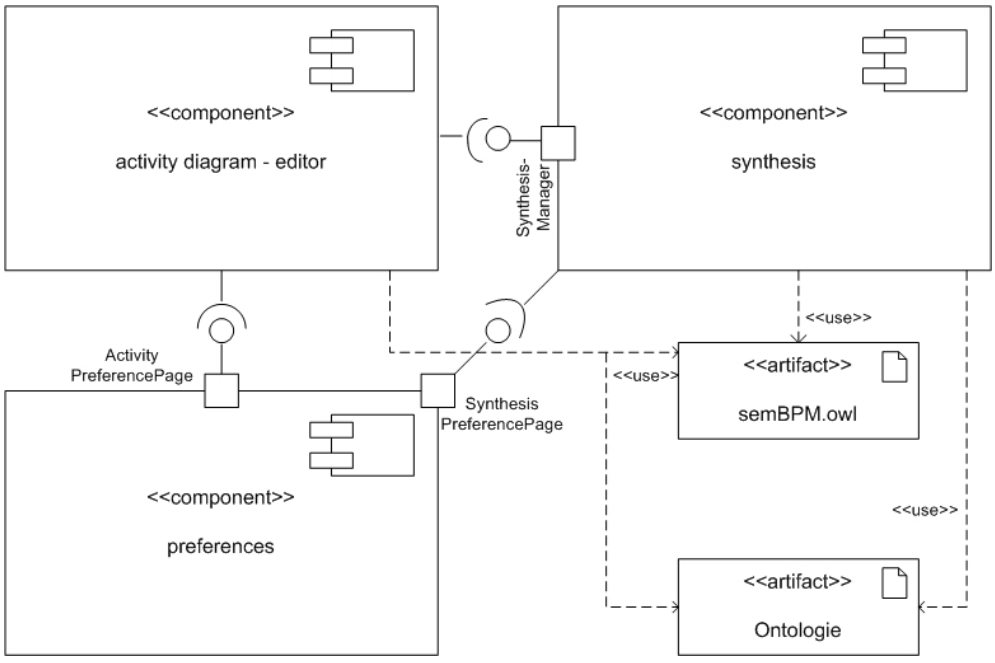


Abbildung 5.3: Komponenten des Plugins SemBPM

Kapitel 6

Evaluation

6.1 Fallstudien

Die im vorigen Kapitel vorgestellten Algorithmen sollen an verschiedenen Beispielen getestet und die Ergebnisse verglichen werden. Als Beispiele werden drei typische Geschäftsprozesse verwendet: Das Buchen einer Geschäftsreise (inkl. Buchen eines Mietwagens), die Bearbeitung eines eingehenden Auftrages sowie die Operationen, welche auf einem Bankkonto ausgeführt werden können.

6.1.1 Beispiel: Buchen einer Geschäftsreise

Das Buchen einer Geschäftsreise wurde bereits im Kapitel 3.2.2 beschrieben. Das entwickelte Tool erlaubt es zusätzlich einen Prozess als Stellvertreter mehrerer anderer Prozesse herzunehmen. So kann der im Beispiel genannte Prozess „Mietwagen buchen“ in verschiedene Teilprozesse exemplarisch aufgeteilt werden: zuerst wird eine Mietwagenvermietung gesucht und dort ein Angebot eingeholt. Ist dieses Angebot akzeptabel, also geringer als ein vorher definierter Wert, wird der Mietwagen gebucht, ansonsten wird der Mietwagen woanders gebucht. Der Prozess „Mietwagen woanders buchen“ könnte beispielsweise als Referenz für dieselbe Aktivität stehen. Die genannten Prozesse haben folgende funktionale Eigenschaften (IOPEs):
„Autovermietung suchen“ liefert als Output eine Autovermietung, welche der Prozess „Angebot einholen“ als Input benötigt und nach seiner Ausführung eine Variable Leihwagengebühr setzt. Die beiden Prozesse „Mietwagen buchen“ und „Mietwagen woanders buchen“ überprüfen diese Variable Leihwagengebühr (Leihwagengebühr biggerThan 100, Leihwagengebühr lessThan 100) und setzen nach Ihrer Ausführung die Variable Auto_gebucht auf True (Auto_gebucht setValue true). Der letzte Prozess „Mietwagenbuchung abschliessen“ verlangt, dass ein Auto gebucht wurde (mittels der angegebenen globalen Variablen) und gibt als Output das ausgewählte Auto zurück. In der Ausführungssemantik wird zusätzlich spezifiziert, dass vor dem Buchen eines Mietwagens immer ein Angebot eingeholt werden muss (Mietwagen_buchen previous Angebot_einholen).
Die Ontologie für dieses Beispiel sähe (vereinfacht) folgermassen aus:

```
<semBPM:Object rdf:ID='Autovermietung'/>

<semBPM:Object rdf:ID='Auto'/>

<semBPM:SemDataProperty rdf:ID='vermietet'>
```

```
   <rdfs:domain rdf:resource='#Autovermietung'/>
   <rdfs:range rdf:resource='#Auto'/>
</semBPM:SemDataProperty>

<semBPM:globalVariable rdf:ID='Auto_gebucht'/>

<semBPM:globalVariable rdf:ID='Leihwagengebühr'/>

<semBPM:globalVariable rdf:ID='Angebot_akzeptabel'/>
```

6.1.2 Beispiel: Bearbeitung eines Auftrages

Im zweiten Beispiel wird es um die Auftragsbearbeitung eines Unternehmens gehen. Wenn ein Auftrag eingegangen ist, wird zuerst das Lager überprüft, ob die notwendigen Rohstoffe zur Produktion des Auftragsartikels vorhanden sind. Sollten diese vorhanden sein, werden diese aus dem Lager geholt, ansonsten müssen diese Rohstoffe erst bestellt werden und erst wenn die Rohstoffe geliefert wurden, kann mit der Aktivität fortgefahren werden. Sind alle benötigten Rohstoffe vorhanden, wird das Endprodukt produziert und danach sowohl ausgeliefert als auch (vermutlich parallel dazu) eine Rechnung geschrieben und versandt.

Abbildung 6.1: Die Prozesse einer Auftragsbearbeitung

Die Prozesse (wie in Abbildung 6.1 zu sehen) haben dabei folgende funktionale Eigenschaften:

- Auftrag eingegangen:
  - Output: Auftrag
- Lager überprüfen:
  - Input: Auftrag
  - Effect: Rohstoffbestand bekommt einen Wert
- Rohstoffe bestellen:

  - Precondition: Rohstoffbestand = 0
  - Output: Bestellung
- Rohstoffe werden geliefert:
  - Input: Bestellung
  - Output: Rohstoffe
- Rohstoffe aus Lager holen:
  - Precondition: Rohstoffbestand > 0
  - Output: Rohstoffe
- Endprodukt produzieren:
  - Input: Rohstoffe
  - Output: Endprodukt, Rechnung
- Ausliefern:
  - Input: Endprodukt
- Rechnung versenden:
  - Input: Rechnung

Die Ontologie für dieses Beispiel sähe (wieder gekürzt) aus wie folgt:

```
<semBPM:Object rdf:ID='Rohstoffe'/>

<semBPM:Object rdf:ID='Lager'/>

<semBPM:Object rdf:ID='Auftrag'/>

<semBPM:Object rdf:ID='Bestellung'/>

<semBPM:Object rdf:ID='Endprodukt'/>

<semBPM:Object rdf:ID='Rechnung'/>

<semBPM:globalVariable rdf:ID='Rohstoffbestand'/>

<semBPM:globalVariable rdf:ID='Endprodukt_fertig'/>
```

6.1.3 Beispiel: Arbeiten mit einem Bankkonto

Im letzten Beispiel geht es um die prinzipiellen Vorgänge beim Arbeiten mit einem Bankkonto. Zuerst muss dieses Konto angelegt werden, dann kann man Geld einzahlen. Ist genügend Geld auf dem Konto vorhanden, kann man dieses entweder auf ein anderes Konto überweisen oder wieder abheben. Möchte man das Konto auflösen, muss man es zuerst leeren, also den Restbetrag ebenfalls abheben.

Die angesprochenen Prozesse wurden wie folgt modelliert:

- Konto erstellen:
  - Output: Konto
  - Effect: Kontostand = 0
- Geld einzahlen:
  - Input: Konto, Bargeld
  - Output: Konto
  - Precondition: Kontostand = 0
  - Effect: Kontostand hat einen neuen Wert
- Geld abheben:
  - Input: Konto
  - Output: Konto, Bargeld
  - Precondition: Kontostand > 0
- Geld überweisen:
  - Input: Konto
  - Output: Konto
  - Precondition: Kontostand > 0
  - Effect: Kontostand hat einen neuen Wert
- Konto leeren:
  - Input: Konto
  - Output: Konto, Bargeld
  - Precondition: Kontostand > 0
  - Effect: Kontostand = 0
- Konto auflösen:
  - Input: Konto
  - Precondition: Kontostand = 0

Die zugrundeliegende Ontologie besteht aus folgenden Konzepten:

```
<semBPM:Object rdf:ID='Bargeld'/>

<semBPM:Object rdf:ID='Konto'/>

<semBPM:Object rdf:ID='Bank'/>

<semBPM:Object rdf:ID='Girokonto'>
   <rdfs:subClassOf rdf:resource='#Konto'/>
   <owl:disjointWith rdf:resource='#Sparbuch'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Sparbuch'>
```

```
   <rdfs:subClassOf rdf:resource='#Konto'/>
   <owl:disjointWith rdf:resource='#Girokonto'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Zins'/>

<semBPM:Object rdf:ID='BLZ'/>

<semBPM:Object rdf:ID='Kontonummer'/>

<semBPM:Object rdf:ID='Kunde'/>

<semBPM:SemDataProperty rdf:ID='hat_BLZ'>
   <rdfs:domain rdf:resource='#Bank'/>
   <rdfs:range rdf:resource='#BLZ'/>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='hat_Zins'>
   <rdfs:domain rdf:resource='#Konto'/>
   <rdfs:range rdf:resource='#Zins'/>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='hat_Konto'>
   <rdfs:domain rdf:resource='#Kunde'/>
   <rdfs:range rdf:resource='#Konto'/>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='hat_Kontonummer'>
   <rdfs:domain rdf:resource='#Konto'/>
   <rdfs:range rdf:resource='#Kontonummer'/>
</semBPM:SemDataProperty>

<semBPM:globalVariable rdf:ID='Kontostand'/>

<semBPM:globalVariable rdf:ID='Kontobewegung'/>
```

Wie hier deutlich sichtbar wird, sind bei der Modellierung und auch bei der Erstellung der Ontologie keine Grenzen gesetzt. In der Ontologie können auch Konzepte definiert sein, die im Aktivitätsdiagramm nicht verwendet werden.

6.2 Tests und Auswertung

Beim Buchen einer Geschäftsreise und dem damit verbundenen Buchen eines Mietwagens gelangen beide Algorithmen (modified Prim und RandomWalk) auf dasselbe Ergebnis, welches man auch bei der Modellierung erwartet hat. Abbildung 6.2 zeigt das fertige Aktivitätsdiagramm.

Auch bei der Synthese des zweiten Beispiels liefern beide Algorithmen das vom Modellierer erwartete Ergebnis. Abbildung 6.3 zeigt die hier entstandene Aktivität.

Beim letzten Beispiel unterscheiden sich die Ergebnisse beider Algorithmen. Sie zeigen beide nicht die Lösung an, welche man erwartet hätte, die aber auch nicht falsch sind. Abbildungen 6.4 und 6.5 zeigen zwei der erhaltenen Lösungen (jeweils zur leichteren Ansicht etwas aufbereitet).

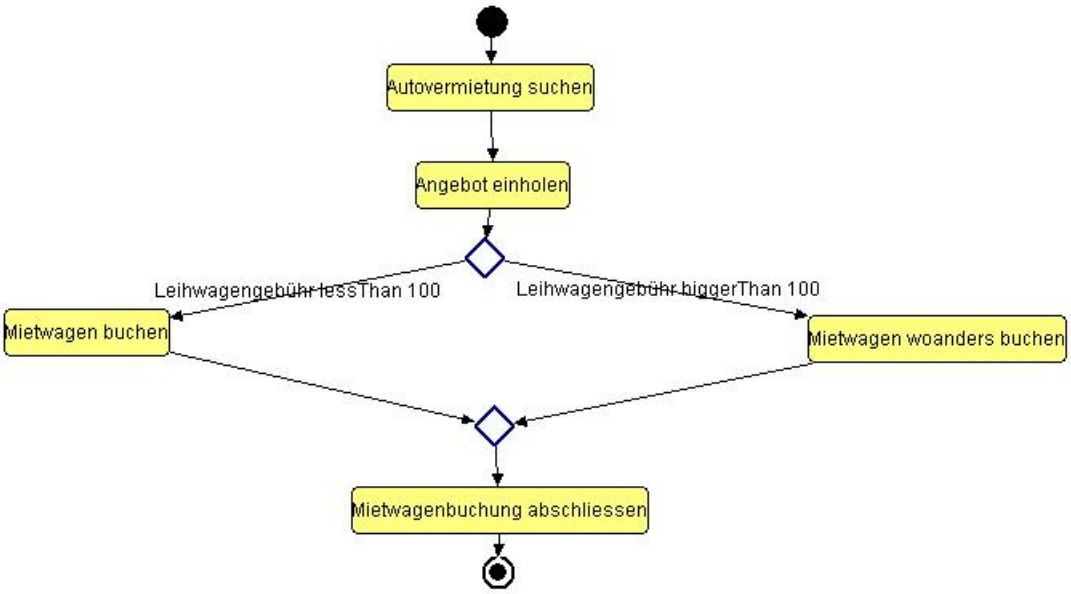


Abbildung 6.2: Das Buchen eines Mietwagens

Ergebnis des modified Prim-Algorithmus

Der beste Lösungsvorschlag, der mit dem Algorithmus modified Prim ermittelt wurde, beginnt mit dem Erstellen eines Kontos und bietet dann die Möglichkeit dieses Konto sofort wieder aufzulösen oder alternativ Geld einzuzahlen. Wurde Geld eingezahlt, gibt es verschiedene Möglichkeiten: sollte der Kontostand immer noch 0 sein (es wurde nicht spezifiziert wieviel Geld eingezahlt wird, der Wert könnte also auch gleich null sein!) kann man das Konto wieder auflösen. Andernfalls kann man das Konto leeren oder Geld überweisen bzw. Geld abheben. Auffällig ist dabei, dass nach dem Geld überweisen nur der Prozess Geld abheben möglich ist, man hätte erwarten können, dass es alternativ (wenn kein Geld mehr auf dem Konto ist) das Konto auflösen kann. Dies könnte auf einen Modellierungsfehler zurückzuführen sein. Der Prozess Geld abheben hat im Gegensatz zum Prozess Geld überweisen keine Effekte (es wurde vergessen zu modellieren, dass der Kontostand einen neuen Wert haben soll). Daher ist es immer möglich nach dem Überweisen von Geld auch Geld abzuheben (der Kontostand bleibt ja gleich).
Auch ist hier möglich, nach dem Abheben von Geld oder Leeren des Kontos das Konto einfach bestehen zu lassen. Der Algorithmus erwartet nicht zwangsweise, dass das Konto am Ende aufgelöst werden muss.

Ergebnis des RandomWalk-Algorithmus

Das Ergebnis des Algorithmus RandomWalk sieht auf den ersten Blick wesentlich verwirrender aus. Bei genauerer Betrachtung zeigen sich folgende Abläufe: nach dem Erstellen des Kontos kann entweder Geld eingezahlt, Geld überwiesen oder das Konto aufgelöst werden. Da der Kontostand anfangs (ausser durch Guthaben welches von der Bank gesponsort wurde) meist den Wert Null hat, wird wohl der zweite Fall nur selten eintreffen. Nach dem Einzahlen von Bargeld ist es möglich dieses zu überweisen, abzuheben oder das Konto zu leeren (wie man es auch erwartet hätte). Hat

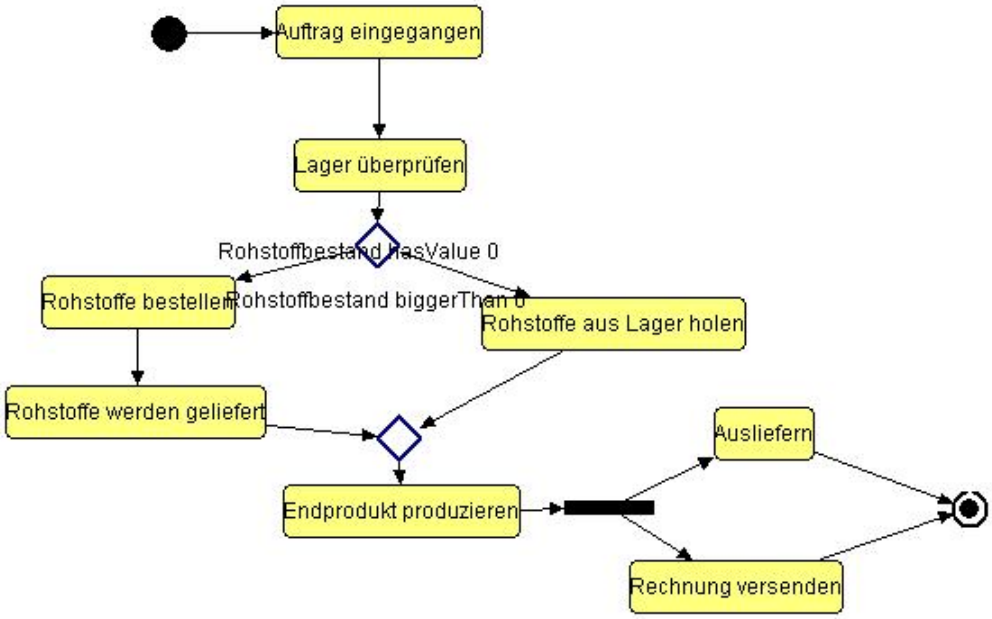


Abbildung 6.3: Die Aktivität der Auftragsbearbeitung

man Geld überwiesen, kann man noch Geld abheben (vermutlich wieder aufgrund des oben genannten Modellierungsfehlers) das Konto leeren oder das Konto auflösen. Wurde Geld abgehoben oder das Konto geleert, bleibt nur noch die Möglichkeit das Konto aufzulösen.
Da der momentane Algorithmus noch keine Schleifen erlaubt, ist dies ein fast korrektes Aktivitätsdiagramm (auch wenn es anfangs etwas verwirrend scheint).

Fazit:

Bei einfachen Beispielen und korrekter sowie ausführlicher Modellierung der Prozesse kommen die Algorithmen auf die vom Modellierer erwarteten Ergebnisse. Bei Fehlern in der Modellierung oder komplexeren Gebilden können nicht vorhergesehene Ergebnisse entstehen, die aber in der Logik selbst Sinn machen. Die Algorithmen versprechen nicht die optimale Lösung zu finden, sondern nur eine möglichst optimale, was sie in allen drei Beispielen erreicht haben.

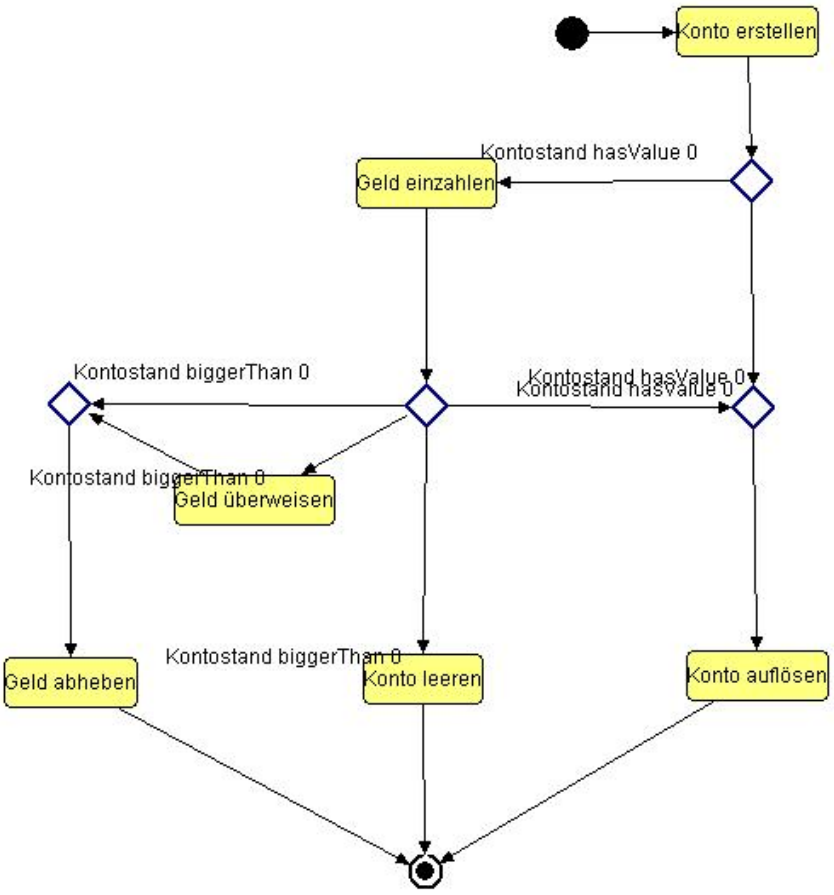


Abbildung 6.4: Aktivitätsdiagramm nach der Synthese mittels modified Prim

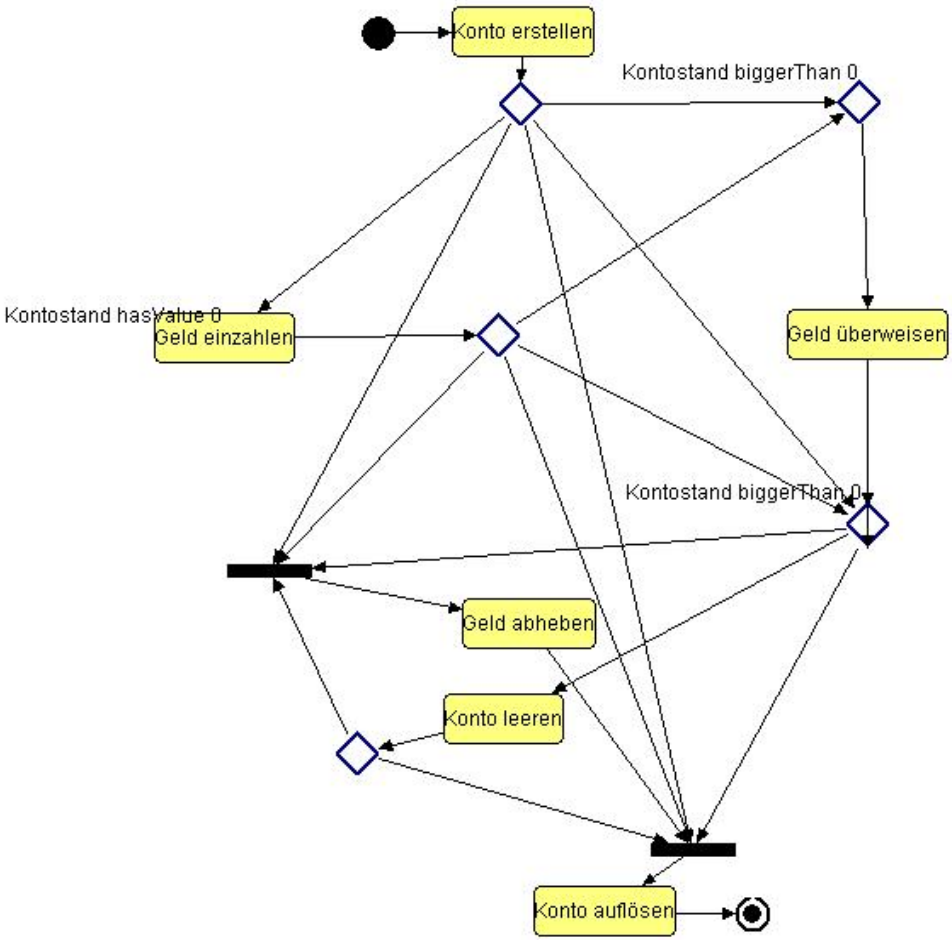


Abbildung 6.5: Nach der Synthese des Kontovorganges mittels RandomWalk

Kapitel 7

Zusammenfassung und Ausblick

7.1 Zusammenfassung

Der hier vorliegende Entwurf zur Modellierung von Geschäftsprozessen ist ein erster Ansatz, um die Vorteile des Semantic Web mit dem seit langen bestehenden Forschungsfeld der Prozessmodellierung zusammenzubringen. Durch die Beschreibungen eines Prozesses, die nicht wie bisher nur in natürlicher Sprache oder der OCL vorliegen, sondern erstmals in einer Form wie sie auch von Maschinen interpretiert werden kann, kann der Computer den Menschen bei der Erstellung von Diagrammen unterstützen, was vor allem für komplexe Geschäftsprozesse das menschliche Eingreifen um einiges verringert. Damit kann bei einer Veränderung des Gesamtprozesses (z.B. ein Teilprozess fällt weg) eine neue optimale Zusammenstellung der Prozesse berechnet werden und das Diagramm (und damit die realen Abläufe im Unternehmen) sehr schnell an die veränderte Situation angepasst werden. Die semantische Modellierung basiert dabei auf den wichtigsten Abschnitten der momentan diskutierten Semantic Web Services-Standards (OWL-S, WSMO, SWSF und METEOR-S). Das entwickelte Tool basiert derzeit auf der Semantic Web-Sprache OWL, kann aber jederzeit durch geringe Änderungen an einen neuen (erweiterten oder alternativen) Standard angepasst werden. Das Tool wurde als Eclipse-Plugin entwickelt und bietet damit die Möglichkeit leicht mit anderen PlugIns verbunden zu werden, um beispielweise von den modellierten Geschäftsprozessen automatisch zu den darunter liegenden (Semantic) Web-Services zu gelangen. Eclipse ist für die meisten Software-Entwickler die beliebteste Entwicklungsumgebung und wird vermutlich in Zukunft auch ausserhalb der Informatik-Branche für verschiedenste Zwecke verwendet werden (z.B. wie in dieser Arbeit zur Modellierung von Geschäftsprozessen). Durch das modulare Konzept innerhalb des PlugIns können unterschiedliche Algorithmen zur Synthese von Geschäftsprozessen getestet werden und auch sehr einfach neue Algorithmen hinzugefügt werden. Dabei ist das Plugin direkt in die Entwicklungsumgebung eingebunden und kann über ein Menü gesteuert werden (Undo, Redo ebenso möglich wie die normale Arbeit im Workspace oder der Aufruf der Synthese aus der Toolbar von Eclipse).

7.2 Ausblick und mögliche Erweiterungen

Das hier entwickelte Tool ist im Moment stärker an der Modellierung von Prozessen im IT-Bereich orientiert. Eine mehr betriebs-orientierte Sichtweise wäre sicher vorstellbar (z.B. bei einem Flug mit Lufthansa erhält man 20% Rabatt). Um solche Regeln der Geschäftslogik (Business Rules)

zu erstellen, wäre eine Erweiterung auf die Logic-Schicht des Semantic Web notwendig, in welcher man mit Standards wie SWRL, RuleML oder WRL arbeiten kann. Hierfür müsste ein neuer Reasoner verwendet werden (z.B. Pellet auf Basis von Jena), was durch die modul-basierte Struktur des Plugins relativ leicht zu verändern wäre. Um solche Instanzen wie Lufthansa modellieren zu können, ist es ausserdem notwendig von der bisher verwendeten Meta-Modell-Ebene (Fluganbieter, Flug) auf die Modellebene zu wechseln (Lufthansa, Flug LH 379).
Der momentane Stand des Plugins ermöglicht noch keine optionale Verwendung von Prozessen oder IOPEs (der Prozess A würde einen Fluganbieter als Input wünschen, sollte keiner übergeben werden, kann sich der Prozess über den Aufruf anderer Prozesse auch selbst einen suchen). Eine dynamische Änderung von Prozessen erfordert momentan auch einen erneuten Modellierungsaufwand, es wäre vorstellbar die Prozesse im Plugin direkt mit darunterliegenden Web-Services zu verknüpfen und damit automatisch auf Veränderungen (statt Flug LH 379 jetzt Flug DE 3276) zu reagieren. Im vorliegenden Stand werden in der Synthese keinerlei Schleifen unterstützt, diese könnten nachträglich hinzugefügt werden (durch einen neuen Algorithmus oder die Anpassung von bestehenden Algorithmen). Eine bessere Verknüpfung der Diagramme (Prozess A zeigt auf Diagramm B und übernimmt automatisch die IOPEs des kompletten Aktivitätsdiagrammes B für den Prozess A) wäre ebenso denkbar. Auch eine Erweiterung der semantischen Informationen eines Prozesses ist vorstellbar: so können in bestimmten Situationen Invarianten notwendig werden (z.B. während eine Überweisung getätigt wird muss dauernd das Konto bestehen und darf nicht aufgelöst werden). Für die momentan bestehenden semantischen Informationen könnte man auch einen genaueren Detaillierungsgrad erreichen, in dem beispielsweise zwischen erwünschten Effekten und unerwünschten Nebenwirkungen (Endprodukt produziert, aber auch Abgase entstanden) unterscheidet.

Der hier vorliegende Ansatz diskutiert die verschiedenen Möglichkeiten, welche in zukünftigen Projekten verwendet und erweitert werden können. Dabei wurde in dieser Arbeit eine semantische Anreicherung von Prozessen in einem UML2-Aktivitätsdiagramm untersucht. Eine Anreicherung wäre selbstverständlich auch in anderen Diagrammtypen der UML (Interaktions-Übersichts-Diagramm o.a.) oder auch in anderen Standards (z.B. ARIS) denkbar.

Literaturverzeichnis

[Agg04] R. Aggarwal. *Semantic Web Services Languages and Technologies: Comparison and Discussion*, 2004. http://citeseer.ist.psu.edu/700682.html.

[Bau04] B. Bauer. *UML 2.0 - An Update*, April 2004. Tutorial at LogOn Briefing OMG.

[BBB+05] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuiness, J. Su und S. Tabet. *Semantic Web Services Framework (SWSF) Overview*, 2005. http://www.daml.org/services/swsf/1.1/overview/.

[BLHL01] T. Berners-Lee, J. Hendler und O. Lassila. *The Semantic Web*. Scientific American, 2001.

[BM05] D. Brickley und L. Miller. *FOAF Vocabulary Specification*, June 2005. http://xmlns.com/foaf/0.1/.

[BRSS98] M. Balser, W. Reif, G. Schellhorn und K. Stenzel. *KIV 3.0 for Provably Correct Systems*, 1998. in Current Trends in Applied Formal Methods.

[CMSP04] J. Cardoso, J. Miller, J. Su und J. Pollock. *Academic and Industrial Research: Do Their Approaches Differ in Adding Semantics to Web Services?* In: *Semantic Web Services and Web Process Composition (SWSWPC04)*, Seiten 14–21, 2004.

[dB05] J. de Bruijn. *Web Rule Language (WRL)*, June 2005. http://www.wsmo.org/wsml/wrl/.

[dBLPF05] J. de Bruijn, H. Lausen, A. Polleres und D. Fensel. *the Web Service Modeling Language WSML: An Overview*, June 2005. http://www.wsmo.org/wsml/wsml-resources/deri-tr-2005-06-16.pdf.

[DIS05] DSTC, IBM und Sandpiper Software. *Ontology Definition Metamodel*, 2005. http://www.itee.uq.edu.au/ colomb/Papers/ad-05-01-01.pdf.

[DS05] M. Duerst und M. Suignard. *Internationalized Resource Identifiers (IRIs)*, January 2005. RFC3987 on http://www.ietf.org/rfc/rfc3987.txt.

[Dum02] E. Dumbill. *XML Watch: Finding friends with XML and RDF*, June 2002. http://www-106.ibm.com/developerworks/xml/library/x-foaf.html.

[Ecl05] Eclipse.org. *Eclipse technical articles*, 2005. http://www.eclipse.org/articles.

[FB02] D. Fensel und C. Bussler. *The Web Service Modeling Framework WSMF*, 2002. http://deri.semanticweb.org.

[FFJ03] R. Fikes, G. Frank und J. Jenkins. *JTP: A System Architecture and Component Library for Hybrid Reasoning*, 2003. http://ksl.stanford.edu/pub/KSL_Reports/KSL-03-01.pdf.

[FFJ04] R. Fikes, G. Frank und J. Jenkins. *JTP: An Object Oriented Modular Reasoning System*, 2004. http://www.ksl.stanford.edu/software/JTP/.

[FL04] P. Fettke und P. Loos. *Referenzmodellierungsforschung - Langfassung eines Aufsatzes.* Wirtschaftsinformatik, 46, 2004. http://www.wirtschaftsinformatik.de/wi_artikel.php?sid=1491.

[GHM05] M. Gruninger, R. Hull und S. McIlraith. *FLOWS: a First-Order-Logic Ontology for Web Services*, June 2005. http://www.w3.org/2005/04/FSWS/Submissions/59/Flows.PDF.

[Gie05] M. Giereth. *Partial RDF Encryption as a Method for Addressee-Oriented Publishing in the Semantic Web.* In: *2nd European Semantic Web Conference, ESWC 2005*, June 2005. http://www.iis.uni-stuttgart.de/personen/giereth/eswc05_encryption.pdf.

[Gru93] T. Gruber. *A translation approach to portable ontology specifications*, 1993.

[HBB+05] D. Haneberg, S. Bäumler, M. Balser, H. Grandy, F. Ortmeier und W. Reif. *the User Interface of the KIV Verification System - A System description*, 2005. Universität Augsburg.

[HE04] L. Hart und P. Emery. *A Description Logic for Use as the ODM Core*, 2004. http://www.sandsoft.com/edoc2004/HartEmeryDLCoreMDSW.pdf.

[HST00] I. Horrocks, U. Sattler und S. Tobies. *Practical reasoning for very expressive description logics*, 2000. http://www3.oup.co.uk/igpl/Volume_08/Issue_03/pdf/horrocks1.pdf.

[Inf05] Stanford Medical Informatics. *The Protégé Ontology Editor and Knowledge Acquisition System*, 2005. http://protege.stanford.edu/.

[Jec05] M. Jeckle. *UML 2.0 - Tools im Überblick*, 2005. http://www.jeckle.de/umltools.htm.

[JRH+04] M. Jeckle, C. Rupp, J. Hahn, B. Zengler und S. Queins. *UML2 glasklar.* Hanser Verlag München, 2004.

[JUG03] Java User Group JUG. *Web-Services*, 2003. http://www.java-user-group.de/protokolle/webservices.pdf.

[LGM+02] R. Letz, C. Gresser, M. Moser, A. Wolf und O. Ibens. *the Theorem Prover SETHEO*, 2002. http://www4.informatik.tu-muenchen.de/ letz/setheo/.

[Lin05] N. Lin. *Semantic Search Engine*, 2005. http://www.cs.umd.edu/ nwlin/828w/.

[LRPF04] R. Lara, D. Roman, A. Polleres und D. Fensel. *A Conceptual Comparison of WSMO and OWL-S.* In: *European Conference on Web Services (ECOWS04)*, Seiten 254–269, 2004.

[LSD05] Large Scale Distributed Information Systems Lab LSDIS. *METEOR-S: Semantic Web Services and Processes*, 2005. http://swp.semanticweb.org/.

[Mur04] D. Murtagh. *Automated Web Service Composition.* Diplomarbeit, University of Dublin, 2004.

[NM01] F. Noy und L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*, 2001. http://www.ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

[Nüt05] M. Nüttgens. *Ontologie hilft bei der Sprachverwirrung.* Computer-Zeitung, 11, March 2005.

[OMG03a] Object Management Group OMG. *Unified Modeling Language (UML) Specification: Infrastructure*, December 2003. version 2.0.

[OMG03b] Object Management Group OMG. *XML Metadata Interchange (XMI) Specification*, May 2003. http://www.omg.org/cgi-bin/doc?formal/2003-05-02.

[OMG05] Object Management Group OMG. *Unified Modeling Language*, 2005. http://www.omg.org/uml.

[OWS+04] B. Oestereich, C. Weiss, C. Schröder, T. Weilkiens und A. Lenhard. *Objektorientierte Geschäftsprozessmodellierung mit der UML.* dpunkt.verlag, 2004.

[Pla94] D.A. Plaisted. *The Search Efficiency of Theorem Proving Strategies: An Analytical Comparison*, 1994. http://citeseer.ist.psu.edu/plaisted94search.html.

[PS04] B. Parsia und E. Sirin. *Pellet: An OWL DL Reasoner*, 2004. http://iswc2004.semanticweb.org/posters/PID-ZWSCSLQK-1090286232.pdf.

[RLK04] D. Roman, H. Lausen und U. Keller. *D2v03. Web Service Modeling Ontology - Standard*, 2004. http://www.wsmo.org/2004/d2/v0.3/20040329/.

[sB05] H. Österle und D. Blessing. *Ansätze des Business Engineering*, Band 241 der Reihe *HMD - Praxis der Wirtschaftsinformatik*, Seiten 7–17. dpunkt.Verlag, 2005.

[Sch98] A.-W. Scheer. *ARIS - Vom Geschäftsprozess zum Anwendungssystem.* Springer Verlag, 1998.

[Sch00] S. Schulz. *the E Equitational Theorem Prover*, 2000. http://www4.informatik.tu-muenchen.de/s̃chulz/WORK/eprover.html.

[Sei02] H. Seidlmeier. *Prozessmodellierung mit ARIS.* vieweg Verlag, 2002.

[SIC05] Semantic Interoperability Community of Practice SICoP. *Introducing Semantic Technologies and the Vision of the Semantic Web*, February 2005.

[SP04] E. Sirin und B. Parsia. *Planning for Semantic Web Services*, 2004. http://www.ai.sri.com/SWS2004/final-versions/SWS2004-Sirin-Final.pdf.

[SPKR96] W. Swartout, R. Patil, K. Knight und T. Russ. *Towards distributed use of large-scale ontologies.* In: *Proceedings of the 10th Knowledge Acquisition for Knowledge-based Systems Workshop*, 1996.

[SS98] G. Sutcliffe und C.B. Suttner. *The TPTP Problem Library: CNF Release v1.2.1.* Journal of Automated Reasoning, 21(2):177–203, 1998. http://www.cs.miami.edu/t̃ptp.

[Sun03] Sun. *Java Web Services Tutorial*, 2003. http://java.sun.com/webservices/docs/1.0/ /tutorial/doc/glossary.html.

[SWS05] SWSI. *Semantic Web Services Initiative*, 2005. http://www.swsi.org/.

[Tec05] BBN Technologies. *SWeDE*, 2005. http://owl-eclipse.projects.semwebcentral.org/.

[TH01] D. Tsarkov und I. Horrocks. *Reasoner Demonstrator*, 2001. http://wonderweb.semanticweb.org/deliverables/documents/D14.pdf.

[UG96] M. Uschold und M. Grüninger. *Ontologies: Principles, methods and applications*, 1996.

[Uni05] Columbia University. *Small World Project*, 2005. http://smallworld.columbia.edu/description.html.

[VSC04] K. Verma, A. Sheth und F. Curbera. *Service Oriented Architectures and Semantic Web Processes*, 2004. http://lsdis.cs.uga.edu/lib/presentations/ICSOC-tutorial-final.ppt.

[W3C04a] World Wide Web Consortium W3C. *OWL-S: Semantic Markup for Web Services*, 2004. http://www.w3.org/Submission/OWL-S/.

[W3C04b] World Wide Web Consortium W3C. *OWL Web Ontology Language Guide*, 2004. http://www.w3.org/TR/owl-guide/.

[W3C04c] World Wide Web Consortium W3C. *RDF - Resource Description Framework*, 2004. http://www.w3.org/rdf.

[W3C04d] World Wide Web Consortium W3C. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[Wik05] Wikipedia.org. *Web Ontology Language (OWL)*, 2005. http://en.wikipedia.org/wiki/Web_Ontology_Language.

[WO04] T. Weilkiens und B. Oestereich. *UML2 Zertifizierung Fundamental*. dpunkt.verlag, 2004.

Abbildungsverzeichnis

Anhang A

Abkürzungsverzeichnis

Abox assertorisches Wissen

API Application Program Interface

ASM Abstract State Machine

BPEL4WS Business Process Execution Language for Web Services *(mittlerweile WS-BPEL)*

CWM Common Warehouse Metamodel

DAML Darpa Agent Markup Language

DFG Deutsche Forschungsgemeinschaft

DL Description Logic, Beschreibungslogik

DTD Document Type Definition

DV-Konzept DatenVerarbeitungs-Konzept

eEPK erweiterte Ereignisgesteuerte Prozess-Ketten

EMF Eclipse Modeling Framework

ER-Diagramm Entity-Relationship-Diagramm

FLOWS First-Order Logic Ontology for Web Services

FOAF Friend-Of-A-Friend

FOL First-Order Logic, Prädikatenlogik erster Ordnung

GEF eclipse Graphical Editing Framework

GPS General Problem Solver

HOL Higher-Order Logic, Prädikatenlogik zweiter Stufe

HTML HyperText Markup Language

HTTP HyperText Transfer Protocol

IOPE Inputs, Outputs, Preconditions and Effects

IOPEsame Sowohl die Inputs und Outputs als auch Preconditions und Effects sind gleich (*are the same*)

IOPEsim Inputs und Outputs sowie Preconditions und Effects sind ähnlich (*are similar*)

IOsame Inputs und Outputs sind gleich (*are the same*)

IOsim Inputs und Outputs sind ähnlich (*are similar*)

IRI Internationalized Resource Identifier

ISO International Organisation for Standardisation

JDT eclipse Java Development Tooling

JESS Java Expert System Shell

JTP Java Theorem Prover

KIF Knowledge Interchange Format

KIV Karlsruhe Interactive Verifier

LP Logic-Programming

METEOR-S Managing End-To-End OpeRations for Semantic web services

MOF Meta Object Facility

MVC Model-View-Controller pattern

MWSAF METEOR-S Web Service Annotation Framework

MWSCF METEOR-S Web Service Composition Framework

MWSDI METEOR-S Web Service Discovery Infrastructure

MWSDPM METEOR-S Web Service Dynamic Process Manager

OCL Object Constraint Language

ODM Ontology Definition Metamodel

ODT Ontology Definition Toolkit *(von IBM)*

OIL Ontology Interface Layer *oder* Ontology Interchange Language

OMG Object Management Group

OMT Object Modeling Technique

OOD Object-Oriented Design

OOSE Object-Oriented Software Engineering

ORL Ontology Rules Language

OWL Web Ontology Language

OWL-S Web Ontology Language for Services

PDE eclipse Plug-In Developer Environment

PEsame Preconditions und Effect sind gleich (*are the same*)

PEsim Preconditions und Effects sind ähnlich (*are similar*)

PL1 Prädikatenlogik erster Stufe

PSL Process Specification Language

QoS Quality of Service

RDF Resource Description Framework

RDFS RDF Schema

ROWS Rules Ontology for Web Services

RPC Remote Procedure Call

RuleML Rules Markup Language

SCOR-Modell Supply-Chain-Operation-Reference Modell

SemBPM Semantic-based Business Process Modeling

SETHEO SEquential THEOrem prover

SOA Service-Oriented Architecture

SOAP Simple Object Access Protocol

SWRL Semantic Web Rules Language

SWSF Semantic Web Services Framework

SWSI Semantic Web Services Initiative

SWSL Semantic Web Services Language

SWSL-FOL Semantic Web Services Language in First-Order Logic

SWSO Semantic Web Services Ontology

SWT Standard Widget Toolkit

Tbox terminologisches Wissen

UDDI Universal Description, Discovery and Integration

UML *siehe UML2*

UML2 Unified Modeling Language in Version 2.0

URI Uniform Resource Identifier

URL Uniform Resource Locator

WRL WSMO Rules Language

WS-BPEL Web Services Business Process Execution Language *(Nachfolger von BPEL4WS)*

WSDL Web Service Description Language

WSDL-S Web Service Description Language with Semantics *(Teil des METEOR-S Projekts)*

WSMF Web Service Modeling Framework

WSML Web Service Modeling Language

WSMO Web Service Modeling Ontology

WSMX Web Service Modeling eXecution environment

XMI XML Metadata Interchange

XML eXtensible Markup Language

Anhang B

Installation und Bedienung

Installation

Zur Installation des Plugins benötigt man das Open-Source-Framework Eclipse. Dies kann von http://www.eclipse.org heruntergeladen werden. Zum Installieren des im Rahmen dieser Arbeit erstellten Plugins wählt man in Eclipse unter Help - Software Updates - Find and Install. Im aufgehenden Fenster selektiert man „Search for new features to install“ und wählt eine neue Remote Site aus. Den Namen dieser Remote Site kann man frei wählen (zum Beispiel Universität Augsburg), die URL, auf welcher sich das Plugin im Eclipse Repository befindet, lautet:

http://pvs.informatik.uni-augsburg.de/eclipse.

Ist die neu angelegte Site (Universität Augsburg) ausgewählt, kann dort automatisch nach dem Plugin gesucht werden. Während der Installation wird Eclipse zwischenzeitlich neu gestartet und nach Beendigung steht das Eclipse-Plugin zur Arbeit zur Verfügung.

Eventuell stehen bei der Installation notwendige Plugins nicht zur Verfügung. Benötigt werden das GEF SDK (aktuell Version 3.0.1), welches unter der Eclipse Update-Site zu finden ist, sowie org.daml.jena, welches unter folgendem Link gefunden werden kann:

http://owl-eclipse.projects.semwebcentral.org/Jena/.

Die Installation dieser Plugins läuft nach dem selben Prinzip wie die Installation des eigenen Plugins (wie oben beschrieben: neue Remote Site erstellen, etc.). Wurde die Installation abgeschlossen, kann es notwendig sein, den Pfad zur Schema-Datei SemBPM.owl zu aktualisieren, damit alle Prädikate zur Beschreibung von Preconditions und Effects gefunden werden können oder eine Synthese angestossen werden kann. Die Datei SemBPM.owl findet sich in einem Unterverzeichnis von Eclipse unter

[eclipse]/plugins/de.uniAugsburg.semBPM_1.0.0/semBPM.owl.

Diese Grundeinstellung kann unter „Window“ -> „Preferences“ im „synthesis“-Bereich von „Semantic-based Business Process Modeling“ vorgenommen werden.

Bedienung

Um mit dem neu installierten Plugin eine Modellierung von Geschäftsprozessen vorzunehmen, muss zuerst ein neues Projekt eröffnet werden. Dieses Projekt hat den Namen SemBPM-Projekt und

ist im Bereich semantic-based Business Process Modeling zu finden (siehe Screenshot B.1). Dort wählt man den Namen des Projekts, den Namen der zu verwendenden Ontologie und kann hier eine bereits vorhandene Ontologie auswählen, die automatisch importiert und in dem Projekt unter dem angegebenen Namen abgespeichert wird. Ausserdem muss hier der Name des anzulegenden Diagramms ausgewählt werden. Der Wizard erstellt zum Abschluss die geforderten Dateien in dem neuen Projekt. In diesem Projekt können jederzeit weitere Diagramme (New -> Other -> SemBPM Activity diagram) erstellt werden, die sich dann ebenfalls auf die angegebene Ontologie beziehen.

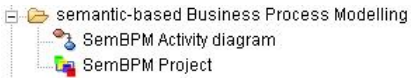


Abbildung B.1: Erstellung eines neuen Projektes

Beim Bearbeiten der Ontologie ist das Plugin SWeDE [Tec05] sehr hilfreich, da es automatisch ein Syntax-Highlighting vornimmt und eine Unterstützung bei der Eingabe bietet. Das Erstellen des Aktivitätsdiagramms wird durch die Palette ermöglicht (siehe B.2), die man auch automatisch minimieren kann, sollte sie einen bei der Modellierung behindern. Hier kann man Aktionen, Start- und Endknoten sowie Parallelisierungs- und Entscheidungsknoten auswählen und durch einen Klick auf die Diagrammfläche hinzufügen. Verbindungen können durch das Connection-Tool „ActivityEdge“ ebenfalls erstellt werden, in dem zwei vorhandene Knoten nacheinander ausgewählt werden.

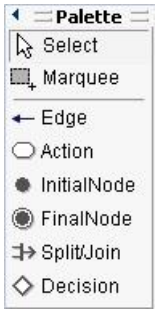


Abbildung B.2: Die Palette des PlugIns

Wurde eine Aktion erstellt, kann mit der semantischen Modellierung dieser Aktion begonnen werden. Dazu wählt man im Kontext-Menü der Aktion den Punkt „Open Properties“. Im neuen Fenster wählt man aus, welche Art von Property man hinzufügen möchte (Input, Output, Precondition oder Effect) und gelangt beim Auswählen von „Add“ in das in Abbildung B.3 zu sehende Fenster, wo man die Property auswählen kann. Für Inputs und Outputs interessieren nur die Subjects (Konzepte der Ontologie), welche automatisch aus der angegebenen Ontologie geladen werden (hat sich einmal die Ontologie verändert, kann im Kontextmenü der Punkt „Reload Ontology“ ausgewählt werden, damit wieder mit dem neuesten Stand gearbeitet werden kann). Für Preconditions und Effects wählt man zuerst die Variable aus, welche für diesen Prozess interessant ist und weist dieser dann einen Wert zu (setValue) oder nimmt einen Vergleich auf Basis dieser Variablen vor (hasValue, biggerThan, lowerThan).

Sind alle Prozesse erstellt, kann man die Ausführungssemantik festlegen. Im Kontext-Menü unter „Open Execution Semantic“ (Abbildung B.4) sucht man zwei beliebige Prozesse heraus und

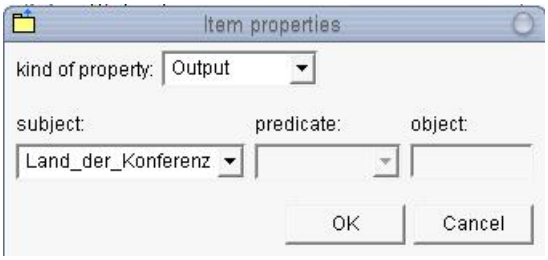


Abbildung B.3: Die Einstellung der IOPEs

legt für diese Prozesse fest, wie sie zueinander in Verbindung stehen. Dabei ist möglich auszuwählen, dass diese parallel ablaufen sollen oder dass ein Prozess vor dem anderen kommen soll (oder irgendwann danach).

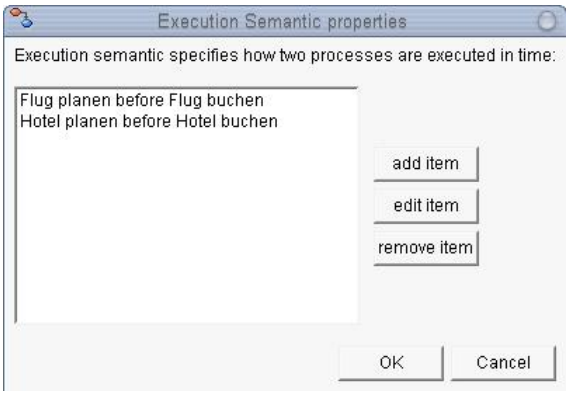


Abbildung B.4: Ein Screenshot der Ausführungssemantik

Sowohl für das Modellieren der Aktivität als auch für die Durchführung der Synthese können diverse Voreinstellungen vorgenommen werden: für Aktionen kann die Farbe festgelegt werden mit welcher diese gezeichnet werden (standardmässig lemon-gelb). Man hat ausserdem die Möglichkeit die Rahmenfarbe der Entscheidungsknoten (Standard: blau) zu ändern und kann auch eine Änderung der Datei-Endung von semantisch angereicherten Aktivitätsdiagrammen vornehmen (Standard: SUAD als Abkürzung für *Semantic-based Uml Activity Diagram*).
Für die Synthese kann der Pfad der zugrundeliegenden Standard-Ontologie (für alle Projekte) zugewiesen werden. Der Pfad zur projekt-spezifischen Ontologie, die auf dieser Standard-Ontologie aufbaut, wird in den Settings eines jeden Projektes gespeichert und kann nur dort verändert werden. Auch hier gibt es wieder die Möglichkeit die Datei-Endung zu verändern, dieses Mal die Endung der Ontologie-Dateien (standardgemäss: OWL, könnte beispielsweise später auf SWRL umgeändert werden). Desweiteren hat man die Möglichkeit den Algorithmus auszuwählen, der für die Synthese verwendet wird. Hier sind anfangs die im Kapitel 4.4 beschriebenen Algorithmen modified Prim und RandomWalk vorhanden. Führte ein Algorithmus zu einer oder mehreren Lösungen kann festgelegt werden, wieviele Lösungen angezeigt werden sollen und wie der Dateiname dieser Lösungen sein soll (Standard: Solution). Dieser Dateinamen wird noch mit einer Nummer versehen je nach Anzahl der Lösungen, also z.B. Solution01.suad. Die Abbildungen B.5 und B.6 zeigen Screenshots der eben genannten Preferences.

Wurden alle Einstellungen vorgenommen und alle Aktionen erstellt, kann die Synthese gestartet

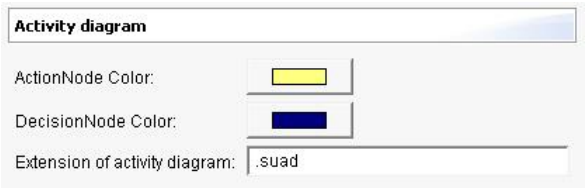


Abbildung B.5: Die Preferences eines Activity-Diagramms

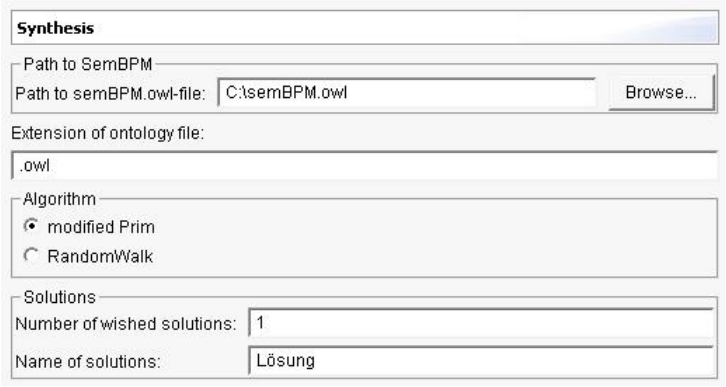


Abbildung B.6: Die Preferences der Synthese

werden. Dies geschieht entweder durch den Menüpunkt „SemBPM -> start synthesis“ oder über ein Symbol aus der Toolbar (Tooltip ebenfalls „start synthesis“). Wurden ein oder mehrere Ergebnisse gefunden, erzeugt die Synthese automatisch verschiedene Lösungsdateien und öffnet diese. Wurde keine zufriedenstellende Lösung gefunden, kann eine Verfeinerung der semantischen Modellierung vorgenommen und eine neuerliche Synthese gestartet werden.

Anhang C

Die Datei SemBPM.owl

Die folgende Datei mit dem Namen SemBPM.owl ist die Basis für die Erstellung von Ontologien zur ontologie-basierten Modellierung von Geschäftsprozessen. SemBPM steht dabei für Semantic-based Business Process Modeling. Im ersten Teil werden die Konstrukte für die Datensemantik definiert, im darauf folgenden Abschnitt die Konstrukte der Ausführungslogik.

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE uridef[
   <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns'>
   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl'>
   <!ENTITY swrl 'http://www.w3.org/2003/11/swrl'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema'>
   <!ENTITY owls 'http://www.daml.org/services/owl-s/1.1/Process.owl'>
   <!ENTITY semBPM 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
   <!ENTITY DEFAULT 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
]>

<rdf:RDF
   xmlns:rdf = '&rdf;#'
   xmlns:rdfs = '&rdfs;#'
   xmlns:swrl = '&swrl;#'
   xmlns:owl = '&owl;#'
   xmlns:xsd = '&xsd;#'
   xmlns:owls = '&owls;#'
   xmlns:semBPM = '&semBPM;#'
   xmlns = '&DEFAULT;#'
   xml:base = '&semBPM;#'>

<!-
   ######################################################
         semBPM Version 1.0 Ontology Structure
   ######################################################

semBPM is the semantic-based business process modeling-
standard for enabling a process synthesis of semantically
```

```
annotated business processes in e.g. an UML2 - interaction
overview diagram or activity diagram.
This is part of an Eclipse PlugIn that assists the semantic
enabled business process modeling work.
This PlugIn is called semBPM as well.
For further information ask:
University of Augsburg / Germany
Bernhard Bauer, Florian Lautenbacher

2005-07-15 ->

<owl:Ontology rdf:about=' '>
   <owl:versionInfo>
      $Id: semBPM.owl,v 1.0 2005/07/15 16:00:00 Florian Lautenbacher$
   </owl:versionInfo>
   <rdfs:comment>
      semBPM is the semantic-based business process modeling standard
   </rdfs:comment>
</owl:Ontology>

<!- Data Semantics ->

<owl:Class rdf:ID='Object'>
   <rdfs:comment>
      An Object specifies every important part of a data semantic.
   </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID='SemDataProperty'>
   <rdfs:subClassOf rdf:resource='&owl;#DatatypeProperty'/>
   <rdfs:comment>
      A datatype property for the data and execution semantic.
   </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID='SemObjectProperty'>
   <rdfs:subClassOf rdf:resource='&owl;#ObjectProperty'/>
   <rdfs:comment>
      An object property for the data and execution semantic.
   </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID='SemExecutionProperty'>
   <rdfs:subClassOf rdf:resource='&owl;#ObjectProperty'/>
   <rdfs:comment>
      An object property for the functional semantic.
   </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID='Variable'>
   <rdfs:comment>
      A variable is abstract and enables working with local and global
      variables. Local variables are not used by now.
```

```
   </rdfs:comment>
</owl:Class>

<owl:Class rdf:ID='globalVariable'>
   <rdfs:subClassOf rdf:resource='#Variable'/>
   <rdfs:comment>
      A globalVariable will be used in more than one process and needs to
      be declared all the time. This can be a global state of e.g. a database
      or a bank account.
   </rdfs:comment>
</owl:Class>

<semBPM:SemDataProperty rdf:ID='hasValue'>
   <rdf:type rdf:resource='#SemDataProperty'/>
   <rdfs:domain rdf:resource='#Variable'/>
   <rdfs:range rdf:resource='&xsd;#boolean'/>
   <rdfs:comment>
      Returns whether there is a value of a variable. Boolean.
   </rdfs:comment>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='setValue'>
   <rdf:type rdf:resource='#SemDataProperty'/>
   <rdfs:domain rdf:resource='#Variable'/>
   <rdfs:range rdf:resource='&owl;#Thing'/>
   <rdfs:comment>
      Sets a new value for the Variable. This can be every xsd-value.
   </rdfs:comment>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='lessThan'>
   <rdf:type rdf:resource='#SemDataProperty'/>
   <rdfs:domain rdf:resource='#Variable'/>
   <rdfs:range rdf:resource='&xsd;#boolean'/>
   <rdfs:comment>
      Tests whether one variable is less than another.
   </rdfs:comment>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='biggerThan'>
   <rdf:type rdf:resource='#SemDataProperty'/>
   <rdfs:domain rdf:resource='#Variable'/>
   <rdfs:range rdf:resource='&xsd;#boolean'/>
   <rdfs:comment>
      Tests whether one variable is bigger than another.
   </rdfs:comment>
</semBPM:SemDataProperty>

<!- Execution Semantics ->

<owl:Class rdf:ID='Process'>
   <owl:sameAs rdf:resource='&owls;#AtomicProcess'/>
   <rdfs:comment>
```

```
      A process is defined similar than an atomicProcess in OWL-S.
   </rdfs:comment>
</owl:Class>

<semBPM:SemExecutionProperty rdf:ID='next'>
   <rdfs:domain rdf:resource='#Process'/>
   <rdfs:range rdf:resource='#Process'/>
   <owl:inverseOf rdf:resource='#previous'/>
   <rdfs:comment>
      One process followes directly after the other.
   </rdfs:comment>
</semBPM:SemExecutionProperty>

<semBPM:SemExecutionProperty rdf:ID='previous'>
   <rdfs:domain rdf:resource='#Process'/>
   <rdfs:range rdf:resource='#Process'/>
   <owl:inverseOf rdf:resource='#next'/>
   <rdfs:comment>
      This process comes directly before the other one.
   </rdfs:comment>
</semBPM:SemExecutionProperty>

<semBPM:SemExecutionProperty rdf:ID='eventually'>
   <rdfs:domain rdf:resource='#Process'/>
   <rdfs:range rdf:resource='#Process'/>
   <rdf:type rdf:resource='&owl;#TransitiveProperty'/>
   <owl:inverseOf rdf:resource='#before'/>
   <rdfs:comment>
      Sometime later on this process will follow.
   </rdfs:comment>
</semBPM:SemExecutionProperty>

<semBPM:SemExecutionProperty rdf:ID='before'>
   <rdfs:domain rdf:resource='#Process'/>
   <rdfs:range rdf:resource='#Process'/>
   <rdf:type rdf:resource='&owl;#TransitiveProperty'/>
   <owl:inverseOf rdf:resource='#eventually'/>
   <rdfs:comment>
      Sometime before this process must appear.
   </rdfs:comment>
</semBPM:SemExecutionProperty>

<semBPM:SemExecutionProperty rdf:ID='parallel'>
   <rdfs:domain rdf:resource='#Process'/>
   <rdfs:range rdf:resource='#Process'/>
   <rdf:type rdf:resource='&owl;#SymmetricProperty'/>
   <rdfs:comment>
      These two processes are parallel.
   </rdfs:comment>
</semBPM:SemExecutionProperty>

</rdf:RDF>
```

Anhang D

Vollständige Semantik des Beispiels „Buchen einer Geschäftsreise“

Datensemantik

```
<!DOCTYPE uridef[
   <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns'>
   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema'>
   <!ENTITY semBPM 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
   <!ENTITY DEFAULT 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
]>

<rdf:RDF
   xmlns:rdf = '&rdf;#'
   xmlns:rdfs = '&rdfs;#'
   xmlns:owl = '&owl;#'
   xmlns:xsd = '&xsd;#'
   xmlns:semBPM = '&semBPM;#'
   xmlns = '&DEFAULT;'
   xml:base = '&semBPM;'>

<semBPM:Object rdf:ID='Urlaubsland'/>

<semBPM:Object rdf:ID='Datum'/>

<semBPM:Object rdf:ID='Anfangsdatum'>
   <rdfs:subClassOf rdf:resource='#Datum'/>
   <owl:disjointWith rdf:resource='#Enddatum'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Enddatum'>
```

```
   <rdfs:subClassOf rdf:resource='#Datum'/>
   <owl:disjointWith rdf:resource='#Anfangsdatum'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Flughafen'/>

<semBPM:Object rdf:ID='Abflughafen'>
   <rdfs:subClassOf rdf:resource='#Flughafen'/>
   <owl:disjointWith rdf:resource='#Zielflughafen'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Zielflughafen'>
   <rdfs:subClassOf rdf:resource='#Flughafen'/>
   <owl:disjointWith rdf:resource='#Abflughafen'/>
</semBPM:Object>

<semBPM:Object rdf:ID='Autovermietung'/>

<semBPM:Object rdf:ID='Auto'/>

<semBPM:SemDataProperty rdf:ID='vermietet'>
   <rdfs:domain rdf:resource='#Autovermietung'/>
   <rdfs:range rdf:resource='#Auto'/>
</semBPM:SemDataProperty>

<semBPM:Object rdf:ID='Hotelanbieter'/>

<semBPM:Object rdf:ID='Hotel/>

<semBPM:SemDataProperty rdf:ID='liegtIn'>
   <rdfs:domain rdf:resource='#Hotel'/>
   <rdfs:range rdf:resource='#Urlaubsland'/>
</semBPM:SemDataProperty>

<semBPM:SemDataProperty rdf:ID='bietetAn'>
   <rdfs:domain rdf:resource='#Hotelanbieter'/>
   <rdfs:range rdf:resource='#Hotel'/>
</semBPM:SemDataProperty>

<semBPM:globalVariable rdf:ID='Flug_gebucht'/>
<semBPM:globalVariable rdf:ID='Hotel_gebucht'/>
<semBPM:globalVariable rdf:ID='Auto_gebucht'/>
<semBPM:globalVariable rdf:ID='Buchung_OK'/>

</rdf:RDF>
```

funktionale Semantik

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE uridef[
```

```
   <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns'>
   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema'>
   <!ENTITY owl 'http://www.w3.org/2002/07/owl'>
   <!ENTITY xsd 'http://www.w3.org/2001/XMLSchema'>
   <!ENTITY owls 'http://www.daml.org/services/owl-s/1.1/Process.owl'>
   <!ENTITY semBPM 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
   <!ENTITY DEFAULT 'http://pvs.informatik.uni-augsburg.de/eclipse/semBPM.owl'>
   <!ENTITY dataSem 'DataSemantics.owl'>
]>

<rdf:RDF
   xmlns:rdf = '&rdf;#'
   xmlns:rdfs = '&rdfs;#'
   xmlns:owl = '&owl;#'
   xmlns:xsd = '&xsd;#'
   xmlns:owls = '&owls;#'
   xmlns:semBPM = '&semBPM;#'
   xmlns:dataSem = '&dataSem;#'
   xmlns = '&DEFAULT;#'
   xml:base = '&semBPM;'>

<!- Geschäftsreise planen ->
<semBPM:Process rdf:ID='Geschäftsreise_planen'>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Urlaubsland'/>
   </owls:hasOutput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Anfangsdatum'/>
   </owls:hasOutput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Enddatum'/>
   </owls:hasOutput>
</semBPM:Process>

<!- Flug planen ->
<semBPM:Process rdf:ID='Flug_planen'>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Urlaubsland'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Anfangsdatum'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Enddatum'/>
   </owls:hasInput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Abflughafen'/>
   </owls:hasOutput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Zielflughafen'/>
   </owls:hasOutput>
</semBPM:Process>
```

```
<!- Flug buchen ->
<semBPM:Process rdf:ID='Flug_buchen'>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Abflughafen'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Zielflughafen'/>
   </owls:hasInput>
   <owls:hasResult>
      <owls:Result>
         <owls:hasEffect>
            <owls:Effect>
               <semBPM:globalVariable semBPM:variable='&dataSem;#Flug_gebucht'
                  semBPM:setValue='True'/>
            </owls:Effect>
         </owls:hasEffect>
      </owls:Result>
   </owls:hasResult>
</semBPM:Process>

<!- Hotel planen ->
<semBPM:Process rdf:ID='Hotel_planen'/>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Urlaubsland'/>
   </owls:hasInput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Hotelanbieter'/>
   </owls:hasOutput>
   <owls:hasOutput>
      <owls:Output rdf:ID='&dataSem;#Hotel'/>
   </owls:hasOutput>
</semBPM:Process>

<!- Hotel buchen ->
<semBPM:Process rdf:ID='Hotel_buchen'/>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Hotelanbieter'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Hotel'/>
   </owls:hasInput>
   <owls:hasResult>
      <owls:Result>
         <owls:hasEffect>
            <owls:Effect>
               <semBPM:globalVariable semBPM:variable='&dataSem;#Hotel_gebucht'
                  semBPM:setValue='True'/>
            </owls:Effect>
         </owls:hasEffect>
      </owls:Result>
   </owls:hasResult>
</semBPM:Process>
```

```
<!- Mietwagen buchen ->
<semBPM:Process rdf:ID='Flug_buchen'/>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Urlaubsland'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Anfangsdatum'/>
   </owls:hasInput>
   <owls:hasInput>
      <owls:Input rdf:ID='&dataSem;#Enddatum'/>
   </owls:hasInput>
   <owls:hasResult>
      <owls:Result>
         <owls:hasEffect>
            <owls:Effect>
               <semBPM:globalVariable semBPM:variable='&dataSem;#Auto_gebucht'
                  semBPM:setValue='True'/>
            </owls:Effect>
         </owls:hasEffect>
      </owls:Result>
   </owls:hasResult>
</semBPM:Process>

<!- Geschäftsreise durchführen ->
<semBPM:Process rdf:ID='Geschäftsreise_durchführen'/>
   <owls:hasPrecondition>
      <owls:Precondition>
         <semBPM:globalVariable semBPM:variable='&dataSem;#Flug_gebucht'
            semBPM:hasValue='True'/>
      </owls:Precondition>
   </owls:hasPrecondition>
   <owls:hasPrecondition>
      <owls:Precondition>
         <semBPM:globalVariable semBPM:variable='&dataSem;#Hotel_gebucht'
            semBPM:hasValue='True'/>
      </owls:Precondition>
   </owls:hasPrecondition>
   <owls:hasPrecondition>
      <owls:Precondition>
         <semBPM:globalVariable semBPM:variable='&dataSem;#Auto_gebucht'
            semBPM:hasValue='True'/>
      </owls:Precondition>
   </owls:hasPrecondition>
   <owls:hasResult>
      <owls:Result>
         <owls:hasEffect>
            <owls:Effect>
               <semBPM:globalVariable semBPM:variable='&dataSem;#Buchung_OK'
                  semBPM:setValue='True'/>
            </owls:Effect>
         </owls:hasEffect>
      </owls:Result>
   </owls:hasResult>
```

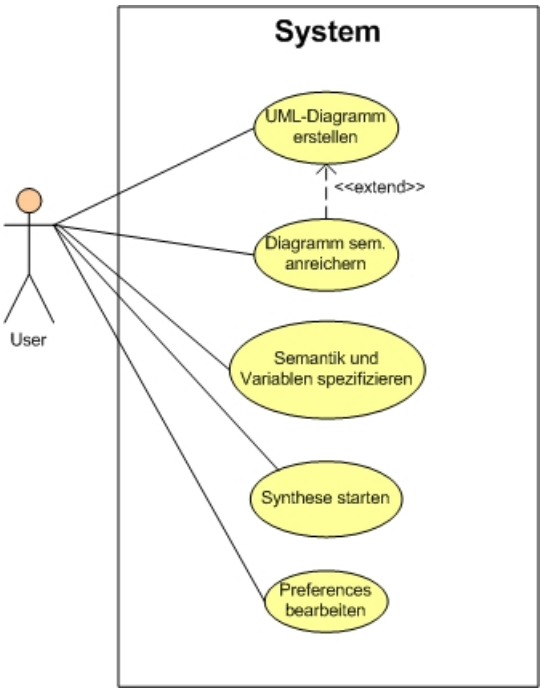
```
</semBPM:Process>
</rdf:RDF>
```

Anhang E

Designdokumente des Plugins

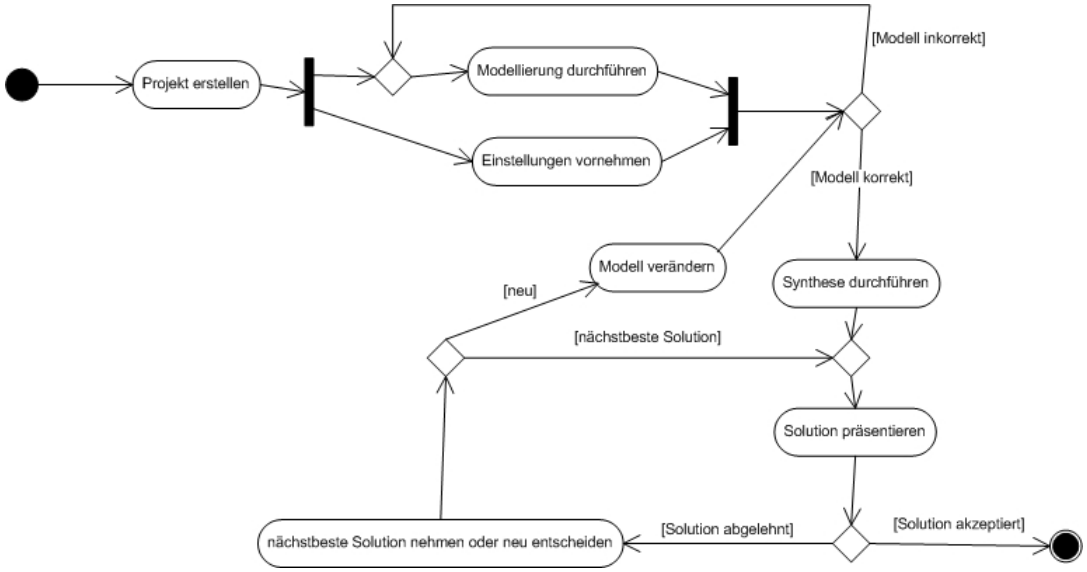
UseCase

Das UseCase-Diagramm beschreibt die Möglichkeiten, die ein Benutzer bei der Arbeit mit dem SemBPM-Plugin hat: er kann ein normales UML2-Aktivitätsdiagramm erstellen und dieses mit semantischen Informationen anreichern. Dazu benötigt er eine weitergehende Semantik (Datensemantik und Ausführungssemantik) und muss ausserdem Variablen definieren, die für die funktionale Semantik benötigt werden. Aufbauend auf vorher erstellten Präferenzen, kann eine Synthese gestartet werden und mit der erstellten Lösung weiter gearbeitet werden.



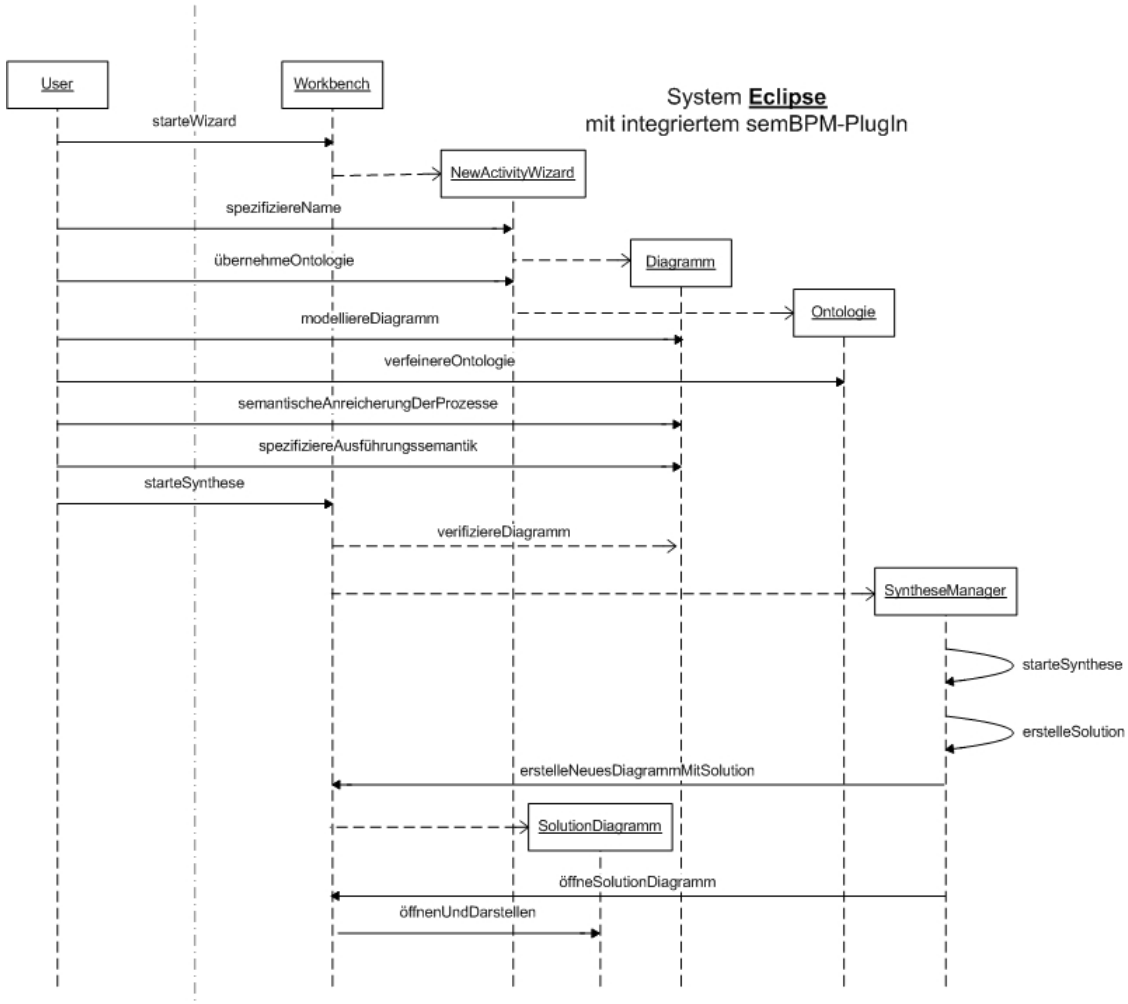
Interaktionsübersichtsdiagramm

Das Interaktionsübersichtsdiagramm liefert einen groben Überblick über die Interaktionen, die ein Nutzer mit dem Plugin vornehmen kann. Der Benutzer kann in einem eigens dafür erstellten Projekt die semantische Modellierung vornehmen und dann eine Synthese durchführen. Diese lässt sich nur starten, wenn das vorliegende Modell auch korrekt im Sinne von UML2-Aktivitätsdiagrammen ist. Wurde die Synthese durchgeführt, wird eine Lösung präsentiert, die der Nutzer entweder für die weitere Arbeit verwenden kann oder er verändert sein vorliegendes Modell und startet erneut eine Synthese.



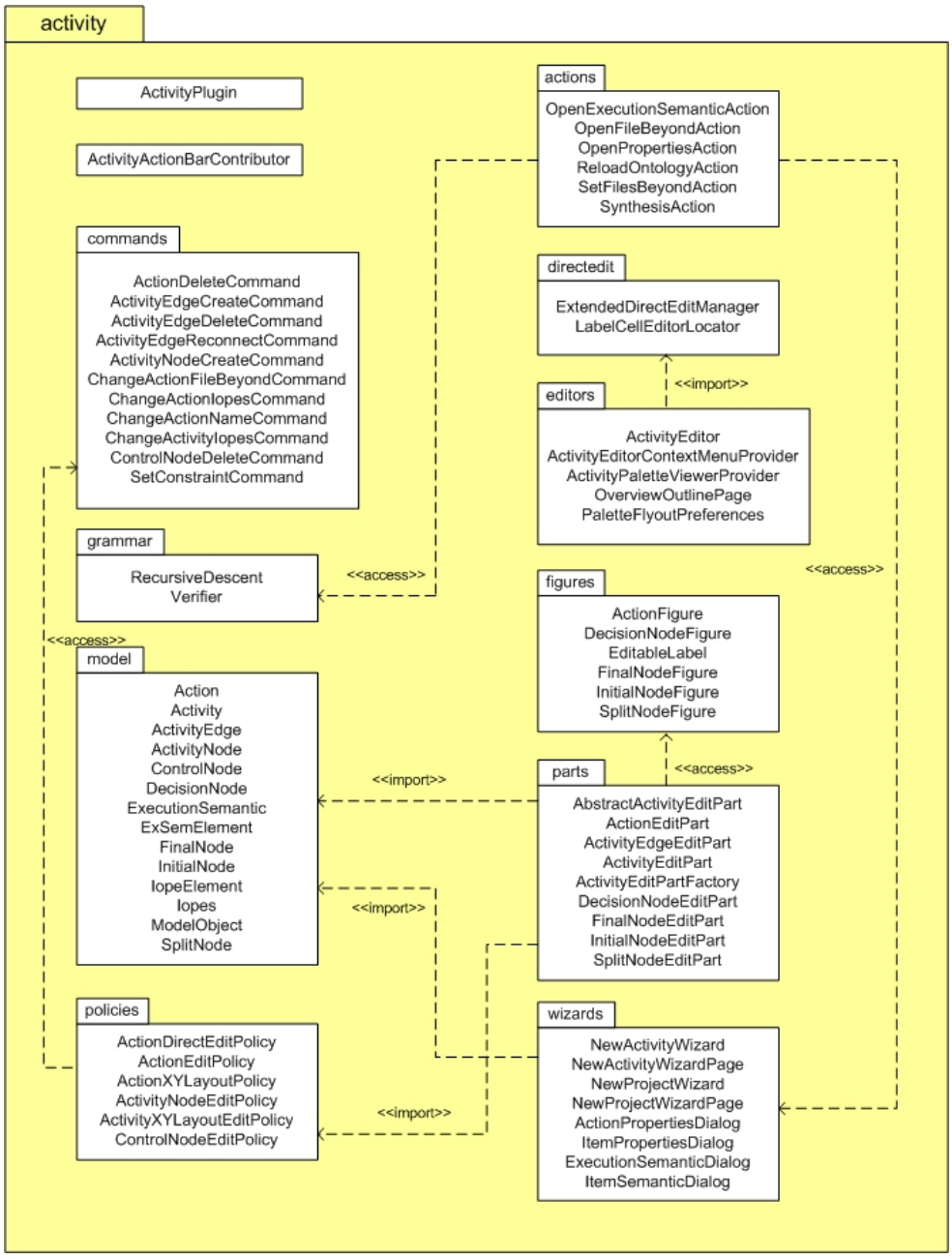
Sequenzdiagramm

Das Sequenzdiagramm verfeinert die Übersicht, wie ein Benutzer mit dem System umgehen muss und welche Aufgaben er damit in welcher Reihenfolge erledigen kann.



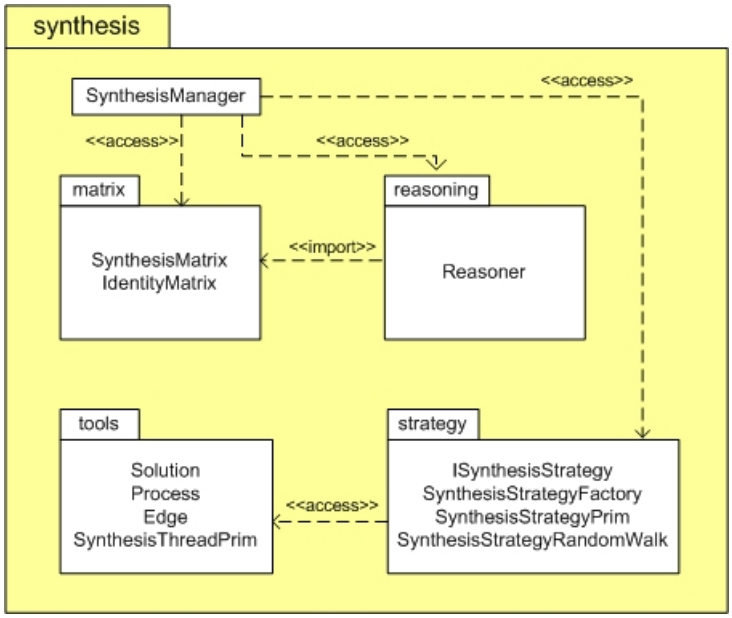
Paket-Diagramme

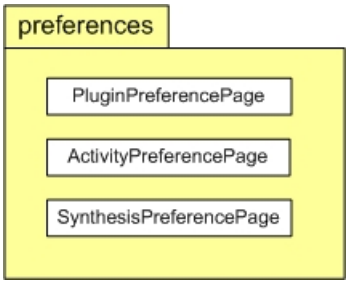
Die Paket-Diagramme zeigen einen Überblick über die implementierten Klassen und Pakete sowie deren Zusammenhänge. Es existieren drei grosse Pakete, die wiederum in kleinere Pakete aufgeteilt wurden. Die drei Hauptpakete sind **activity** zur Erstellung und Änderung von Aktivitätsdiagram-



men. In diesem Paket sind alle Klassen des Editors enthalten (sowohl die Modell- und Controller-Klassen als auch die View-Klassen). Im Paket **synthesis** sind alle Klassen die sich mit der Synthese von Aktionen in einem Aktivitätsdiagramm befassen. Dafür existieren spezielle Unterpakete für die verwendeten Matrizen, die Algorithmen sowie die Arbeit mit der Inferenz-Maschine. Im letzten

Paket (**preferences**) wird die Möglichkeit gegeben diverse Grundeinstellungen vorzunehmen, die sowohl für die Modellierung von Aktivitäten als auch für den Syntheseprozess gelten.





weitere Diagramme

Für weitere Diagramme (z.B. alle Klassendiagramme der Pakete und aller Unterpakete) und Dokumentation des Plugins siehe die SemBPM-Update-Site unter

http://pvs.informatik.uni-augsburg.de/eclipse.