Eine Methodik zur Entwicklung von Multiagenten-Systemen auf Basis von Geschäftsprozess-Modellen

vorgelegt von
Diplom-Informatiker
Tobias Küster
geb. in Berlin-Buch

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Odej Kao
Gutachter: Prof. Dr. Dr. h. c. Sahin Albayrak (TU Berlin)
Gutachter: Prof. Dr. René Schumann (HES-SO Valais-Wallis)
Gutachter: Prof. Dr. Lars Braubach (Hochschule Bremen)

Tag der wissenschaftlichen Aussprache: 30. November 2017

Berlin 2017

Zusammenfassung

Seit vielen Jahren werden Multiagenten-Systeme als Abstraktion für komplexe und verteilte Softwaresysteme eingesetzt, und durch ihren Einsatz können zahlreiche Abläufe auf verständliche und intuitive Art und Weise dargestellt werden. Grundlegende Probleme verteilter Softwaresysteme bleiben jedoch oft bestehen: Der Entwickler sieht immer nur einen Teil des Programmcodes, nur eine Seite der Interaktion, nicht das Gesamtsystem.

In einen anderen Bereich verteilter Systeme, den serviceorientierten Architekturen, werden seit längerem Geschäftsprozess-Diagramme eingesetzt. Durch die verwendeten Abstraktionen haben diese zahlreiche Konzepte mit Agenten gemein, und sind daher gut dazu geeignet, Multiagenten-Systeme als Ganzes zu modellieren. Zugleich stellen Agenten eine mächtige Ausführungsplattform für die Prozesse dar.

In dieser Arbeit wird der Frage nachgegangen, wie Multiagenten-Systeme ganzheitlich mit Hilfe von Geschäftsprozessdiagrammen modelliert, und wie Agenten für deren Ausführung eingesetzt werden können. Zu diesem Zweck werden abstrakte Metamodelle sowohl für Agenten als auch für Prozesse aufgestellt und eine Abbildungsvorschrift beschrieben, über die sich die Prozessmodelle eindeutig in entsprechende Agenten-Systeme überführen lassen.

Diese Abbildungsvorschrift ist in eine Methodik eingebettet, wie diese in Kombination mit anderen Techniken eingesetzt werden kann, um den agentenorientierten Software-Entwicklungsprozess zu vereinfachen und zu beschleunigen. Zur Unterstützung dieser Methodik wird ein spezielles Werkzeug zur Prozessmodellierung vorgestellt, in dem die beschriebene Abbildung auf mehrere verschiedene Arten umgesetzt wird. Die Abbildung und ihre Implementierungen wurden erfolgreich in verschiedenen Projekten eingesetzt.

Abstract

For many years, multi-agent systems have been used as an abstraction for complex and distributed software systems, and have helped in modelling many programs in an understandable and intuitive way. However, fundamental problems of distributed systems often remain: The developer sees just one part of the program code, just one side of the interaction, but not the entirety of the system.

In another field of distributed systems, for service-oriented architectures, business process diagrams are being used. Those have many abstractions in common with agents, and are thus well suited for modelling multi-agent systems as a whole. At the same time, agents constitute a powerful execution environment for those processes.

This thesis will investigate the question, how multi-agent systems can be modelled holistically by means of business process diagrams, and how agents can be employed for their execution. To this end, we will establish meta-models for both, agents and processes, and describe a mapping specification, how process models can be unambiguously converted into accordant agent systems.

This mapping is embedded into a method, combining it with other techniques, in order to simplify and accelerate the agent-oriented software development process. This method is further supported by a special process modelling tool, implementing the described mapping in different ways. Both the mapping and its implementations have been used successfully in a number of research projects.

Inhaltsverzeichnis

I. Einleitung und Analyse **1**

1. Einleitung **3**

1.1. Problemstellung und Motivation 3
1.2. Hintergründe 5
1.3. Ziele und Lösungsansatz 8
1.4. Gliederung 10

2. Agentenorientierte Softwaresysteme **11**

2.1. Definition und Eigenschaften 12
2.2. Architekturen 14
2.3. Multiagenten-Systeme 18
2.4. Agenten-Sprachen und -Frameworks 21
2.5. Fazit 25

3. Techniken der Prozessmodellierung **27**

3.1. Prozessnotationen 28
3.2. Vergleich der Prozessnotationen 33
3.3. Prozessmodellierung in der Softwareentwicklung 38
3.4. Fazit 43

4. Verwandte Arbeiten und Ansätze **45**

4.1. Schnittstellen zwischen Agenten und Webservices 46
4.2. Ausführung von Prozessen durch Agenten 47
4.3. Übersetzungen zwischen Prozessmodellen und Agenten 51
4.4. Notationen, Methodologien und Werkzeuge 53
4.5. Fazit 60

II. Konzept und Ausarbeitung 63

5. Lösungsansatz und Konzept 65
5.1. Auswertung des Stands der Forschung . . . 65
5.2. Ziel dieser Arbeit . . . 67
5.3. Ansatz . . . 70

6. Beschreibung der Metamodelle für Agenten und Prozesse 73
6.1. Ausarbeitung des Agenten-Metamodells . . . 74
6.2. Ausarbeitung des Prozess-Metamodells . . . 80
6.3. Beispiel . . . 87

7. Konzeption und Ausarbeitung der Abbildung von Prozessen auf Agenten 91
7.1. Überblick und Konzeption . . . 92
7.2. Diskussion und mögliche Erweiterungen . . . 99
7.3. Formale Beschreibung der Abbildungsregeln . . . 108

8. Methodik und Implementierung 121
8.1. Methodik zur prozessorientierten Entwicklung von Multiagenten-Systemen 122
8.2. Implementierung des Modellierungswerkzeugs . . . 127
8.3. Implementierung der Abbildung . . . 140

III. Auswertung und Zusammenfassung 153

9. Anwendung und Evaluation 155
9.1. Anwendungsbeispiel . . . 155
9.2. Anwendung in Projekten . . . 158
9.3. Diskussion . . . 163

10. Zusammenfassung 171
10.1. Zusammenfassung der Arbeit . . . 171
10.2. Ergebnisse und Diskussion . . . 173
10.3. Fazit . . . 175

IV. Anhänge **177**

A. Auktions-Beispiel: Modelle und Ressourcen **179**

A.1. Formale Beschreibung des Beispiel-Modells 179

A.2. Erzeugter JIAC V AgentBeans-Quellcode 183

Literaturverzeichnis **189**

Abbildungsverzeichnis

1.1. Integration und Kontributionen von MAS, SOA und BPM 5

2.1. Allgemeines Agenten-Input/Output-Diagramm nach Russel und Norvig . 15
2.2. Schematische Darstellung von JIAC-Nodes und Agenten 24

3.1. BPMN-Beispieldiagramm 30
3.2. Strukturierte Workflow-Modelle 40

4.1. GO-BPMN-Beispieldiagramm 49
4.2. GPMN-Beispieldiagramm 50
4.3. Agent UML-Protokolldiagramm „Contract Net“ 55
4.4. PIM4Agents-Interaktionsdiagramm 57
4.5. Gaia: Prozess und Modelle 58
4.6. Phasen der Prometheus-Methodik 59

5.1. Übersicht über den Ansatz und die Arbeitspakete 70

6.1. Klassendiagramm des Agenten-Metamodells, geringfügig vereinfacht . . . 76
6.2. Klassendiagramm des Prozess-Metamodells, geringfügig vereinfacht . . . 83
6.3. BPMN-Beispieldiagramm zur Illustration der Modelle 88

7.1. Überblick über die Abbildung vom Prozessmodell auf Agenten 93
7.2. Abdeckung der Konzepte seitens BPMN und Agenten 100
7.3. Aufspalten von Lanes zu kommunizierenden Pools 102
7.4. Beispiel für die Verwendung des Ad-hoc-Subprozesses 107
7.5. Abbildung der Prozessstruktur und des Kontrollflusses 113

8.1. Prozessorientierte Entwicklung von Multiagenten-Systemen 123
8.2. Klassendiagramm: Messages und Services im BPMN-Metamodell 129
8.3. Klassendiagramm: Struktur-Elemente im BPMN-Metamodell 130
8.4. Screenshot des Visual Service Design Tools 131

8.5. Normalisierung und Strukturierung 135
8.6. Normalisierungsregeln 136
8.7. Strukturierungsregeln 139
8.8. Abbildungsregel für die Activity-Methode 145
8.9. Schematischer Aufbau des Prozess-Interpreter-Agenten 150

9.1. BPMN-Diagramm des Auktionsprotokolls, modelliert im VSDT 156

A.1. BPMN-Diagramm des Auktionsprotokolls, annotiert 180

Tabellenverzeichnis

3.1. Unterstützung von Workflow Patterns 35

7.1. Abdeckung von Task-, Event- und Gateways-Typen 104

8.1. Überblick der Abbildung von BPMN auf JADL++-Services 141
8.2. Überblick der Abbildung von BPMN auf JIAC AgentBeans 144
8.3. Abbildung von Aktivitäten und Ereignissen auf JIAC-Konzepte 146
8.4. Überblick der Abbildung per BPMN-Interpreter-Agent 149

9.1. Vergleich der Anwendung in den verschiedenen Projekten 162
9.2. Vergleich der Eigenschaften der verschiedenen Umsetzungen 164
9.3. Vergleich der verschiedenen Abbildungen von BPMN nach X 165

Veröffentlichungen

Teile dieser Dissertation sind Thema der folgenden Veröffentlichungen:

Artikel

- T. Küster, M. Lützenberger, A. Heßler, and B. Hirsch. Integrating process modelling into multi-agent system engineering. *Multiagent and Grid Systems*, 8(1):105–124, January 2012.
- T. Küster, A. Heßler, and S. Albayrak. Process-oriented modelling, creation, and interpretation of multi-agent systems. *Int. J. Agent-Oriented Software Engineering*, 5(2/3):108–133, 2016.
- J. Fähndrich, T. Küster, and N. Masuch. Semantic service management and orchestration for adaptive and evolving processes. *International Journal on Advances in Internet Technology*, 9(3&4):75–88, 2016.

Konferenzbeiträge

- H. Endert, B. Hirsch, T. Küster, and S. Albayrak. Towards a mapping from BPMN to agents. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering – AAMAS 2007 International Workshop, SOCASE 2007, Honolulu, HI, USA, May 14, 2007. Proceedings*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin Heidelberg, 2007.
- H. Endert, T. Küster, B. Hirsch, and S. Albayrak. Mapping BPMN to agents: An analysis. In M. Baldoni, C. Baroglio, and V. Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies (AWESOME)*, pages 43–58, 2007.
- T. Küster and A. Heßler. Towards transformations from BPMN to heterogeneous systems. In D. Ardagna, M. Mecella, and J. Yang, editors, *Business Process Ma-*

nagement Workshops – BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers, volume 17 of *LNBIP*, pages 200–211. Springer Berlin Heidelberg, 2009.

- T. Küster, M. Lützenberger, A. Heßler, and B. Hirsch. Integrating process modelling into multi-agent system engineering. In M. Huhns, R. Kowalczyk, Z. Maamar, R. Unland, and B. Vo, editors, *Proceedings of the 5th Workshop of Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE)*, 2010.

- T. Küster, A. Heßler, and S. Albayrak. Towards process-oriented modelling and creation of multi-agent systems. In F. Dalpaiz, J. Dix, and M. B. van Riemsdijk, editors, *Engineering Multi-Agent Systems – Second International Workshop, EMAS 2014, Paris, France, May 5-6, 2014, Revised Selected Papers*, volume 8758 of *LNAI*, pages 163–180. Springer International Publishing, 2014.

- C.-E. Hrabia, F. D. Pozo Pardo, T. Küster, and S. Albayrak. Multi-stage smart grid optimisation with a multiagent system (demonstration). In R. H. Bordini, E. Elkind, G. Weiß, and P. Yolum, editors, *Proc. of 14th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, Istanbul, Turkey, May 4–8 2015. IFAAMAS.

- T. Küster and M. Lützenberger. An overview of a mapping from BPMN to agents (extended abstract). In R. H. Bordini, E. Elkind, G. Weiß, and P. Yolum, editors, *Proc. of the 14th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, pages 1783–1784, Istanbul, Turkey, May 4–8 2015.

- T. Küster, M. Lützenberger, and S. Albayrak. A formal description of a mapping from business processes to agents. In M. Baldoni, L. Baresi, and M. Dastani, editors, *Engineering Multi-Agent Systems – Third International Workshop, EMAS 2015, Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers*, volume 9318 of *LNAI*, pages 153–170. Springer International Publishing, 2015.

- J. Fähndrich, T. Küster, and N. Masuch. Semantic service management for enabling adaptive and evolving processes. In *Proc. of 11th Int. Conf. on Internet and Web Applications and Services (ICIW 2016)*, Valencia, Spain, May 22-26 2016.

- N. Masuch, T. Küster, J. Fähndrich, M. Lützenberger, and S. Albayrak. A multiagent platform for augmented reality based product-service systems (demonstration). In S. Das, E. Durfee, K. Larson, and M. Winikoff, editors, *Proc. of 16th Int.*

Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2017), Sao Paulo, Brazil, May 8–12 2017.

Betreute Abschlussarbeiten

- M. Holtz. Programmierung reaktiver Agenten mit Rule Engines. Bachelor thesis, Technische Universität Berlin, April 2010.
- P. S. Tan. Automated generation of JIAC agentbeans from BPMN diagrams. Diploma thesis, Technische Universität Berlin, November 2011.
- M. Voß. Orchestrating Multi-Agent Systems with BPMN by Implementing a Process Executing JIAC Agent Using the Visual Service Design Tool. Master thesis, Humboldt Universität Berlin, May 2014.

Teil I.

Einleitung und Analyse

Kapitel 1.

Einleitung

Komplexe und verteilte Software kann mit Agenten und Multiagenten-Systemen auf besonders intuitive und verständliche Art und Weise modelliert werden, wobei der Schwerpunkt weniger auf dem *Wie* der Umsetzung und mehr auf dem *Was* und dem *Warum* der Handlungen und Interaktionen liegt, doch fehlt es in diesem Bereich mitunter an standardisierten und intuitiv verständlichen Notationen und Werkzeugen. Zugleich hat die Geschäftsprozessmodellierung zahlreiche Konzepte mit Agenten gemein und wird erfolgreich für die Beschreibung von Service-Orchestrierungen eingesetzt, die jedoch oft nur einen Teil ihrer Ausdrucksfähigkeiten nutzen.

In dieser Arbeit soll daher der Frage nachgegangen werden, inwiefern sich Multiagenten-Systeme mit Hilfe etablierter Notationen aus dem Bereich der Geschäftsprozessmodellierung beschreiben lassen.

Hierzu wird eine Methode zur prozessorientierten Entwicklung von Multiagenten-Systemen vorgestellt, in welcher die Verhaltensweisen von Agenten und deren Interaktionen mit Geschäftsprozessdiagrammen modelliert und in entsprechende Multiagenten-Systeme überführt werden. Auf diese Weise lassen sich Agenten einerseits einfach und in einer etablierten Notation modellieren und andererseits für die Ausführung der Prozesse einsetzen.

Im Folgenden soll zunächst die genaue Problemstellung und Motivation dieser Arbeit dargestellt werden. Anschließend wird ein kurzer Überblick über die Hintergründe dargelegt, bevor schließlich die konkreten Ziele dieser Arbeit und den dafür vorgesehenen Lösungsansatz beschreiben werden.

1.1. Problemstellung und Motivation

Auf dem Gebiet der verteilten Softwareentwicklung gibt es seit einigen Jahren zwei hauptsächliche Strömungen: Auf der einen Seite setzt die Industrie auf serviceorientierte

Architekturen (SOA), oft in Form von Webservices die in Geschäftsprozessen miteinander verknüpft werden; auf der anderen Seite werden Multiagenten-Systeme erforscht, die der klassischen SOA in manchen Aspekten überlegen sind, aufgrund mangelnder Standardisierung und Werkzeugunterstützung jedoch oft ein Nischendasein fristen [10].

Jedes dieser Gebiete – serviceorientierte Architekturen, Geschäftsprozesse und Multiagenten-Systeme – hat bestimmte Stärken und Schwächen. Die Welt der SOA besticht durch plattform- und geräteübergreifende Standards und Schnittstellen, wie SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) und REST (Representational State Transfer), und Sprachen wie OWL-S (Web Ontology Language for Web Services) und SWRL (Semantic Web Rule Language) erlauben die Beschreibung der Semantik der einzelnen Dienste. Geschäftsprozesse können dazu genutzt werden, auf hoher Abstraktionsebene Service-Orchestrierungen zu beschreiben (Kapitel 3), und setzen hierbei auf einfach verständliche und zugleich überaus mächtige graphische Notationen wie etwa BPMN (Business Process Model and Notation). Agenten und Multiagenten-Systeme wiederum können durch nachrichtenbasierte Kommunikation und zielorientiertem Verhalten und Planung mit großer Flexibilität und Anpassungsfähigkeit aufwarten (Kapitel 2).

Zugleich verwenden die drei Gebiete zahlreiche gemeinsame Konzepte und Abstraktionen, und es existieren verschiedene Ansätze, welche die Gebiete zu integrieren versuchen (Kapitel 4). Die Verbindung von SOA und Geschäftsprozessen ist weit fortgeschritten, mit zahlreichen Modellen und Notationen die speziell für die Verknüpfung von Webservices konzipiert wurden, wie etwa BPEL (Business Process Execution Language) und BPMN. Ansätze zur Integration von Agenten und SOA gehen in aller Regel von Seiten der Agenten aus, indem die Agenten etwa Dienste anbieten und diese auch als Webservices veröffentlichen, und zugleich erlauben, beliebige existierende Webservices aufzurufen. Auch zur Integration von Agenten und Geschäftsprozessen gab es in den letzten Jahren einige Ansätze, in denen etwa Agenten Prozesse ausführen oder Aspekte des Verhaltens von Agenten mit Prozessdiagrammen modelliert werden.

Aufbauend auf diesen Vorarbeiten soll in dieser Arbeit ein Ansatz entwickelt werden, mit dem alle Gebiete moderner verteilter Systeme miteinander integriert und zum gegenseitigen Vorteil genutzt werden können: Mit den Schnittstellen und Interaktionsmöglichkeiten der serviceorientierten Architektur, den graphischen Notationen der Geschäftsprozesse, und der Flexibilität und Adaptivität der Multiagenten-Systeme als zugrunde liegender Ausführungsumgebung (Abbildung 1.1). Zu diesem Zweck soll eine Abbildung von Prozessmodellen auf Multiagenten-Systeme erarbeitet (Kapitel 7), und

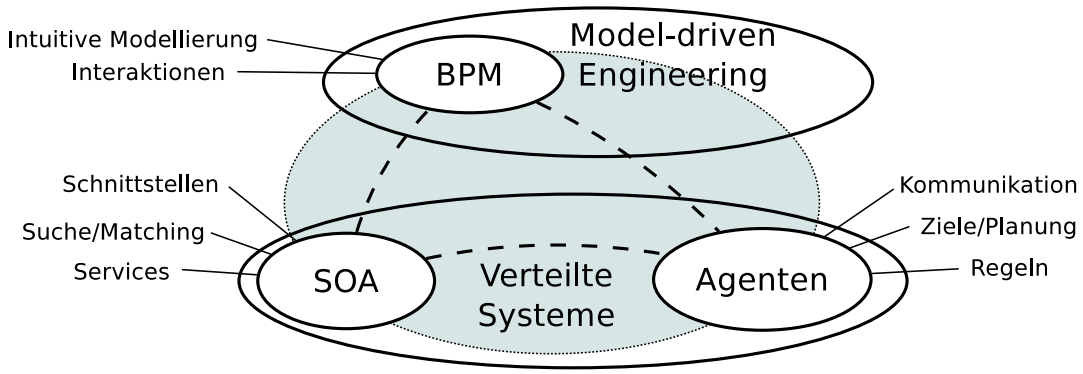


Abbildung 1.1.: Integration und gegenseitige Kontributionen von Multiagenten-Systemen, SOA und Geschäftsprozessmodellierung im Rahmen der modellgetriebenen Softwareentwicklung für verteilte Systeme.

auf deren Basis eine neuartige Methodik zur prozessgetriebenen Entwicklung serviceorientierter Multiagenten-Systeme geschaffen werden (Kapitel 8).

1.2. Hintergründe

Für dieses Problem – die Integration von agentenorientierter Softwareentwicklung mit serviceorientierter Architektur und insbesondere der Modellierung mit Geschäftsprozessdiagrammen – spielen verschiedene Paradigmen der Softwareentwicklung eine Rolle: Neben SOA und Multiagenten-Systemen auch der Bereich des Model-driven Engineering (MDE), und insbesondere die Geschäftsprozessmodellierung. Im Folgenden sollen diese Themen kurz eingeführt werden; die Agenten und Multiagenten-Systeme sowie die Geschäftsprozessmodellierung werden in den Kapiteln 2 und 3, und SOA, MDE und die Integration der verschiedenen Aspekte in Kapitel 4 ausführlicher betrachtet.

1.2.1. Serviceorientierte Architektur

Unter einer *serviceorientierten Architektur* versteht man eine verteilte, oft heterogene Anwendung, deren Grundbausteine eigenständige Dienste (*Services*) sind, die über spezielle Schnittstellen veröffentlicht und plattformübergreifend gefunden, wiederverwendet und verknüpft werden können [47]. Der wichtigste Anwendungsfall einer SOA besteht darin, Dienste unterschiedlicher Anbieter oder Geschäftspartner miteinander zu integrieren. Dementsprechend werden SOAs in der Wirtschaft und der Industrie seit einigen Jahren sehr erfolgreich eingesetzt. Darüber hinaus gibt es zahlreiche Schnittstellen zwi-

schen serviceorientierter Architektur und Geschäftsprozessmodellierung, und in jüngerer Zeit auch zunehmend zwischen SOA und Multiagenten-Systemen.

Hierbei werden die Dienste vom Dienstanbieter (Provider) in einem Dienstverzeichnis (Repository) veröffentlicht, von wo sie vom Dienstnutzer (Client) gefunden und aufgerufen werden können. Die Dienste einer SOA sollen hierbei verschiedene Kriterien erfüllen: Einen klar definierten *Service Contract*, *Loose Coupling* zwischen Diensten, *Abstraction* der zugrunde liegenden Funktionsweise, *Reusability*, *Autonomy* der Dienste, sowie *Statelessness*, *Discoverability* und *Composability* [47].

In der Praxis werden jedoch nicht alle dieser Eigenschaften auch erfüllt. Für die Dienstbeschreibung und deren Interaktion existieren verschiedene Standards und Protokolle, wie etwa WSDL und JSON, SOAP und REST, wodurch die Dienstschnittstellen und die Parameter plattformunabhängig genutzt werden können. Öffentliche und durchsuchbare Dienstverzeichnisse, wie etwa UDDI (Universal Description, Discovery and Integration), werden selten genutzt. Services können in Form von BPEL-Prozessen verknüpft werden [112], müssen dabei jedoch explizit angegeben werden; eine lose Kopplung ist somit nicht gewährleistet. Techniken zur semantischen Beschreibung von Diensten [82], wie etwa mittels OWL-S oder SWRL, mit denen eine Dienstkomposition möglich wäre [81], sind zwar vorhanden, werden außerhalb des akademischen Bereichs jedoch selten eingesetzt.

1.2.2. Multiagenten-Systeme

Einen anderen Ansatz stellen die *Agenten* und *Multiagenten-Systeme* dar. Während es keine allgemein anerkannte Definition des Begriffs „Agent“ gibt, und im Prinzip jede Software als ein solcher verstanden werden kann, zeichnet sich ein typischer Agent durch Eigenschaften wie Reaktivität, Autonomie, Zielorientiertheit und zeitliche Kontinuität aus [53]. Das Verhalten eines Agenten ist nicht fest vorbestimmt, sondern wird zur Laufzeit aus Reaktionsregeln und Zielen abgeleitet. Interaktionen in Multiagenten-Systemen finden in erster Linie mittels nachrichtenbasierter Kommunikation statt.

Zugleich greifen viele Agenten-Frameworks auch Konzepte der SOA auf, und lassen Agenten beispielsweise Dienste anbieten, die von anderen Agenten aufgerufen werden können [66], oder auch Schnittstellen zu bestehenden Webservices darstellen [56]. Hierbei werden die oben beschriebenen SOA-Prinzipien mitunter besser umgesetzt als in einer klassischen SOA: Multiagenten-Systeme verfügen typischerweise über ein Verzeichnis, in dem verfügbare Dienste registriert und gesucht werden können („Directory Facilitator“,

DF). Durch dynamisches Service-Matching und/oder Planung zur Laufzeit kann darüber hinaus das Prinzip der losen Kopplung umgesetzt werden [50].

Ein Problem mit Multiagenten-Systemen – und eine möglicher Ursache, weshalb sie in der Industrie nur vergleichsweise selten eingesetzt werden – stellt die mangelnde Standardisierung und die oftmals fehlende oder eingeschränkte Werkzeugunterstützung dar [10, 12]. Dabei sind übersichtliche Notationen und entsprechende Modellierungswerkzeuge aufgrund der hochgradig dynamischen und verteilten Natur der Multiagenten-Systeme besonders wichtig. Bedingt durch ihre hohe Heterogenität konnte sich nie ein allgemein anerkannter Modellierungsstandard für Agenten durchsetzen, wie es etwa die UML [119] für die objektorientierte Programmierung ist. Stattdessen warten viele Agentenframeworks mit eigenen Notationen, Werkzeugen und Methoden auf.

1.2.3. Modellgetriebene Softwareentwicklung

Bei der *modellgetriebener Softwareentwicklung*, oder *Model-driven Engineering* (MDE), wird das zu entwickelnde System zunächst mit verschiedenen oftmals domänenspezifischen und/oder graphischen Notationen beschrieben, die auch in der weiteren Entwicklung konsequent gepflegt und weiter genutzt werden [147]. Auf diese Weise lassen sich die Modelle nicht nur zur Dokumentation, sondern – je nach vorhandener Werkzeugunterstützung – auch zur Validierung des fertigen Systems oder zur automatischen Erzeugung des eigentlichen Programmcodes nutzen.

In der *Model-driven Architecture* Methodik werden verschiedene Abstraktionsebenen betrachtet: Vom *computation-independent model* über *platform-independent* und *platform-specific model* bis zum eigentlichen Programmcode (der immer noch eine Abstraktion der Maschinensprache darstellt) [115]. Man kann hierbei zwei Teilbereiche unterscheiden: Zum einen die *Domain-specific languages* (DSL), die zur Beschreibung einer bestimmten Domäne genutzt werden können, und zum anderen das *Computer-aided Software-engineering* (CASE), das dabei unterstützt, aus solchen Modellen lauffähige Programme zu erzeugen, etwa per automatischer Codeerzeugung oder Validierung.

Eine der bekanntesten Techniken zur modellgetriebenen Softwareenwicklung ist die Unified Modelling Language (UML) [119], in der Modelle und Notationen für verschiedenste Teilbereiche des Entwicklungsprozesses enthalten sind, von den Anwendungsfällen und Domänenmodellen über Interaktionen bis hin zur Verteilung, wobei die meisten dieser Modelle der Kategorie der *platform-independent models* zugeordnet werden können. Daneben gibt es (quasi-)Standards sowie proprietäre Notationen für zahlreiche andere

Aspekte, und wie oben erwähnt existieren auch für Multiagenten-Systeme verschiedene Modelle, jedoch kein einheitlicher Standard (siehe Kapitel 2 und 4).

1.2.4. Geschäftsprozessmodellierung

Ein weiterer Ansatz zur modellgetriebenen Softwareentwicklung, der gerade im Bereich der verteilten Systeme, der SOA, und zuletzt auch für Multiagenten-Systeme zunehmend an Bedeutung gewinnt, ist die Modellierung von Abläufen mittels Geschäftsprozessdiagrammen (*Business Process Modelling*, BPM).

Wie der Name verrät, werden Geschäftsprozessdiagramme in erster Linie nicht für Software, sondern für Prozesse in der „wirklichen Welt“ eingesetzt [137]. Nach obiger Klassifikation gehören sie somit zu den *computation-independent models*. Zugleich können viele Notationen auch für die Modellierung von Softwaresystemen eingesetzt werden, insbesondere im Bereich der SOA.

Ihr Ursprung im Geschäftsbereich hat unter anderem zur Folge, dass Geschäftsprozessdiagramme, etwa solche des *Business Process Model and Notation* (BPMN) [118], in aller Regel sehr einfach verständlich sind, auch von Anwendern, die wenig Erfahrung mit Softwareentwicklung haben. Im Gegensatz beispielsweise zu Petri-Netzen [108] verfügen sie zudem über viele verschiedene Sprachelemente, mit denen auch komplexe Sachverhalte übersichtlich dargestellt werden können, wie etwa Kommunikation und die Verarbeitung von Ereignissen; zugleich ist die Semantik dieser Elemente allerdings oft weniger klar formuliert als bei Petri-Netzen. Auch die UML enthält mit den Aktivitätsdiagrammen seit jeher eine Art Prozessdiagramm. Verschiedene Prozessnotationen und deren jeweilige Vor- und Nachteile werden im Kapitel 3 näher betrachtet.

Gerade ihr Ursprung in der Beschreibung von Prozessen und Interaktionen zwischen Menschen und Organisationen (im Gegensatz zu Abläufen in einem Computersystem) macht Geschäftsprozessnotationen für die Beschreibung von Multiagenten-Systemen interessant, verwenden diese doch ebenfalls Konzepte wie „Rollen“, „Organisationen“, „Nachrichten“, “Regeln“ und „Ziele“, also Abstraktionen, mit denen die Verhaltensweisen der Agenten auf sehr „menschliche“ Art beschrieben werden.

1.3. Ziele und Lösungsansatz

In dieser Arbeit soll untersucht werden, wie die Entwicklung komplexer, verteilter Systeme vereinfacht und zugänglicher gemacht werden kann. In der Praxis haben sich – in

unterschiedlichen Bereichen – die Paradigmen der agentenorientierten Softwareentwicklung, der serviceorientierten Architekturen, sowie, als besondere Form der modellgetriebenen Softwareentwicklung, die Geschäftsprozessmodellierung etabliert. Diese verschiedenen Ansätze können in unterschiedlicher Weise zur Erreichung dieses Ziels beitragen, und sollen daher in dieser Arbeit miteinander verknüpft werden.

Zu diesem Zweck soll eine Methode entwickelt werden, mit der die Vorteile dieser Paradigmen – serviceorientierte Architektur, Multiagenten-Systeme, modellgetriebene Softwareentwicklung und Geschäftsprozessmodellierung – miteinander kombiniert und zum gegenseitigen Vorteil genutzt werden können (siehe Abbildung 1.1). Das Hauptaugenmerk liegt hierbei auf der Integration von Agenten und Prozessen, d. h. einerseits der Modellierung des Verhaltens von Agenten mit Geschäftsprozessen, und andererseits der Verwendung von Multiagenten-Systemen um selbige Prozesse auszuführen.

Hierbei soll der Frage nachgegangen werden, inwieweit sich Agenten und Multiagenten-Systeme einerseits mit Hilfe etablierter Techniken aus dem Bereich der Geschäftsprozessmodellierung beschreiben lassen, und andererseits für die Ausführung der Prozesse eignen. Damit sollen die folgenden Forschungsfragen beantwortet werden:

- Sind standardisierte Prozessnotationen, ohne bedeutende Veränderungen oder Erweiterungen, für die umfassende Modellierung von Multiagenten-Systemen und deren Verhalten geeignet?
- Sind Multiagenten-Systeme als Ausführungsumgebung von Geschäftsprozessdiagrammen in ähnlichem Maße geeignet, wie gängige serviceorientierte Ansätze?
- Wie muss eine Entwicklungsmethodik gestaltet sein, mit der Multiagenten-Systeme mit Prozessmodellen spezifiziert und erfolgreich für deren Ausführung eingesetzt werden können?

Zu diesem Zweck soll eine Abbildungsvorschrift von Geschäftsprozessmodellen auf Multiagenten-Systeme gefunden werden, mit der diese auf einfache und zugleich umfassende Weise spezifiziert werden können. Hierbei sollen sich insbesondere typische Eigenschaften von Agenten, wie Kommunikation und Interaktion, und sowohl reaktives als auch zielorientiertes Verhalten darstellen lassen, zugleich aber auch gängige SOA-Prinzipien wie die Verwendung von Diensten berücksichtigt werden. Hierfür müssen zunächst Modelle sowohl für Prozesse als auch für Agenten definiert werden, anhand derer die Abbildung beschrieben werden kann. Schließlich soll die Abbildung in einer Entwicklungsmethodik eingebettet und in einem Modellierungswerkzeug prototypisch umgesetzt werden.

Das Ziel dieser Arbeit liegt dabei explizit nicht in der Definition einer weiteren proprietären Sprache für die Beschreibung von Agenten, oder bestimmter Aspekte derselben, sondern darin, bestehende und umfassend akzeptierte Standards, insbesondere aus dem Bereich der Geschäftsprozessmodellierung hierfür einzusetzen.

Das Feld der Multiagenten-Systeme ist außerordentlich weit und heterogen. Es kann und soll daher nicht Ziel dieser Arbeit sein, dass die entwickelten Modelle und Methoden für sämtliche Aspekte der verschiedenen Multiagenten-Plattformen gleichermaßen geeignet sind. Stattdessen konzentriert sich diese Arbeit auf diejenigen Aspekte, für welche die Prozessnotationen am natürlichsten geeignet sind, ohne bestehende Standards übermäßig zu erweitern oder umzudeuten.

1.4. Gliederung

Die Arbeit gliedert sich in drei Teile: In diesem ersten Teil, Einleitung und Analyse, werden die Hintergründe zu dieser Arbeit betrachtet – insbesondere Agenten, Prozesse, SOA, und deren Verbindungen. Kapitel 2 gibt einen Überblick über agentenorientierte Programmierung und Multiagenten-Systeme werfen, und Kapitel 3 widmet sich der Prozessmodellierung und ihrer Anwendungsmöglichkeiten in der Softwareentwicklung. In Kapitel 4 werden schließlich bestehende Ansätze vorgestellt, die auf verschiedene Weise Agententechnologie und Prozessmodellierung miteinander verbinden.

Der zweite Teil, Konzept und Ausarbeitung, befasst sich mit der Konzeption und Ausarbeitung einer Abbildung von Prozessen auf Agenten. Zunächst wird in Kapitel 5 das Konzept für den in dieser Arbeit verfolgten Ansatz dargelegt. Danach werden im Kapitel 6 Metamodelle für Agentensysteme und Geschäftsprozesse konzipiert und formalisiert. In Kapitel 7 wird sodann eine Abbildung ausgearbeitet, mit der die Prozessbeschreibungen in Agentensysteme übertragen werden können. Schließlich, in Kapitel 8 werden eine Methodik sowie eine prototypische Implementierung vorgestellt, mit denen diese Abbildung im Rahmen der agentenorientierten Softwareentwicklung eingesetzt werden kann.

Im letzten Teil, Auswertung und Zusammenfassung, soll der gewählte Ansatz erprobt und ausgewertet werden. Hierfür wird in Kapitel 9 der Einsatz sowohl der Abbildung als auch der Implementierung in verschiedenen Szenarien beschrieben und mit dem Stand der Forschung verglichen. Kapitel 10 schließlich fasst die Arbeit zusammen.

Kapitel 2.

Agentenorientierte Softwaresysteme

Agenten und Multiagenten-Systeme stellen eine besonders intuitive Abstraktion für komplexe und offene Systeme dar. Agenten-Systeme integrieren auch in zunehmendem Maße Techniken und Eigenschaften, die normalerweise mit SOA assoziiert werden, wie etwa semantisch beschriebene Dienste und die Nutzung von Kommunikations- und (Web-)Service-Standards, und verleiben sich somit viele Vorteile der SOA mit ein. Tatsächlich verfügen viele agentenorientierten Softwaresysteme über viele der von einer SOA erwünschten Eigenschaften [66] und werden mitunter als die nächste Stufe in der „Evolution“ der Softwareentwicklung nach der Objektorientierten Programmierung angesehen [75].

So verfügen etwa *deliberative Agenten* über eine Menge möglicher Aktionen, die sie zur Erfüllung vorgegebener Ziele selbständig auswählen und mit von anderen Agenten angebotenen Aktionen kombinieren können. Die Aktionen können dabei entweder direkt den jeweiligen Zielen zugeordnet, oder mit entsprechenden semantischen Beschreibungen (Vorbedingung und Effekt) versehen sein. Diese Autonomie ermöglicht Agenten nicht nur den Einsatz in unbekannten, dynamischen Umgebungen, sondern – gepaart mit robusten Kommunikationswegen und gemeinsam genutzten Wissensrepräsentationen – auch die Interaktion mit anderen Agenten und Diensten unterschiedlicher Art, was sie insbesondere für die Anwendung im Internet und für „Business-to-Business“ interessant macht [78].

In diesem Kapitel wird zunächst ein Überblick über die Grundlagen gegeben und bestimmt, was unter einem Agenten verstanden werden kann. Danach wird eine Klassifikation verschieden mächtiger Typen von Agenten vorgenommen. Nach der Betrachtung von einzelnen Agenten wird dann auf Multiagenten-Systeme und die Möglichkeiten der Interaktion zwischen Agenten eingegangen. Abschließend werden verschiedene Agentenmodelle und -frameworks betrachtet. Das Ziel dieses Kapitels besteht darin herauszustellen, über welche Konzepte Agenten typischerweise verfügen, wie diese in bestehenden

Implementierungen umgesetzt sind, und welche Ansätze für deren Beschreibung vorhanden sind.

2.1. Definition und Eigenschaften

Der Begriff des *Agenten* ist nicht sehr klar definiert. Russel und Norvig schreiben: „An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.“ [141, Kapitel 2.1, S. 32] Dieser einfachen Definition entsprechend werden Agenten bei ihnen anhand ihrer „Agenten-Funktion“ beschrieben, die als Eingabe die aktuelle Wahrnehmung des Weltzustandes und – je nach Agententypus – weitere Informationen, wie etwa das aktuelle Wissen des Agenten oder eine Regelmenge erhält, und als Ergebnis die als nächstes auszuführende Aktion zurück gibt.

Etwas weiter geht die Definition eines *rationalen Agenten* [141, Kapitel 2.2, S. 36]:

> „For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.“

Diese Definition besagt, dass ein rationaler Agent sich, gegeben seiner Fähigkeiten und seiner Kenntnis der Umgebung, so verhalten soll, dass er den größten Nutzen daraus zieht.

Franklin und Grasser [53] haben verschiedene solche Definitionen von Agenten gesammelt und in Beziehung gesetzt. Dabei fällt auf, dass jede Definition einen anderen Schwerpunkt setzt, abhängig vom Hintergrund der jeweiligen Autoren. Franklin und Grasser selbst geben die folgende zusammenfassende Definition [53, Seite 6]:

> „An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.“

Beide Definitionen sind sehr liberal. So treffen diese Kriterien nicht nur auf verschiedene Softwaresysteme, sondern auch auf viele Arten von Maschinen und Robotern (einschließlich so simpler Mechanismen wie eines Thermostats [53]) und insbesondere auch auf Menschen und Gesellschaften zu. Gerade dieser letzte Aspekt legt auch die Modellierung von Agenten mit Geschäftsprozessen nahe, deren eigentlicher Verwendungszweck eben die Beschreibung von menschlichem Verhalten ist.

Die Definition des rationalen Agenten besagt nicht, welche Mittel dem Agenten zur Verfügung stehen, oder allgemeiner, welche Eigenschaften ein Agent aufweisen soll. Im Rahmen dieser Arbeit soll unter einem Agenten, wenn nicht anders genannt, ein Softwaresystem mit bestimmten Eigenschaften verstanden werden, die im nächsten Abschnitt näher beschrieben werden.

2.1.1. Eigenschaften von Agenten

In ihrer Studie haben Franklin und Grasser auch die Eigenschaften gesammelt, die den Agenten in den verschiedenen Definitionen zugeschrieben werden [53]. Die folgenden vier Eigenschaften sehen sie dabei als notwendig für einen Agenten an:

- *reactive*: Der Agent reagiert auf Veränderungen in seiner Umgebung.
- *autonomous*: Der Agent hat Kontrolle über seine eigenen Aktionen.
- *goal-oriented*: Der Agent reagiert nicht nur auf die Umgebung, sondern verfolgt aktiv ein bestimmtes Ziel.
- *temporally continuous*: Der Agent wird nicht für eine bestimmte Aktion gestartet sondern läuft kontinuierlich.

Weitere fünf typisch Eigenschaften von Agenten, die je nach Anwendungsgebiet hinzu kommen können, werden wie folgt angegeben:

- *communicative*: Der Agent kommuniziert mit anderen Agenten oder Menschen.
- *learning*: Der Agent kann sein Verhalten basierend auf Erfahrungen ändern.
- *mobile*: Der Agent kann sich von einer Maschine auf eine andere bewegen.
- *flexible*: Die Aktionen des Agenten sind nicht festgeschrieben.
- *character*: Der Agent ist mit einer glaubhaften Persönlichkeit und einem emotionalen Zustand ausgestattet.

Eine ähnliche Zuschreibung von Eigenschaften findet sich auch bei Wooldridge und Jennings [178]. „Schwache“ Agenten besitzen demnach die Eigenschaften *autonomy*, *social ability*, *reactivity* und *pro-activeness*. „Starke“ Agenten haben zusätzlich noch mentale Zustände bis hin zu Emotionen. Die Frage, ob die Fähigkeit zur Kommunikation (bzw.

social ability bei Wooldridge und Jennings) zu den notwendigen Eigenschaften des Agenten gehört, hängt dabei nicht zuletzt davon ab, ob der Agent allein agiert, oder ob er Teil eines Multiagenten-Systems ist.

In dieser Arbeit sind – neben den obligatorischen Eigenschaften des ersten Blocks – insbesondere *kommunikative* Agenten von Interesse, da es diese Eigenschaft ist, die sich in Form von Geschäftsprozessdiagrammen besonders anschaulich darstellen lässt. Und während die Prozessabläufe das Verhalten der Agenten weitestgehend fest vorgeben, bieten sie doch auch noch Raum für *Flexibilität*, etwa in Form von *ad-hoc*-Regionen.

2.1.2. Eigenschaften von Umgebungen

Neben den Eigenschaften des Agenten selbst spielen auch die Eigenschaften der Umgebung, in der der Agent angesiedelt ist, eine große Rolle bezüglich dessen, was der Agent leisten kann. Russel und Norvig identifizieren die folgenden Eigenschaften von Umgebungen [141]:

- *fully / partially observable*: Kann der Agent über seine Sensoren (zu jedem Zeitpunkt) die gesamte Umgebung wahrnehmen, oder nur einen Teil?
- *deterministic / stochastic*: Ist die Veränderung der Umgebung (als Reaktion auf die Aktion des Agenten) vorhersehbar?
- *episodic / sequential*: Haben die Aktionen des Agenten Einfluss auf den zukünftigen Zustand der Umgebung?
- *static / dynamic*: Kann sich die Umgebung verändern, während der Agent seine nächste Aktion auswählt?
- *discrete / continuous*: Sind die möglichen Zustände kontinuierlich?
- *single agent / multi agent*: Ist der Agent die einzige handelnde Instanz?

Im Kontext dieser Arbeit kommt den Eigenschaften der Umgebung dabei keine besondere Bedeutung zu. Prinzipiell lassen sich Agenten in jede Art von Umgebung mit Geschäftsprozessen modellieren – nur die Prozesse fallen entsprechend anders aus.

2.2. Architekturen

Russel und Norvig [141, Kapitel 2] unterscheiden in ihrem Buch zwischen verschiedenen Klassen von Agentensystemen: Simple Reflex Agents, Model-based Reflex Agents,

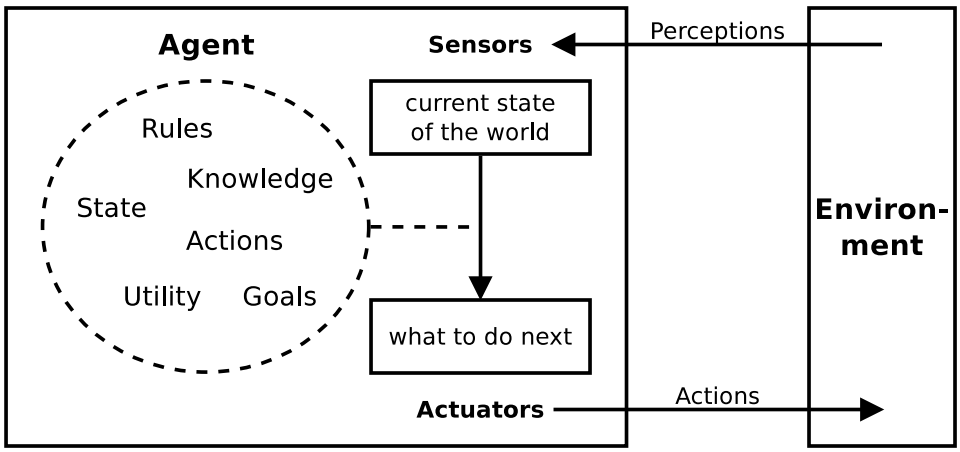


Abbildung 2.1.: Allgemeines Agenten-Input/Output-Diagramm nach Russel und Norvig [141]: Die Umgebung wird wahrgenommen und je nach Agenten-Typus wird durch verschiedene Mechanismen eine Aktion ausgewählt.

Goal-based Agents und Utility-based Agents. Im Folgenden soll diese Klassifikation kurz wiedergegeben werden, wobei eine zusätzliche Unterteilung in reaktive und deliberative Agentensysteme vorgenommen wird. Zuletzt wird auch ein kurzer Blick auf Hybridsysteme und lernende Agenten geworfen.

Allen Agentenklassen ist dabei gemein, dass sie ihre Umgebung über Sensoren wahrnehmen und über Aktuatoren manipulieren; sie unterscheiden sich lediglich in der Art und Weise, auf welche die nächste Aktion ausgewählt wird (siehe Abbildung 2.1).

Weiterhin ist zwischen den hier beschriebenen *Klassen* und den oben genannten *Eigenschaften* von Agenten zu unterscheiden: So bedeutet die (notwendige) Eigenschaft der Zielorientierung nicht, dass jeder Agent notwendigerweise als zielorientierter (*goal-oriented*) Agent implementiert ist, das heißt dass Ziele explizit repräsentiert werden. Stattdessen entscheiden die hier beschriebenen Agentenklassen, wie die oben genannten Eigenschaften von Agenten umgesetzt werden.

2.2.1. Reaktive Agenten

Unter *reaktiven Agenten* werden solcherlei Agenten verstanden, die anhand fester Regeln handeln. Es spielt hierbei keine Rolle, ob für die Programmierung eine spezielle Regelengine verwendet wird, oder ob die Regeln direkt im Quellcode des Agenten, in Form von *if/else*-Statements, untergebracht sind.

Während diese Art Agent sehr einfach aufgebaut ist, sind sie für viele Einsatzgebiete doch die erste Wahl, insbesondere wenn die Situationen, die der Agent bewältigen muss,

einfach vorhersehbar sind, sodass es möglich ist, diese mit einer entsprechenden Regelmenge abzudecken. Reaktive Agenten stoßen hingeben insbesondere in komplexeren Umgebungen an ihre Grenzen, wenn sie mit unvorhergesehenen Situationen konfrontiert werden.

Simple reflex Agent Der *Simple reflex Agent* ist der einfachste reaktive Agent. Er ist zustandslos, hat also keinen internen Langzeitspeicher, weswegen er sich in jeder Situation nur auf die aktuellen Sensordaten verlassen muss. Der Vorteil eines solchen Agenten ist, dass die Agentenfunktion sehr einfach zu überprüfen und zu testen ist, da nur die Sensoreingaben entsprechend variiert werden müssen. Ein solcher Agent kann jedoch prinzipiell nur in voll beobachtbaren Umgebungen optimal handeln, da es ihm nicht möglich ist, Informationen über die zur Zeit nicht sichtbaren Bereiche der Umgebung zu speichern.

Model-based reflex Agent Der *Model-based reflex Agent* ist ein reaktiver Agent mit internem Speicher. Hiermit kann er den Zustand der Umgebung (insbesondere der derzeit nicht beobachtbaren Teile) sowie Informationen über sich selbst, beispielsweise die zuletzt ausgeführten Aktionen, in die Auswahl der auszuführenden Aktion mit einfließen lassen. Auf diese Weise kann der Agent etwa eine Karte seiner Umgebung erstellen und diese systematisch erkunden. Somit kann ein Model-based reflex Agent auch in partiell beobachtbaren Umgebungen gutes Verhalten aufweisen.

2.2.2. Deliberative Agenten

Deliberative Agenten unterscheiden sich von reaktiven Agenten dadurch, dass sie nicht einfach nur auf ihre Umgebung reagieren, sondern „proaktiv“ handeln und vorausschauend planen, oder – ihrer Bezeichnung entsprechend – abwägen, welche Aktion ausgeführt werden soll.

Um planen zu können ist es notwendig, dass der Agent über seinem aktuellen Weltwissen (den aktuellen Sensordaten und dem Speicherinhalt) schlussfolgern kann, wozu das Weltwissen in der Regel in logische Ausdrücke überführt wird. Außerdem müssen sämtliche möglichen Aktionen des Agenten neben Eingabeparametern und Ergebnissen auch mit Vorbedingungen und (erwarteten) Effekten annotiert sein; schließlich kann der Agent nur planen, welche Aktion zum Ziel führt, wenn er weiß, welche Effekte die Aktionen haben.

Sind diese Voraussetzungen erfüllt, ist der eigentliche Planungsvorgang nur noch eine Suche im Raum der möglichen Aktionen, wofür es neben einfachen Suchverfahren auch ausgeklügelte (zum Teil jedoch auch mit starken Einschränkungen versehene) Planungsalgorithmen gibt.[1]

Während deliberative Agenten in der Theorie sehr gute Ergebnisse erzielen und auch mit unvorhergesehenen Situationen zurecht kommen, stellt die in der Praxis oft sehr viel größere Zahl an möglichen Aktionen ein Problem dar, da der zu durchsuchende Planungsraum schnell zu groß wird. Speziell in Multiagenten-Systemen ist der Agent zudem darauf angewiesen, dass nicht nur seine eigenen, sondern auch die Aktionen der anderen Agenten (oder externer Dienste) ausreichend und korrekt annotiert sind.

Goal-based Agent Die einfachere Art deliberativer Agenten sind *zielorientierte Agenten*. Diese Agenten haben ein oder mehrere Ziele, die sie durch die deliberative Anwendung ihrer Aktionen zu erfüllen versuchen. Hierbei sind insbesondere zwei Arten von Zielen üblich: Achievement Goals und Maintenance Goals, wobei bei dem ersten das Erreichen eines Zustands, und bei letzterem dessen Erhaltung das Ziel ist [16, 162].

Was genau die Ziele des Agenten sind wird diesem in der Regel von außen mitgeteilt; es ist jedoch auch möglich, dass in Zuge der Planung weitere Ziele aufgestellt werden, d. h. die Erfüllung eines Teilzieles (die ihrerseits wieder Planung erfordert) wird als Teil des Plans zur Erfüllung eines größeren Zieles verwendet. Ein Problem, das zielorientierte Agenten haben, ist jedoch, dass sie nicht gut mit sich widersprechenden Zielen umgehen können.

BDI-Agent Eine Untergruppe der zielorientierten Agenten stellen die *BDI-Agenten* dar. Basierend auf der Arbeit von Michael Bratman [14] über die Reduktion menschlichen Verhaltens auf die drei Modalitäten *Belief*, *Desire* und *Intention* (BDI) wurden verschiedene logische Formalismen vorgeschlagen, mit denen sich „menschliches“ logisches Denken ausdrücken lässt.

Hierbei wird besondere Aufmerksamkeit auf die Intentions gelegt, also jene Pläne, zu denen sich der Agent bereits verpflichtet, oder *committed* hat. Diese werden explizit repräsentiert und können bei einer späteren Planung wieder in Betracht gezogen werden. Der eigentliche Planungsvorgang hat im BDI-Modell weniger Bedeutung.

[1] Welcher Algorithmus verwendet wird ist im Rahmen dieser Arbeit nicht von Bedeutung, weswegen auf die Details an dieser Stelle nicht weiter eingegangen wird.

Utility-based Agent Eine andere Art deliberativer Agent ist der *nutzenorientierte Agent*. Dieser hat keine speziellen Ziele, sondern ordnet jedem Zustand einen Nutzen zu. Statt auf die Erfüllung eines Zieles hin zu planen, wird dann eine Folge von Aktionen gesucht, die dem Agenten kurz-, mittel- oder langfristig den größten Nutzen zu bringen versprechen.

Der Nutzen eines Zustands wird hierbei mithilfe einer Nutzenfunktion gemessen, in die verschiedene Aspekte des aktuellen Zustands mit unterschiedlichen Gewichtungen einfließen.[2] Während ein solcher Agent besser mit Ungewissheit und sich widersprechenden Zielen umgehen kann, als ein zielorientierter Agent, stellt das Finden einer geeigneten Nutzenfunktion (bzw. der richtigen Gewichte für die einzelnen Aspekte) mitunter eine große Schwierigkeit dar.

2.2.3. Hybride und Lernende Agenten

Neben reinen reaktiven und deliberativen Agenten gibt es auch Mischformen, sogenannte *hybride Agenten*, etwa als Mehr-Schichten-Architektur [107], wobei in der Regel für höhere Funktionen Planung eingesetzt wird, bestimmte Ereignisse aber auch reaktiv behandelt werden können. Dies hat insbesondere in der Robotik Vorteile, wo komplexe Umgebungen die Fähigkeit zur Planung erfordern, zugleich aber die Reaktionsgeschwindigkeit eines reaktiven Agenten gefordert ist [141, Kapitel 25].

Ferner gibt es auch *lernende Agenten*, d. h. Agenten, die ihr Verhalten über ein bestimmtes Lernelement an ihre Umgebung anpassen können, etwa indem einzelne Regeln oder Ziele aktiviert oder deaktiviert oder Parameter angepasst werden.

2.3. Multiagenten-Systeme

In den bisherigen Abschnitten lag der Fokus auf einzelnen Agenten, auch wenn bereits Eigenschaften wie *communicative* und *multi-agent* eingeführt wurden. In diesem Abschnitt sollen nun aus *mehreren* Agenten bestehende Systeme betrachtet werden.

Die Abgrenzung zwischen Einzel- und Multiagenten-Systemen ist dabei nicht immer eindeutig [53]. So kann etwa bei der Subsumption Architektur jede Schicht als ein eigenständiger Agent betrachtet werden, der mit den anderen Schichten kooperiert, dabei aber eigene Ziele verfolgt (auf höherer Ebene etwa „von A nach B kommen“ und auf

[2] Insbesondere wäre es möglich, einen nutzenorientierten Agenten zum zielorientierten Agenten umzuwandeln, indem man den Grad der Erfüllung des aktuellen Ziels als Nutzenfunktion verwendet.

niedrigerer Ebene „Hindernissen ausweichen“); nach außen hin handeln die verschiedenen Teile aber als ein einzelner Agent, und sollte auch als solcher betrachtet werden. Das gleiche trifft auch auf Unternehmen und generell auf Gruppen und Teams zu. Ein gutes Kriterium, um zwischen Einzelagenten und Multiagenten-Systemen zu unterscheiden, ist, wie stetig die Gruppierung ist.

Ein wichtiger Aspekt eines Multiagenten-Systems ist, dass einzelne Agenten jederzeit dem System beitreten oder es verlassen können. Auch müssen die Agenten nicht unbedingt kooperieren, sondern können auch in Konkurrenz zueinander stehen. In jedem Fall muss die *Kommunikation* zwischen den einzelnen Teilen des Systems, also den Agenten, gewährleistet werden.[3] Hierfür müssen die Agenten über geeignete Kommunikationsmittel verfügen, insbesondere gemeinsam genutzte *Protokolle, Sprechakte und Ontologien.*

Kommunikation und Koordination in Multiagenten-Systemen, bzw. die Modellierung derselben, stellt auch einen wichtigen Aspekt in dieser Arbeit dar, weswegen in den nächsten Abschnitten näher auf diese Aspekte eingegangen werden soll.

2.3.1. Vokabular: Ontologien

Vor allem anderen müssen Agenten, wenn sie untereinander kommunizieren sollen, eine gemeinsame Repräsentation für den Gegenstand ihrer Kommunikation haben, über die die Semantik transportiert wird. Im Kontext agentenorientierter Softwaresysteme spricht man hierbei in der Regel von *Ontologien* [64].

Eine Ontologie stellt ein wiederverwendbares, kommunizierbares Modell der Welt dar, in der der Agent situiert ist – eine Sprache, in die er seine Sensoreindrücke übersetzen kann, mit der er sein Weltmodell aufbauen, Schlüsse daraus ziehen und sein Wissen mit anderen Agenten teilen kann.

Dabei spielt die Form zunächst keine Rolle: Neben einer Darstellung über Prädikate (Prädikat, Subjekt, Objekt) sind hier insbesondere auch die aus der objektorientierten Programmierung und der Unified Modeling Language (UML) bekannten Klassenmodelle verbreitet. Darüber hinaus gibt es auch spezielle Ontologiesprachen, wie etwa die Web Ontology Language (OWL) [99]. Diese verbindet Klassenhierarchien mit Aspekten der Prädikatenlogik und bietet weitergehende Möglichkeiten an, Fakten darzustellen und Schlussfolgerungen zu ziehen (*reasoning*).

[3] Es wäre zwar auch möglich, dass die Agenten in keiner Weise interagieren; in diesem Fall wäre es jedoch fraglich, ob man überhaupt von einem Multiagenten-System sprechen kann, oder ob die einzelnen Agenten lediglich einen Teil der dynamischen Umgebung der jeweils anderen Agenten darstellen.

Neben speziell auf eine Domäne zugeschnittenen Ontologien gibt es auch zahlreiche Bestrebungen, universelle Ontologien aufzustellen [33] in denen das gesamte Weltwissen maschinenlesbar verknüpft wird, etwa im Bereich des *Semantic Web*. In offenen Multiagenten-Systemen, in denen Agenten ein- und austreten können, spielt insbesondere auch die Abstimmung von und die Verständigung mit voneinander abweichenden Ontologien eine große Rolle, doch diese Aspekte gehen über den Rahmen dieser Arbeit hinaus.

2.3.2. Kommunikation: Sprechakte

Während über Ontologien die Inhalte beschrieben werden, welche die Agenten untereinander kommunizieren, wird noch eine Form benötigt, wie diese ausgetauscht werden können. Hierfür werden *Sprechakte* [148] verwendet.

Während bei der objektorientierten Programmierung Methodenaufrufe verschickt werden, die bedingungslos ausgeführt werden, enthalten Sprechakte lediglich eine Information oder eine Anfrage; wie der Agent mit diesen verfährt bleibt ihm überlassen [36]. Diese sogenannten „Locutions“ werden über verschiedene Typen von Sprechakten explizit gemacht, bspw. INFORM(FACT), REQUEST(ACTION) oder CONFIRM(REQUEST).

Die *Foundation for Intelligent Physical Agents* (FIPA) hat eine Reihe von Standards entwickelt, welche die Interoperabilität von Agenten sicherstellen sollen, darunter die *Communicative Acts Library* und die *Agent Communication Language* (ACL).[4]

2.3.3. Koordination: Protokolle

Diese Sprechakte ermöglichen es den Agenten beispielsweise, Informationen auszutauschen oder Anfragen zu stellen. Für komplexere Kommunikation, die mehrere Nachrichtenwechsel oder mehr als zwei Kommunikationspartner erfordert, werden zusätzlich aber auch gemeinsam genutzte *Interaktionsprotokolle* benötigt.

Bekannte Einsatzgebiete für Interaktionsprotokolle sind beispielsweise Auktionen, Verhandlungen und Ausschreibungen, wie „Contract Net“ [152]. Dabei kommt das Protokoll für eine Auktion längst nicht nur dann zum Einsatz, wenn mit dem Multiagenten-System tatsächlich eine Auktion umgesetzt werden soll (etwa bei einer agentenbasierten Implementierung eines Online-Auktionshauses); Auktionsprotokolle können auch an verschiedenen anderen Stellen eingesetzt werden, beispielsweise zur Lastverteilung. Das gleiche gilt analog für andere Arten von Protokollen.

[4] Foundation for Intelligent Physical Agents: `http://www.fipa.org`.

Mehrere Protokolle, insbesondere für verschiedene Auktionstypen, wurden von der FIPA spezifiziert. Für die Darstellung der Protokolle werden hierbei Interaktionsdiagramme der *Agent UML* verwendet [4].

2.4. Agenten-Sprachen und -Frameworks

Die oben genannten Formalismen wurden bis heute in unterschiedlicher Ausprägung in einer Vielzahl von Sprachen und Frameworks umgesetzt, von denen viele jedoch rein akademischer Natur sind. In diesem Abschnitt wird ein Überblick über einige ausgewählte Agenten-Frameworks gegeben, in denen diese in unterschiedlichem Maße implementiert werden. Angesichts der großen Anzahl verfügbarer Sprachen und Frameworks liegt der Fokus dabei auf besonders weit verbreiteten Frameworks, und solchen, die in ähnlichen Arbeiten (siehe Kapitel 4) eingesetzt werden. Umfangreiche Vergleiche verschiedener Sprachen, Plattformen und deren Anwendungen wurden unter anderem von Bordini et al. aufgestellt [11, 10, 12]. Ein umfassender Überblick zahlreicher Agentensprachen, Frameworks, Methodologien und Entwicklungsumgebungen und die Agentenorientierten Softwareentwicklung im Allgemeinen wurde von Gerhard Weiß [168] zusammengestellt.

2.4.1. JADE

Das wohl am weitesten verbreitete Agentenframework ist das *Java Agent Development Framework (JADE)* [5], das von der Telecom Italia entwickelt wurde.[5] Ein Ziel bei der Entwicklung von JADE war hierbei, konform mit den Spezifikationen der FIPA zu sein – man kann JADE guten Gewissens als die Referenzimplementierung der FIPA-Spezifikationen betrachten. Darüber hinaus sind verschiedene Werkzeuge zum Debuggen, Deployen und Überwachen von JADE-Agenten verfügbar.

Jeder JADE-Agent läuft in einem eigenen Thread und verfügt über seine eigene *message box*, über die er Nachrichten von anderen Agenten empfangen kann. Diese können wahlweise die FIPA *Agent Communication Language (ACL)* implementieren oder ein einfaches serialisiertes Java-Objekt sein. Das Verhalten des Agenten wird über sogenannte *behaviours* definiert, mit denen der Agent auf eintreffende Nachrichten oder Zustandsänderungen reagieren kann. Reaktives Verhalten und einfaches Reasoning wird über die Integration einer JESS-Engine erreicht (Java Expert System Shell). Mittels

[5] JADE: `http://jade.tilab.com`

WADE kann das Verhalten der Agenten auch in Form von Prozessen erstellt werden (siehe Abschnitt 4.3.1).

Durch die große Verbreitung in der Agenten-Gemeinschaft existieren auch zahlreiche Erweiterungen für JADE, beispielsweise auch für die Erweiterung von JADE um BDI-Konzepte [111]. Für die Beschreibung der Ziele, Pläne und Beliefs werden Java-Klassen genutzt, die entsprechende Interfaces implementieren. Die Pläne bestimmen dabei selbst, welche Ziele sie erfüllen können; eine Planung anhand von Vorbedingungen und Effekten findet nicht statt.

2.4.2. Jadex

Auch *Jadex* wurde als Erweiterung zu JADE begonnen [133], in der JADE um BDI-Konzepte ergänzt wird [16], kann heute aber als eigenständige Agenten-Sprache angesehen werden. Jadex-Agenten verfügen über explizites Wissen über ihr Wissen und ihre Aktionsmöglichkeiten und können mit diesen zielorientiert handeln und planen. Hierbei wird jedem Ziel ein oder mehrere Pläne zugeordnet, mit denen dieses Ziel erreicht werden kann, aus denen dann ein geeigneter ausgewählt wird; Planung anhand von Vorbedingungen und Effekten findet nicht statt.

Die Ziele, Pläne und Beliefs der Agenten wurden in frühen Versionen von Jadex in XML-Dateien beschrieben, die parallel zu den Implementierungen in Java gepflegt werden mussten [16]. In späteren Versionen wurden spezielle Java-Annotations eingeführt, mit denen die entsprechenden Klassen, Methoden und Attribute direkt im Java-Code als Ziele, Pläne und Beliefs gekennzeichnet werden können [131].

Mit *Jadex Active Components* [130] können Jadex-Agenten modular aufgebaut und an unterschiedliche Ausführungsumgebungen angepasst werden [132]. Zudem existiert eine breite Auswahl an Werkzeugen zum Erstellen und Überwachen von Jadex-Agenten, darunter auch ein visueller Editor für Ziele und Pläne (in Form von BPMN) und eine entsprechende Interpreter-Komponente (siehe auch *Go4Flex* im Abschnitt 4.2.3).

2.4.3. Jason und Agentspeak

Ein weiteres populäres Agentenframework ist *Jason* [13]. Dieses ist, wie JADE auch, in Java implementiert, basiert jedoch auf *AgentSpeak* [136], einer theoretisch fundierten, logikbasierten Beschreibungssprache für BDI-Agenten.

Die Pläne bzw. Aktionen des Agenten sind mit Precondition (*context*) und Effekt (*trigger-condition*) semantisch beschrieben. Wenn der Effekt eines Plans einem Ziel des

Agenten entspricht wird dieser zur Ausführung ausgewählt, sofern die Precondition erfüllt ist. Die Agenten können mittels SACI (Simple Agent Communication Infrastructure) miteinander kommunizieren; alternativ kann hierfür auch JADE eingesetzt werden. Auf diese Weise können die Agenten durch Sprechakte Annahmen (*beliefs*) austauschen, wobei der Agent zwischen selbst erworbenen und von anderen Agenten mitgeteilten *beliefs* unterscheidet.

2.4.4. JACK Intelligent Agents

Ebenfalls dem BDI-Paradigma folgt *JACK Intelligent Agents*, das wie JADE und Jason auch auf Java basiert, die Sprache hierbei aber um zusätzliche syntaktische Elemente erweitert [24]. Eine Reihe von Anwendungen, unter anderem im militärischen Bereich, werden auf der Webseite der Entwickler genannt.[6]

Neben *beliefs*, *desires* und *intentions* verfügen JACK-Agenten über sogenannte *capabilities* (Mengen von Aktionsmöglichkeiten und Verhaltensweisen, vergleichbar mit Rollen in anderen Systemen) und können auf *events* reagieren, die von der Umgebung erzeugt werden [23]. Im Rahmen ihres BDI-Zyklus wählen die Agenten aus verschiedenen Plänen, in denen unterschiedliche Aktionen aggregiert werden können. Diese haben eine recht einfache Struktur, können jedoch nicht von einem Planer automatisch erstellt werden. Für die Kommunikation wird Standard-Middleware wie CORBA eingesetzt, Unterstützung für komplexere Protokolle ist aber ebenso möglich.

Wie auch für JADE gibt es für JACK zahlreiche von der Community entwickelte Erweiterungen, beispielsweise *JACKTeams*, welches Konzepte wie Teams, Knowledge Sharing und Koordination mitbringt, oder *CoJACK*, welches den BDI-Zyklus des Agenten um zusätzliche kognitive Konzepte, wie etwa *situation awareness*, erweitert.

2.4.5. JIAC V

JIAC V (Java-based Intelligent Agent Componentware, Version 5) ist ein auf Java basierendes Multiagenten-Framework, das Agententechnologien mit Aspekten von serviceorientierten Architekturen (SOA) verbindet [67, 96, 98].

Die Agenten sind auf verschiedene Agentenknoten, oder *Nodes*, verteilt und können über Nachrichten untereinander kommunizieren und diese sowohl an individuelle Agenten als auch an Gruppenadressen schicken (Broadcast/Multicast). Die Fähigkeiten (Actions) eines Agenten können als Dienste allen anderen Agenten zugänglich gemacht

[6] JACK: `http://www.agent-software.com.au/`

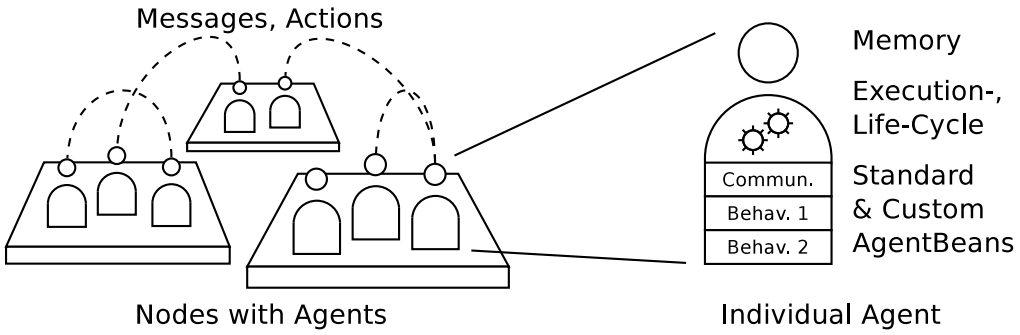


Abbildung 2.2.: Schematische Darstellung von JIAC-Nodes und Agenten. [89]

werden. Auch können zur Laufzeit weitere Dienste, Agenten oder auch Agentenknoten hinzugefügt oder entfernt werden, und JIAC-Agenten können von einem Agentenknoten zum anderen migrieren.

Die einzelnen Agenten, bzw. deren Verhalten und Fähigkeiten, setzen sich hierbei aus sogenannten *AgentBeans* zusammen, die über den Life-Cycle des Agenten kontrolliert werden. Der Zugriff auf bestimmte Kernkomponenten des Agenten, etwa den als *Tupel Space* [55] implementierten Speicher, ist hierbei allen Agenten Beans möglich. Über diese Komponenten lassen sich – neben domänen-spezifischen Funktionalitäten – auch abstrakte Eigenschaften wie Reaktivität oder Zielorientierung zum Agenten hinzufügen (siehe Abbildung 2.2). Die Knoten sowie die darauf befindlichen Agenten werden in XML-Dateien konfiguriert.

Die AgentBeans verfügen über verschiedene Mechanismen, mit denen sich das gewünschte Verhalten umsetzen lässt. Sie können etwa *life-cycle*-Methoden implementieren, die beispielsweise aufgerufen werden, sobald der Agent startet, eine *execute*-Methode bereit stellen, die in regelmäßigen Intervallen vom Execution-Cycle des Agenten aufgerufen wird, *Observer* an den Agenten-Speicher anhängen um auf Änderungen in der Faktenbasis zu reagieren, oder *Actions*, oder Services, bereit stellen, die von anderen Agenten oder anderen AgentBeans aufgerufen werden können.

Eine solche AgentBean, mit der jeder Agent standardmäßig ausgestattet ist, ist die *Communication Bean*, die es dem Agenten ermöglicht, Nachrichten zu verschicken und zu empfangen. Außerdem gibt es unter anderem eine *Rule Engine Bean*, mit der der Speicher des Agenten durch eine Drools-Regel-Engine[7] ausgewertet und aktualisiert werden kann, um so reaktives Verhalten herzustellen [68]. Die *JADL++ Interpreter Bean* erlaubt es dem Agenten, in einer speziellen Skriptsprache geschriebene Dienste anzubieten

[7]Drools: `http://www.drools.org/`

und auszuführen [66]. Die Sprache JADL++ (JIAC Action Description Language) bietet Sprachelemente, mit denen sich viele grundlegende Funktionen des Agenten sehr einfach nutzen und aggregieren lassen, und verfügt über semantische Dienstbeschreibungen, Event-Handling und einfache Parallelisierbarkeit von Dienstaufrufen.

2.5. Fazit

Agenten und Multiagenten-Systeme bieten viele Eigenschaften, die von einer heutigen serviceorientierten Architektur gefordert werden: Agenten können auf einfachste Weise miteinander kommunizieren und sich untereinander abstimmen, zielorientiert handeln und planen, dynamisch Dienste und Aktionen auswählen und selbst Dienste für andere Agenten und externe Systeme zur Verfügung stellen und vieles mehr – in Details natürlich abhängig von der jeweiligen Sprache. Während einige Agenten-Frameworks und agentenbasierte Anwendungen in der Industrie Fuß fassen konnten, ist deren Verbreitung allen Vorteilen zum Trotz nicht zu vergleichen mit der „klassischer“ SOA.

Somit werden Agenten und die sie umgebende Infrastruktur in erster Linie im akademischen Bereich weiterentwickelt. Bezogen auf die Entwicklungswerkzeuge wird von Bordini et al. [10] dabei festgestellt, dass noch nicht der Standard erreicht ist, den man etwa von Werkzeugen für objektorientierte Sprachen gewohnt ist, was mit der Inhomogenität der verschiedenen Agenten-Systeme und dem resultierenden Mangel an Austausch zu erklären sei, es gäbe jedoch einen Trend dahingehend, von der zugrunde liegenden Sprache zu abstrahieren und visuelle Programmierung einzusetzen.

Die Verwendung einer auf einem verwandten Gebiet bereits etablierten graphischen Notation wie der Geschäftsprozessmodellierung erscheint hier als ein sinnvoller nächster Schritt, spielen Agenten und Multiagenten-Systeme doch bereits in vielen Ansätzen zum Geschäftsprozessmanagement und in serviceorientierten Architekturen eine entscheidende Rolle. Im folgenden Kapitel soll daher ein Überblick über verschiedene Prozessnotationen gegeben und bestehende Ansätze betrachtet werden, in denen bereits eine Integration von Agenten und Prozessmodellierung angestrebt wird.

Kapitel 3.

Techniken der Prozessmodellierung

Die Anfänge der Prozessmodellierung in der Informatik gehen auf die erste Hälfte des vorigen Jahrhunderts zurück, mit dem Ziel eine formale Grundlage zu den bis dahin intuitiven und informellen Konzepten von Berechenbarkeit zu schaffen. Die ersten formalen Modelle, wie etwa das Lambda-Kalkül, waren noch auf die sequenzielle Abarbeitung der Arbeitsschritte ausgelegt. In späteren Ansätzen, wie CCS und CSP,[1] wurden darüber hinaus Nebenläufigkeit und Kommunikation als grundlegende Konzepte aufgenommen [151].

Die in mathematischen Formeln beschriebenen Prozesskalküle wurden bald durch Graphnotationen ergänzt, etwa Petri-Netze, und fanden jenseits der theoretischen Informatik weitere Verbreitung, beispielsweise als Aktivitätsdiagramme im Rahmen der Unified Modelling Notation (UML). Schließlich fand die Prozessmodellierung ihren Weg in die Wirtschaftsinformatik, wo schon seit längerem formale und semi-formale graphische Notationen für die Beschreibung von Unternehmensabläufen genutzt wurden, und heute spezialisierte Notationen wie die Business Process Modeling Notation (BPMN) zusammen mit Konzepten der SOA einen wichtigen Bestandteil moderner Geschäftsprozessmanagementsysteme (Business Process Management Systems, BPMS) darstellen [151].

Die Konzepte *Nebenläufigkeit* und *Kommunikation* sind auch in den heutigen Geschäftsprozessnotationen noch vertreten – in einigen mehr, in anderen weniger explizit – und gerade sie sind es, die Geschäftsprozesse als Ansatz für die Modellierung von Multi-Agentensystemen interessant machen, stellen sie doch auch einen elementaren Aspekt der agentenorientierten Technologien dar.

In diesem Kapitel werden zunächst verschiedene Prozessnotationen betrachtet und unter mehreren Gesichtspunkten, wie etwa der Ausdrucksfähigkeit und der Verständlichkeit, miteinander verglichen. Anschließend wird ein kurzer Überblick über die Rolle der Prozessmodellierung in der Softwareentwicklung gegeben. Hierbei werden insbeson-

[1] CCS: Calculus of Communicating Systems; CSP: Communicating Sequential Processes

dere die strukturellen Unterschiede zwischen Prozessdiagrammen und „normalen“ Programmen betrachtet, sowie die Probleme und Möglichkeiten, die damit einher gehen. Während einige dieser Notationen ihren Ursprung im Geschäftsprozessmanagement haben, liegt der Fokus in diesem Kapitel auf ihrer Anwendung in der Softwareentwicklung. Ferner konzentriert sich die Betrachtung auf graphische Notationen.

Das Ziel dieses Kapitels liegt zum einen in der Wahl einer geeigneten Notation für die Verwendung in dieser Arbeit, und zum anderen darin, prinzipielle Stärken und Schwächen der Prozessmodellierung im Bereich der Softwareentwicklung zu beleuchten.

3.1. Prozessnotationen

Prozesse werden visuell meist mit einem sehr ähnlichen Set aus Knoten und Kanten modelliert, so dass die einzelnen Modelle einen hohen Wiedererkennungswert haben und intuitiv zu verstehen sind. Neben zahlreichen sehr einfachen Varianten gibt es auch Prozessnotationen, die durch zusätzliche Attribute genug Ausdruckskraft besitzen, sodass daraus ausführbarer Code generiert werden kann.

Ein bekannter Vertreter der Prozessdiagramme ist auch in der UML enthalten: Das Aktivitätsdiagramm. Andere Notationen sind die Business Process Modeling Notation (BPMN), die mathematisch und theoretisch gut fundierten Petri-Netze sowie eine große Zahl weiterer standardisierter und auch proprietärer Notationen.

In diesem Abschnitt soll ein kurzer Überblick über verschiedene Vertreter der Prozesssprachen gegeben werden. Hierbei werden in erster Linie graphische Prozessnotationen betrachtet, aber auch einzelne textuelle Formate. Anschließend wird auf einzelne Sprachen genauer eingegangen und ein Vergleich angestellt, dessen Ergebnis zu der in dieser Arbeit verwendeten Notation führen wird.

3.1.1. UML Aktivitäts- und Sequenz-Diagramme

Ein Teil der UML – die Aktivitätsdiagramme – befasst sich ebenfalls mit der Modellierung von Prozessen [119, Kapiel 12]. Da die Softwareentwicklung bereits häufig von der Erstellung von UML-Diagrammen begleitet wird, erscheint es naheliegend, für die Prozessmodellierung auf UML Aktivitätsdiagramme zurückzugreifen.

Die Basiselemente von Aktivitätsdiagrammen – Aktivitäten, Kontrollfuss, Verzweigungen, Nebenläufigkeit und Kommunikation – sind denen der im folgenden Abschnitt vorgestellten BPMN sehr ähnlich und einfach zu verstehen. Die einzelnen Elemente sind

klar gezeichnet und gut zu unterscheiden, jedoch sind gerade die weniger gebräuchlichen Elemente, etwa für Timer und Ausnahmen, nicht immer intuitiv verständlich. Zugleich stellen Aktivitätsdiagrammen nur einen kleinen Teil der UML dar, und ihnen fehlen die Informationen der anderen Diagramme. Für ein vollständiges Bild eines Programms reichen Aktivitätsdiagramme somit nicht aus.

Interaktionen zwischen verschiedenen Systemkomponenten können mit Sequenzdiagrammen modelliert werden, wobei jeder Akteur durch eine vertikale Lebenslinie dargestellt wird, zwischen denen Nachrichten ausgetauscht werden. Ferner können Abschnitte der Interaktion in Schleifen oder nur unter bestimmten Bedingungen ausgeführt werden; die Möglichkeiten, den Kontrollfluss darzustellen, sind jedoch begrenzt, und immer auf die Interaktion bezogen.

Es gibt Werkzeuge, mit denen aus bestimmten UML-Diagrammtypen Quellcode-Teile generiert werden können, jedoch in der Regel keine vollständigen, ausführbaren Programme. Der Fokus der UML liegt auf der Modellierung.

3.1.2. Business Process Model and Notation

Business Process Model and Notation (BPMN, vormals auch Business Process Modeling Notation, aktuell in Version 2.0.2) [118] ist ein relativ neuer Standard für eine graphische Prozessnotation, der zunehmend an Bedeutung gewinnt [137]. Er wurde mit dem Ziel entworfen, zum einen für alle am Entwicklungsprozess beteiligten Personen verständlich zu sein, zum anderen aber auch eine Basis für die Implementierung des modellierten Prozesses darzustellen. Eine kurze Einführung in BPMN kann unter anderem in [169] gefunden werden.

Eine Besonderheit von BPMN, durch die sie sich von vielen anderen Notationen abhebt, ist, dass sowohl das Verhalten der verschiedenen am Prozess beteiligte Akteure, als auch deren Interaktionen und der zwischen ihnen stattfindende Nachrichtenaustausch explizit in einem Diagramm dargestellt werden. Somit vereint BPMN Aspekte von UML Aktivitätsdiagrammen und Sequenzdiagrammen in einer Notation.

BPMN-Diagramme verfügen hierbei über eine breite Palette von Elementen, die sie für die Modellierung von verteilten Systemen, und insbesondere auch Multiagenten-Systemen, interessant machen. So sind die Prozessdiagramme in Pools unterteilt, die jeweils einen anderen Geschäftspartner, oder Akteur im Prozess darstellen, wobei explizit zwischen dem Kontrollfluss innerhalb eines Pools (Sequence Flow) und dem Nachrichtenaustausch zwischen Pools (Message Flow) unterschieden wird. Abbildung 3.1 zeigt ein einfaches Beispieldiagramm. Des Weiteren verfügt die Notation über Elemente zum

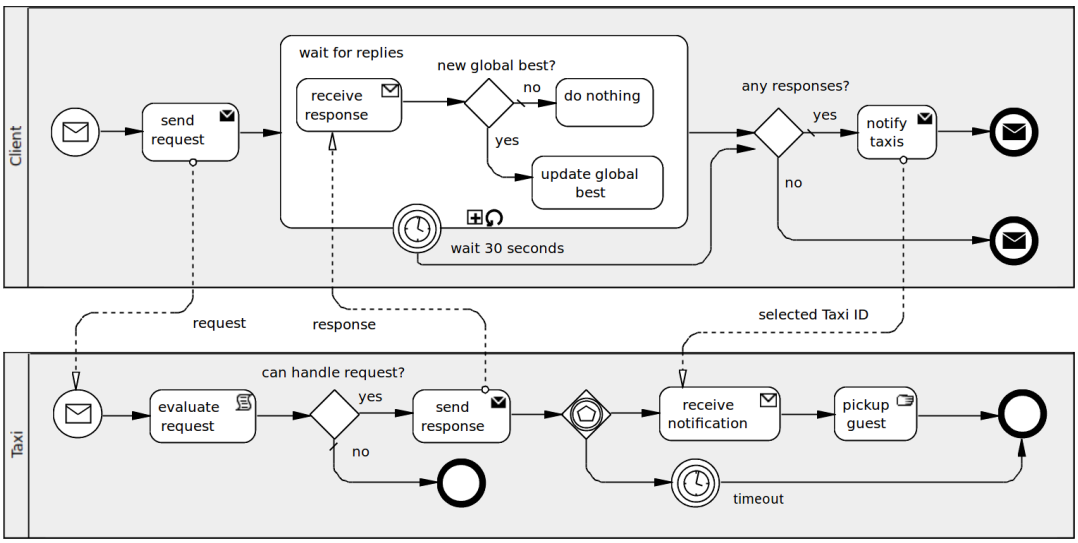


Abbildung 3.1.: BPMN-Beispieldiagramm „Taxi Rufen“ [89], mit Pools, Kontrollfluss, Nachrichten, und Event-Handling.

Modellieren von Ereignis- und Fehlerbehandlung, sowie zur Beschreibung von Transaktionen und Ad-Hoc-Verhalten.

Die Prozessdiagramme können dabei auf drei Abstraktionsebenen verstanden werden:

1. Die oberste Ebene stellt die verschiedenen Typen der sogenannten Flow Objects dar: Die Activities, Events und Gateways. Diese können ohne besondere Kenntnis der Sprache als solche erkannt und aufgrund ihrer Form besonders einfach unterschieden werden.

2. Diese Basiselemente lassen sich in mehrere Subtypen unterscheiden, beispielsweise Service Tasks und Timer Events, welche die gleiche Form wie ihre Basistypen haben, zudem aber ein einprägsames und intuitiv verständliches Symbol tragen.

3. Zuletzt verfügen alle Elemente über zahlreiche Attribute, die der Übersicht halber nicht in den Diagrammen dargestellt werden. Diese tragen zusätzliche Informationen, die für die Erzeugung von Programmcode für ausführbare Sprachen benötigt werden.

Somit kann BPMN zugleich auf intuitive Weise verstanden werden, etwa von Domänenexperten, und als Grundlage für die Implementierung des Prozesses in einem Softwaresystem dienen – eine entsprechende Abbildung von BPMN auf die Business Process Execution Language (BPEL) ist seit der ersten Version in der Spezifikation enthalten [118,

Kapitel 14]. Insbesondere mit Version 2 der Notation entstanden zudem verschiedene BPMN-Interpreter, die direkt auf BPMN arbeiten.

Da der Schwerpunkt der Spezifikation auf der visuellen Repräsentation liegt, sind die Beschreibungen der Attribute jedoch eher vage ausgefallen. Die Bedeutung zahlreicher Attribute und Prozessstrukturen ergibt sich nur indirekt aus der in der Spezifikation definierten Abbildung nach BPEL [118, Kapitel 14]; eine formale Spezifikation der Semantik von BPMN wird nicht gegeben. Es gibt jedoch einige Bemühungen, eine Semantik für BPMN zu definieren. Dijkman et al. [42, 43] haben eine partielle Abbildung von BPMN auf Petri-Netze entwickelt, die es ermöglicht, BPMN-Diagramme mit Petri-Netz-Werkzeugen zu validieren. Sie weisen in ihrer Arbeit aber auch auf Schwachstellen und Mehrdeutigkeiten in der Spezifikation hin, die einer vollständigen Abbildung von BPMN auf Petri-Netze bislang im Weg standen. Eine ähnliche Abbildung wird auch von Raedts et al. vorgeschlagen [135]. Wong und Gibbons [176] verfolgen einen Ansatz zur Formalisierung von BPMN mittels Communicating Sequential Processes (CSP). In Version 2.0 der BPMN-Spezifikation [118] werden, neben zahlreichen Erweiterungen der Notation, auch viele Aspekte der Semantik genauer beschrieben sowie ein einheitliches Austauschformat definiert.

3.1.3. Petri-Netze

Im starken Kontrast zur BPMN, sind Petri-Netze [126] eine vergleichsweise einfache Notation für nebenläufige Prozesse, die sich jedoch durch zahlreiche mathematische Methoden zur Analyse und Validierung auszeichnet, etwa um Eigenschaften wie Lebendigkeit und Erreichbarkeit zu bestimmen. Die Anwendungsgebiete von Petri-Netzen sind dabei weit gefächert [108]: Sie werden sowohl dazu eingesetzt, um systemnahe Abläufe zu untersuchen, können aber auch für „high-level“-Prozesse eingesetzt werden.

Die Notation ist dabei sehr abstrakt: So bestehen die Graphen aus nur zwei Arten von Elementen – Stellen und Transitionen – die durch (gewichtete) Kanten miteinander verbunden sind. Es gibt keine explizite Unterscheidung zwischen verschiedenen Typen von Ereignissen und Aktionen, zwischen der sequenziellen Abarbeitung von Prozessschritten und der Kommunikation zwischen Prozessen, oder zwischen am Prozess beteiligten Akteuren.

Während der bestechend einfache Aufbau die Analyse der Petri-Netze vereinfacht, werden die einzelnen Prozessmodelle, speziell wenn Konzepte wie Fehlerbehandlung umgesetzt werden sollen, schnell sehr komplex, und größere Zusammenhänge sind selbst für Experten nicht immer leicht zu durchschauen. Wenngleich Petri-Netze als primäre

Prozessnotation weniger geeignet erscheinen, können sie an anderer Stelle doch genutzt werden, etwa zur Beschreibung der Semantik und der Analyse und Verifikation der eigentlichen, für den Benutzer sichtbaren, Prozessdiagramme.

3.1.4. YAWL

YAWL („Yet Another Workflow Language“) wurde nach der Evaluation bestehender Workflow-Management-Systeme entwickelt [158, 160]. Die Sprache ist die Referenzimplementierung der dabei gesammelten *Workflow Patterns*, auf die in Abschnitt 3.2 noch näher eingegangen wird.

Die Motivation bestand darin, dass die meisten der untersuchten Prozesssprachen (darunter auch viele proprietäre) keine klar definierte Semantik hatten, wie bei der Evaluation anhand der Workflow Patterns ersichtlich wurde. Petri-Netze, auf der anderen Seite, haben zwar eine klar definiert Semantik, erwiesen sich bei einigen der Patterns jedoch ebenfalls als problematisch, insbesondere bei nicht-lokaler Semantik, etwa beim Cancellation Pattern, das nur sehr umständlich als Petri-Netz realisiert werden kann. YAWL basiert auf Petri-Netzen und erbt deren klar definierte Semantik, bringt aber einige Erweiterungen mit, etwa Composite-Tasks, dedizierte Split- und Join-Knoten (AND, OR, und XOR), sowie einen Cancellation Task, durch den sämtliche Tokens von allen assoziierten Aktivitäten entfernt werden.

Die Notation, insbesondere die unterschiedlichen Symbole für AND, OR und XOR, sind jedoch wenig intuitiv, und während YAWL von einigen Unternehmen im Produktivbetrieb eingesetzt wird, ist die Verbreitung nicht zu vergleichen mit der von BPMN, EPK oder UML. Die größte Verbreitung findet die Sprache im akademischen Bereich.

3.1.5. Ereignisgesteuerte Prozessketten

Eine weitere Prozessnotation sind die Ereignisgesteuerten Prozessketten (EPK; engl. Event-driven Process Chains, EPC) [79], wie sie im ARIS-Konzept[2] verwendet werden. Im Wesentlichen bestehen EPKs aus alternierenden Folgen von Ereignissen und Funktionen und sind den Petri-Netzen in dieser Hinsicht nicht unähnlich [157]. Darüber hinaus verfügen sie über einige weitere Elemente, über die beispielsweise Organisationseinheiten und Ressourcen in die Diagramme aufgenommen werden können. Wie bei Petri-Netzen wird die Bedeutung eines Knotens jedoch nur durch seine Beschriftung und ggf. den Kon-

[2] ARIS ist ein Produkt der Software AG: `http://www.softwareag.com`

text, nicht jedoch durch seinen Typen deutlich; das gleiche gilt für die Unterscheidung von Kontrollfluss innerhalb von und zwischen Prozessen.

In der Praxis sind EPKs ähnlich verbreitet wie BPMN, was sich nicht zuletzt mit der einfachen und leicht erlernbaren Syntax erklären lässt. Durch die geringe Zahl an unterschiedlichen Elementen sind sie für die Modellierung von Prozessen, die in ausführbare Programme übersetzt werden sollen, jedoch weniger geeignet. Zwar ist es möglich, BPEL-Prozesse aus EPKs abzuleiten [185], hierbei kann jedoch nur der Kontrollfluss generiert werden, doch keine Zuweisungen oder Dienstaufrufe.

3.1.6. Business Process Execution Language

Im Gegensatz zu den anderen hier besprochenen Prozess-Sprachen ist die Business Process Execution Language (BPEL) [112] keine Prozessnotation, sondern eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen. Diese werden von BPEL-Engines interpretiert und dazu verwendet, bestehende Webservices zu komplexeren Prozessen zu orchestrieren. In der Spezifikation wird keine visuelle Notation angegeben – diese variiert unter den Modellierungswerkzeugen. Hierbei wird immer häufiger BPMN oder eine vereinfachte, an BPMN angelehnte Notation für die Visualisierung von BPEL-Prozessen verwendet. BPMN und BPEL sind hierbei jedoch nicht isomorph: Sowohl in Hinblick auf die individuellen Diagrammelemente als auch bezüglich der möglichen Prozessstrukturen bietet BPMN sehr viel mehr Möglichkeiten als BPEL.

3.1.7. XPDL

Die XML Process Definition Language (XPDL) ist ein Interchange-Format und verfügt wie BPEL über keine eigene graphische Notation [180]. Die XPDL hat mit einer minimalen Menge an Basiselementen begonnen und wurde im Folgenden um zahlreiche zusätzliche Elemente erweitert einschließlich eines großen Teils der BPMN; mit BPMN 2.0 und dem darin definierten XML-Schema wurde XPDL als Austauschformat für BPMN-Diagramme jedoch überflüssig.

3.2. Vergleich der Prozessnotationen

In den folgenden Abschnitten sollen die verschiedenen Prozessnotationen miteinander verglichen werden, um sich für eine in dieser Arbeit zu verwendende Notation zu ent-

scheiden. Hierfür soll zunächst ein Blick auf die Workflow Patterns geworfen werden, bevor die aussichtsreichsten Kandidaten genauer betrachtet werden.

3.2.1. Workflow Patterns

Um einen objektiven Vergleich der Ausdrucksmöglichkeiten verschiedener Prozessbeschreibungssprachen zu ermöglichen, haben Wil van der Aalst und Arthur ter Hofstede [142, 161] verschiedene *Workflow Patterns* entwickelt, vergleichbar mit den Design Patterns der objektorientierten Programmierung [54]. Sie haben sich seitdem zu einem oft genutzten Werkzeug zur Evaluation verschiedener Prozessbeschreibungssprachen und Ausführungsumgebungen entwickelt. Neben einfachen Kontrollfluss-Patterns wie Sequence, Parallel Split und Exclusive Choice werden auch komplexere Strukturen sowie die Behandlung von Daten, Ressourcen und Exceptions behandelt. Tabellen mit den Evaluationsergebnissen verschiedener Notationen finden sich auf der Webseite des Projekts.[3]

White, einer der Entwickler der BPMN, nutzte die Workflow Patterns, um BPMN mit Aktivitätsdiagrammen der UML zu vergleichen [170]. Sein Fazit war, dass während beide Notationen sowohl in ihrer Präsentation als auch in ihrer Mächtigkeit recht ähnlich sind, BPMN durch die Verwendung von nur wenigen Grundformen und Variationen derselben einfacher zugänglich ist. Seine Analyse beschränkte sich hierbei auf die Control Flow Patterns. Später wurde die Untersuchung von Wohed et al. unter Einbeziehung der übrigen Patterns wiederholt [173, 174]. Ihr Ergebnis war, dass während die meisten Control Flow Patterns mit BPMN modelliert werden können, nur wenige Data Flow Patterns und kaum Resource Patterns unterstützt werden. Das gleiche Bild zeigte sich auch bei der Untersuchung von UML Aktivitätsdiagrammen [143, 175].

Ähnliche Untersuchungen gab es auch für EPKs [104]. Wie die Tabellen zeigen, bieten EPKs nur sehr eingeschränkte Unterstützung für die Workflow Patterns. Etwas besser sieht die Unterstützung bei der Erweiterung yEPC aus. Die Entwicklung der Workflownotation YAWL [160] wurde maßgeblich von den Workflow Patterns beeinflusst; sie stellt gewissermaßen die Referenzimplementierung der Workflow Patterns dar und deckt dementsprechend viele der Patterns ab – jedoch auch nicht alle. XPDL zeigt erwartungsgemäß eine ähnliche Abdeckung wie BPMN, ist in einzelnen Aspekten jedoch auch schwächer.

[3] Workflow Patterns: `http://www.workflowpatterns.com/`

Tabelle 3.1.: Unterstützung von Workflow Patterns, nach [142, 160]

	Pattern	**UML-AD**	**BPMN**	**YAWL**	**EPK**	**XPDL**
1	Sequence	+	+	+	+	+
2	Parallel Split	+	+	+	+	+
3	Synchronisation	+	+	+	+	+
4	Exclusive Choice	+	+	+	+	+
5	Simple Merge	+	+	+	+	+
6	Multiple Choice	+	+	+	+	+
7	Synch. Merge	-	+	+	+	+
8	Multiple Merge	+	+	+	-	+
9	Discriminator	+/-	+/-	+	-	+/-
10	Arbitrary Cycle	+	+	+	+	+
11	Impl. Termin.	+	+	-	+	+
12	MI w/o synch.	+	+	+	-	+
13	MI w/ dsgn-tm knowl.	+	+	+	-	+
14	MI w/ runtime knowl.	+	+	+	-	+
15	MI w/o runtime knowl.	-	-	+	-	-
16	Deferred Choice	+	+	+	-	+
17	Interl. Parallel Rtng.	-	-	+	-	-
18	Milestone	-	-	+	-	-
19	Cancel Activity	+	+	+	-	+
20	Cancel Case	+	+	+	-	+

Tabelle 3.1 zeigt eine Zusammenfassung der Ergebnisse für die hier betrachteten Notationen.[4] BPMN, UML Aktivitätsdiagramme und YAWL zeigen die beste Unterstützung für die Workflow Patterns. Betrachtet man die ausführlicheren Tabellen auf der Projektwebseite,[3] so sieht man, dass die Notationen typischerweise bei den Kontrollfluss-Patterns, und die Ausführungsumgebungen, wie etwa BPEL, bei den Data- und Resource-Patterns besser abschneiden. Dies ist ein generelles Problem der Prozessnotationen. Recker und Mendling sprechen gar von einem „Conceptual Mismatch“ [138] zwischen der Entwicklung von Prozessbeschreibungssprachen und deren Ausführungsumgebungen. Durch die Abbildung auf BPEL und die dafür in die Spezifikation aufgenommenen Elemente wie Variablen, Zuweisungen und Dienstebeschreibungen, stellt BPMN in dieser Hinsicht einen Kompromiss zwischen reinen Geschäftsprozessbeschreibungssprachen

[4] Je nach Quelle kann die ausgewiesene Unterstützung variieren. Die Werte für UML, BPMN, EPK und XPDL stammen aus [142] und für YAWL aus [160].

wie etwa EPKs und ausführbaren Sprachen wie BPEL dar.

3.2.2. Weitere Kriterien

Während die Workflow Patterns eine gute Orientierung zur objektiven Bewertung der Prozessbeschreibungssprachen bieten, gibt es auch andere Faktoren, die die Wahl einer Prozessnotation mit bestimmen. So soll in dieser Arbeit eine Notation verwendet werden, die auch in der Praxis Anwendung findet – andernfalls könnte man auch eine weitere Notation speziell für die Beschreibung von Agentensystemen mit Prozessen entwickeln, der Nutzen wäre jedoch gering. In jedem Fall wird es notwendig oder zumindest sinnvoll sein, die Notation in mancher Hinsicht zu erweitern, etwa um besondere Eigenschaften der Agentensysteme modellieren zu können, wie etwa Autonomie und Planung, oder um die automatische Erzeugung vollständiger, lauffähiger Systeme zu gewährleisten. Dennoch soll als Basis eine Prozessnotation verwendet werden, die sich in der Praxis etabliert und bewährt hat.

In einer Untersuchung von Figl et al. [51] wird, basierend auf Prinzipien aus der Kognitionsforschung wie Eindeutigkeit, Unterscheidbarkeit, Ausdruckskraft, Verständlichkeit und Einfachheit, die Usability (Benutzbarkeit) verschiedener Notationen verglichen. BPMN besticht hierbei, im Gegensatz zu YAWL, durch seine intuitive Notation, wird jedoch auch für die überladene Symbolpalette kritisiert; EPKs und Petri-Netze wiederum leiden an zu geringer Vielfältigkeit, sodass dieselben Symbole für zu viele verschiedene Konzepte verwendet werden müssen. UML Aktivitätsdiagramme schneiden in alles Aspekten gut ab. In der Studie wurde allerdings nur das *common subset* der verschiedenen Notationen betrachtet, sodass viele Stärken von BPMN, wie etwa der Nachrichtenaustausch zwischen Pools oder das Event Handling, nicht berücksichtigt wurden.

Von den beschriebenen Prozessnotationen erscheinen BPMN und UML Aktivitätsdiagramme als die geeignetsten Alternativen. Beide Notationen sind standardisiert und verfügen, anders als YAWL, über eine breite Anwenderbasis. Vor allem aber sind die verwendeten Metamodelle nicht zu abstrakt: Verglichen etwa mit der ebenfalls weit verbreiteten EPK-Notation verwenden Aktivitätsdiagramme und vor allem die BPMN eine deutlich größere Palette an unterscheidbaren Modellelementen, was zwar eine längere Einarbeitungszeit erfordert, zugleich aber die Erstellung von sehr viel differenzierteren und ausdrucksstärkeren Prozessmodellen ermöglicht. So gibt es etwa in beiden dedizierte Elemente für das Senden und Empfangen von Nachrichten oder Signalen oder für zeitgesteuerte Ereignisse, während solche Ereignisse in einem EPK-Diagramm nur eine entsprechende Bezeichnung hätten.

3.2.3. Ein genauerer Blick auf UML und BPMN

Die UML ist ein sehr umfassender Standard mit vielen verschiedenen Diagrammen. Jedes der Diagramme hat eigene Regeln und eine eigene Notation, in der Elemente wie Pfeile, Kästen und Ellipsen in einem bestimmten Stil eine andere Bedeutung haben können. Zur vollständigen Beschreibung eines Softwaresystems werden mehrere Diagramme benötigt: So wird etwa das Datenmodell mit Klassendiagrammen beschrieben, die Architektur über Komponentendiagramme und die Kommunikation der einzelnen Teile über Sequenzdiagramme. Neben der eigentlichen Geschäftslogik werden hierbei auch sehr viele Implementierungsdetails modelliert. So wird in den Klassendiagrammen nicht nur das Domänenmodell, sondern ebenso auch die verschiedenen Werkzeug- und Hilfsklassen bis hin zu Aspekten der Benutzerschnittstelle spezifiziert. Entsprechend schwer ist es für die eigentlichen Domänenexperten, ein Geschäftsmodell in UML zu modellieren.

Die Spezifikation von BPMN ist ebenfalls sehr umfangreich, jedoch gibt es nur einen wesentlichen Diagrammtypen: das Prozessdiagramm. Diese sind deutlich komplexer als UML Aktivitätsdiagramme und können als eine Kombination aus Aktivitätsdiagramm und Sequenzdiagramm angesehen werden, in der zwischen internen Abläufen und der Kommunikation mit externen Diensten und Partnern unterschieden wird. Wie weiter oben beschrieben, können die Prozessdiagramme in BPMN zudem mit verschiedenen Filtern betrachtet werden: Man kann sie auf eine vereinfachte Teilmenge der BPMN reduzieren oder mit allen Details darstellen, oder man kann nur die Public Processes einzelner Teilnehmer betrachten, also nur solche Aktivitäten, die für die Kommunikation relevant sind. In jedem Fall bleiben dies jedoch Ansichten desselben Modells.

Mit BPMN 2.0 kamen zwei weitere Diagrammtypen hinzu: Das Choreography Diagram und das Conversation Diagram. Das Conversation Diagram ist hierbei vergleichbar mit UML Usecase Diagrams. In ihm werden die verschiedenen Geschäftsprozesse, zu einzelnen Knoten reduziert, mit den zusammenwirkenden Teilnehmern in Beziehung gesetzt. Das Choreography Diagram stellt eine weitere alternative Ansicht auf die Prozesse dar, in welcher der Schwerpunkt auf den Nachrichtenaustausch und das Zusammenspiel der Akteure gelegt wird; dargestellt als Prozess „zwischen“ den Prozessen. Beide sind nur weitere Ansichten auf einzelne oder zusammengehörige Prozesse, stellen aber keine zusätzlichen Informationen dar, die in den eigentlichen Prozessdiagrammen fehlen würden [2].

Während BPMN, was die Prozessbeschreibung betrifft, deutlich über die Möglichkeiten von UML Aktivitätsdiagrammen hinaus geht, fehlen andere Aspekte der UML. So ist die Modellierung von Datentypen in BPMN nicht vorgesehen; es können nur beste-

hende Datentypen namentlich referenziert und verwendet werden. Die Modellierung von Implementierungsdetails fehlt völlig; viel mehr beschränkt sich die BPMN allein auf die Modellierung des Geschäftsmodells. Die ist nicht unbedingt ein Nachteil. Die Idee – nicht nur in dieser Arbeit – ist, dass die Implementierung aus den BPMN-Modellen generiert wird. Dies kann eine einfache Komposition von Diensten sein, oder auch ein vollständiges, verteiltes Softwaresystem, das sich entsprechend der spezifizierten Geschäftslogik verhält.

3.3. Prozessmodellierung in der Softwareentwicklung

Prozessdiagramme stellen eine sehr abstrakte Sicht auf ein System dar. Im Gegensatz beispielsweise zur UML wird nicht die komplette Architektur des Systems dargestellt, mit allen Klassen und deren Attributen. Stattdessen beschränkt sich die Modellierung auf das Wesentliche: die Abläufe, den *Workflow*. Auf diese Weise können insbesondere Verzweigungen, Parallelität, aber auch der Austausch von Nachrichten zwischen den einzelnen Akteuren sehr anschaulich dargestellt werden. Implementierungsdetails werden hierbei in der Regel außen vor gelassen, oder in einzelnen Aktivitäten oder Services gekapselt; andernfalls würden die Diagramme schnell sehr groß werden und der Vorteil der übersichtlichen Darstellung verloren gehen.

Während die UML unter Entwicklern sehr bekannt ist und mit ihrer Vielzahl verschiedener Diagramme mehr Aspekte der Softwareentwicklung abdeckt, als Prozessdiagramme allein, sind diese besonders intuitiv zu verstehen – auch für Geschäftsleute und Domänenexperten, die mit den Abläufen vertraut sind, sich jedoch nicht mit dem zugrunde liegenden System auskennen.

Das Hauptanwendungsgebiet der Geschäftsprozessnotationen ist die Modellierung von realen Abläufen: manuell ausgeführten Tätigkeiten und Interaktionen von verschiedenen Menschen und Abteilungen. Sie werden aber in zunehmendem Maße auch für die Modellierung von Computerprogrammen und die automatische Erzeugung von Programmcode genutzt [137]. Natürlich lässt sich ein System mit dieser Technik weniger detailliert beschreiben, als mit der Ausdruckskraft der UML. Das Gebiet, auf dem die Prozessmodellierung eingesetzt wird, ist daher in erster Linie die Orchestrierung bestehender Dienste, wie es etwa in der serviceorientierten Architektur der Fall ist [177].

3.3.1. Prozesse und Programme: Strukturelle Unterschiede

Ein Problem bei der Übersetzung von Prozessmodellen in Programmcode ist die unterschiedliche Struktur der Quell- und Zielmodelle: Während die gebräuchlichen Programmiersprachen eine *Blockstruktur* haben, haben Prozessdiagramme eine freiere *Graphstruktur* und dadurch eine größere Ausdrucksfähigkeit, bieten jedoch auch zusätzliche Fehlerquellen, etwa die Gefahr von Deadlocks (Verklemmungen). Durch ihre unterschiedliche Struktur ist es mitunter schwer, Prozessmodelle auf ausführbare Programme abzubilden.

Heutzutage ist die strukturierte Programmierung das vorherrschende Paradigma in den meisten geläufigen Programmiersprachen. Zu den ursprünglichen Strukturen – der Sequenz, der Auswahl (`if/else`, `switch/case`), und der Wiederholung (`while`, `repeat`) – kamen mit der Zeit weitere hinzu, etwa zur Fehlerbehandlung (`try/catch`) sowie Subroutinen. In jedem Fall aber ist allen Blockstrukturen gemein, dass sie über einen eindeutigen Eintrittspunkt und einen eindeutigen Austrittspunkt verfügen, und im Kontext des sie umgebenden Codes somit eine abgeschlossene Einheit bilden. Anders bei der Programmierung mit `GOTO`-Statements, die lange Zeit in Konkurrenz zur Strukturierten Programmierung stand und heute fast nur noch auf Spezialgebieten eingesetzt wird: Hier kann der Kontrollfluss jederzeit an praktisch jede beliebige andere Stelle des Programmcodes umgeleitet werden.

Prozessdiagramme ähneln in dieser Hinsicht den Sprachen mit `GOTO`-Statement: Auch hier kann von jeder Aktivität eine Kante zu jeder anderen Aktivität gezogen werden,[5] und nur ein Workflow, der auf bestimmte Weise strukturiert ist, kann auch direkt auf eine blockstrukturierte Sprache abgebildet werden. Ein einfache induktive Definition dessen, was ein Strukturiertes Workflow-Modell (SWM) ist, wird u. a. von Kiepuszewski et al. gegeben [80, Definition 2.2, S. 4]. Demnach sind (a) einzelne Aktivitäten, (b) Sequenzen von SWMs, (c) Verzweigungen von SWMs, und (d) Wiederholungen von SWMs wiederum SWMs (siehe Abbildung 3.2). Je nach Notation können auch weitere Strukturen hinzukommen.

Hieraus ergeben sich mehrere Herausforderungen bei der Modellierung von Prozess bzw. Workflow-Modellen zum Zweck der Programmcodeerzeugung: (a) Die Validierung, ob ein gegebener Workflow bestimmten strukturellen Anforderungen entspricht, etwa Deadlock-Freiheit, (b) die Identifizierung von SWMs innerhalb des Workflows, und die Übersetzung derselben in die entsprechenden Programmstrukturen, und (c) die Restruk-

[5] Tatsächlich gibt es häufig die Einschränkung, dass die Kanten bestimmte Grenzen nicht überbrücken dürfen; dies ändert jedoch nichts grundlegendes.

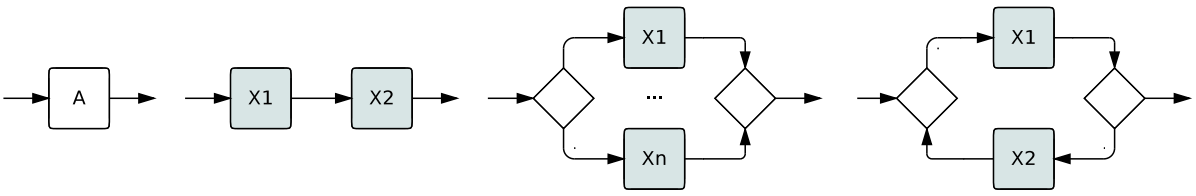


Abbildung 3.2.: Strukturierte Workflow-Modelle (SWMs): einzelne Aktivität, Sequenz, Block, Schleife. Die grau hinterlegten Knoten stehen für Abschnitte des Workflows, die bereits als SWM identifiziert wurden. [86]

turierung von Workflow-Modellen, die keine prinzipiellen Probleme haben, wie etwa Deadlocks, sich aber auch nicht direkt auf strukturierten Programmcode übersetzen lassen.

3.3.2. Validierung

Van der Aalst [156] hat gezeigt, wie verschiedene typische Prozess-Konstrukte mit Petri-Netzen beschrieben werden können und unterscheidet und formalisiert verschiedene Stufen von Strukturiertheit und Wohlgeformtheit (Soundness). Hierzu verwendet er eine erweiterte Petri-Netz-Notation, in der Transitionen zusätzlich einen „Trigger“ haben können, etwa Zeit oder Benutzerinteraktion, von der sich aber ohne Weiteres auch mit klassischen Petri-Netzen abstrahieren lässt.

Auch Liu und Kumar [95] haben verschiedene unstrukturierte Workflows dahingehend untersucht, inwieweit diese sich dennoch korrekt verhalten, z. B. keine Deadlocks aufweisen, und ob sie in eine strukturierte Form überführt werden können. Eine Beobachtung hierbei ist, dass jeder Workflow ohne parallele Ausführungszweige (Nebenläufigkeit) strukturiert werden kann, wohingegen die Einführung von Parallelität sowohl Deadlocks (bspw. XOR-Branch gefolgt von AND-Joins) als auch Synchronisierungsprobleme und Lifelocks (bspw. AND-Splits gefolgt von XOR-Merges) verursachen kann, und dass es Workflows mit Parallelität gibt, die sich grundsätzlich nicht in strukturierte Form bringen lassen.

Um die Strukturiertheit von Prozessgraphen zu untersuchen verwenden Sadiq und Orlowska [145, 146] eine Menge von Regeln. Diese werden iterativ angewendet und ersetzen jeweils einen wohlstrukturierten Subgraphen des Workflows durch einen einzelnen Knoten. Kann der Workflow auf diese Weise zu einem einzelnen Knoten reduziert werden, so ist er wohlstrukturiert und kann in eine Blockstruktur überführt werden. In einer späteren Arbeit wurde noch ein Mangel in dem Verfahren aufgedeckt und korrigiert [94].

Andere Ansätze zielen darauf ab, per *Correctness by Construction* von vornherein nur wohlstrukturierte Prozessgraphen zuzulassen [58]. Hierbei fügt der Benutzer nicht einzelne Knoten und Kanten zu einem Prozessgraphen zusammen, sondern baut diese aus größeren Bausteinen wie Schleifen- und Verzweigungsblöcken auf. Nachteil dieser Methode ist, dass der Entwickler bei der Modellierung der Prozesse eingeschränkt wird, und es ihm ohne Erfahrung in strukturierter Programmierung, die dem typischen Nutzer von Prozessnotationen möglicherweise fehlt, schwer fallen könnte, entsprechende Workarounds zu modellieren.

3.3.3. Übersetzung und Restrukturierung

Die wohl bekannteste Übersetzung von Prozessdiagrammen in ausführbare Programme ist die Abbildung von BPMN auf BPEL-Prozesse [117, Annex A][118, Kapitel 14], mit deren Hilfe die BPMN als Visualisierung für BPEL genutzt werden kann. In vielerlei Hinsicht ist die Abbildung sehr direkt, insbesondere was nicht-visuelle Elemente, wie Service-Beschreibungen anbelangt – hier ist nicht zu übersehen, dass BPMN mit eben dieser Abbildung auf BPEL im Hinterkopf entwickelt wurde.

Während der Abbildung der verschiedenen Elemente und Attribute sehr viel Raum gegeben wird, fällt die Abbildung der Strukturen eher knapp aus. Zur Identifikation von Blöcken und Schleifen wird die Verwendung von *Tokens* empfohlen, die dem Prozessverlauf folgend auf die verschiedenen Knoten verteilt werden. Neben dieser Abbildung, die Teil der BPMN-Spezifikation ist, haben verschiedene Autoren weitere Verfahren zur Abbildung von BPMN auf BPEL entwickelt, von denen einige im Folgenden vorgestellt werden sollen.

Mendling et al. [103] haben unterschiedliche Übersetzungsstrategien evaluiert. Im Fall der Abbildung von BPMN nach BPEL reichen die Ansätze hierbei von einer 1:1-Abbildung der Sequence Flows auf Links, wie sie in ähnlicher Form auch von White [171] verwendet wird, bis zu einer komplexeren, regelbasierten „Struktur-Maximierung“, ähnlich wie bei van der Aalst und Lassen [159]. In diesen Arbeiten werden allerdings nur solche Workflows transformiert, deren Struktur in wesentlichen Teilen bereits einer Blockstruktur entspricht, die sich also, nachdem die entsprechenden Strukturen im Prozessgraphen erkannt wurden, auf Verzweigungen und Schleifen abbilden lässt.

Eine ähnliche Untersuchung der Übersetzungsmöglichkeiten von BPMN nach BPEL kommt von Ouyang et al. [121]. Sie stellen drei Ansätze vor: Die Identifizierung von Strukturen innerhalb der BPMN-Diagramme, die auf BPEL-Kontrollstrukturen abgebildet werden können, die direkte Abbildung der Kanten des BPMN-Diagramms auf

`Link`s in einem `Flow`-Container (möglich nur bei nicht-zyklischen Workflows), sowie die Abbildung der Kanten auf Event Handler, vergleichbar damit, als würde jede einzelne Kante auf ein `GOTO`-Statement abgebildet [120]. Besonders in letzterem Fall ist der resultierende BPEL-Code nur außerordentlich schwer nachzuvollziehen und weiter zu bearbeiten.

Koehler und Hauser [83] stellen ein Verfahren vor, bei dem ein gegebener unstrukturierter, zyklischer Workflow zunächst in ein Gleichungssystem übersetzt wird; die einzelnen Gleichungen ordnen hierbei jedem Schritt des Workflows eine oder mehrere Aktivitäten sowie die jeweiligen Folgeschritte zu. Durch Auflösung des Gleichungssystems erhält man die Entsprechung für eine strukturierte Form des Workflows, in der gegebenenfalls Teile des Workflows dupliziert worden sind. Der Ansatz nutzt die eingangs erwähnten Ähnlichkeiten und überträgt eine ursprünglich zur Eliminierung von `GOTO`-Statements eingeführte Methode auf die Prozessmodellierung.

Kiepuszewski et al. [80] untersuchen verschiedene Klassen von Workflows mittels Bisimulation. Sie zeigen, dass schon leicht unstrukturierte Workflows die Einführung zusätzlicher Kontrollvariablen oder die Duplizierung von Teilen des Workflows (bzw. die Verwendung von Subroutinen) erfordern können. Beides sind Techniken, die in der strukturierten Programmierung ganz selbstverständlich eingesetzt werden, und den Prozess doch komplizierter und schwieriger zu warten machen.

3.3.4. Weniger Struktur, mehr Ausdruckskraft?

Die Frage, ob strukturierte oder unstrukturierte Programmierung „besser“ ist, hat in der Vergangenheit zu einer lebhaften Debatte geführt, in der sich letztlich die strukturierte Programmierung durchgesetzt hat. Die unstrukturierten Prozessgraphen stellen in dieser Hinsicht jedoch keinen Rückschritt dar. Die größte Kritik an der Programmierung mit `GOTO`-Statements war, dass es mitunter sehr schwer sein kann, die Sprünge im Programmfluss nachzuvollziehen: „[the go to statement] is too much an invitation to make a mess of one's program“ (E. W. Dijkstra, [44]). Dieses Problem tritt bei Prozessmodellen jedoch schon deshalb nur im sehr verminderten Maße auf, da die Sprünge durch ihre Visualisierung mit Kanten auf einen Blick ersichtlich sind. Zudem ist die Anzahl der Knoten in einem Prozessgraphen durch den höheren Abstraktionsgrad in der Regel erheblich geringer als die Anzahl der Zeilen in einem Programm.

Vor allem aber können Sprünge im Prozessablauf, sparsam eingesetzt, die Übersichtlichkeit auch verbessern, wie deutlich wird, wenn man die beschriebenen Restrukturierungen betrachtet, die notwendig sind, um bestimmte Prozessgraphen in strukturierte

Workflows und letztlich in Programmcode zu übersetzen. Ähnliche Verfahren wurden auch von anderen Autoren vorgestellt, wobei stets zusätzliche Kontrollvariablen eingeführt oder weite Teile des Prozesses dupliziert werden müssen.

Um so wünschenswerter ist es, dass diese Behelfskonstrukte bei der Übersetzung in strukturierten Programmcode automatisch eingefügt werden, sodass der Prozessdesigner sich – anders als bei auf *Correctness by Construction* basierenden Ansätzen – nicht selbst hierum kümmern muss, und dass die Ergebnisse sofort ausführbar sind. Auf diese Weise kann der Entwickler das gesamte System als Prozessmodell entwickeln und ist nicht gezwungen, nach der Transformation mit dem generierten Code weiterzuarbeiten.

Im Großteil der Papiere zur Strukturierung von Prozessgraphen werden hierbei der Einfachheit halber sehr minimalistische Prozessmodelle verwendet, in denen es nur einfache Verzweigungen und Zusammenführungen – jeweils in den Varianten AND und XOR – und Schleifen gibt. Im Gegensatz dazu warten *echte* Prozessnotationen mit einer Vielzahl weiterer Konstrukte auf, die dem Benutzer mehr Möglichkeiten an die Hand geben, zugleich jedoch auch weit höhere Ansprüche an die Übersetzung unstrukturierter Workflows stellen.

3.4. Fazit

Wenn es um die Modellierung eines Softwaresystems als Ganzes geht, ist die UML einer reinen Prozessbeschreibungssprache wie BPMN klar überlegen: Ein Prozessmodell beschreibt lediglich, wie sich ein System *verhalten* soll, UML hingegen beschreibt detailliert, wie genau dieses Verhalten *implementiert* ist. Im Falle der Multiagenten-Systeme stellt das jeweilige Agenten-Framework viele Aspekte der Implementierung zur Verfügung, beispielsweise einen Life-Cycle, Kommunikationsinfrastruktur, oder Planungsalgorithmen. Das einzige, was noch beschrieben werden muss, ist das eigentliche *Verhalten* der Agenten.

Während sich die UML an Aspekten der objektorientierten Programmierung orientiert, abstrahieren sowohl BPMN als auch Agenten von der Implementierung und stellen Aspekte wie Rollen und Ziele in den Vordergrund. Geschäftsprozessdiagramme, besonders solche in BPMN, lassen sich somit gut auf die Konzepte von Agenten abbilden: Prozessteilnehmer auf Agentenrollen; Prozesse, die diese Rollen ausführen, auf die Fähigkeiten der jeweiligen Rollen, etwa in Form einzelner Funktionsblöcke; und die Kommunikation auf den Austausch von Nachrichten zwischen den Agenten. Implementierungsdetails, etwa bezüglich der Softwarearchitektur, müssen nicht modelliert werden,

da diese bereits in einem geeigneten Agentenframework umgesetzt sind. Für diese Arbeit soll daher BPMN als Beschreibungssprache für Multiagenten-Systeme verwendet werden.

Natürlich gibt es Einschränkungen. Die konzeptionelle Abbildung von Geschäftsprozessen auf Agenten ist nicht perfekt. Während es in BPMN beispielsweise den Ad-Hoc-Prozess gibt, der in seiner Beschreibung große Ähnlichkeit mit einem Ziel hat, ist die Spezifikation in solchen Belangen, die sich nicht in BPEL, BPMNs vorrangiger Ausführungssprache, umsetzen lassen, wenig präzise und auch nicht vollständig. So gibt es etwa die Beschreibung des Ziels, aber keine Beschreibungen, welche Effekte die einzelnen Aktivitäten haben. Diese sind aber notwendig, wenn die Aktivitäten automatisch zur Erfüllung des Ziels ausgewählt werden sollen. Diese Lücken in der Abbildung von in BPMN beschriebenen Geschäftsprozessen auf Multiagenten-Systeme sind es, die in dieser Arbeit geschlossen werden sollen. Auf diese Weise können mit BPMN auch Aspekte wie deliberatives Verhalten und Ziele modelliert werden, die in der klassischen Prozessmodellierung nur eine untergeordnete Rolle spielen, in der Welt der Agenten jedoch einen zentralen Platz einnehmen.

Doch zunächst wird im nächsten Kapitel noch ein Überblick über verwandte Arbeiten gegeben, von denen einige ebenfalls auf BPMN, andere auf Erweiterungen der UML basieren.

Kapitel 4.

Verwandte Arbeiten und Ansätze

Während Multiagenten-Systeme insbesondere in der Forschung weit verbreitet sind, sind serviceorientierte Architekturen (SOAs) und Geschäftsprozessmanagementsystemen (BPMS) in der Praxis sehr viel geläufiger. Obgleich die beiden Ansätze gleichermaßen für den Einsatz in verteilten Systemen konzipiert sind, gibt es zahlreiche wichtige Unterschiede [25]: Der größte Vorteil von SOAs ist ihre Wiederverwendbarkeit. Während die Vision umfassender, semantisch beschriebener und durchsuchbarer Dienstverzeichnisse in der Praxis weitgehend unerfüllt blieb, können die Dienste einer SOA, oft in der Form von Webservices, etwa mittels SOAP oder REST, doch plattformübergreifend aufgerufen werden, wodurch die Services unterschiedlicher Anbieter zu komplexen Prozessen kombiniert werden können. Die Dienste selbst sind dabei rein passiv. Multiagenten-Systeme hingegen leiden unter einer großen Fragmentierung zwischen den verschiedenen Frameworks und uneinheitlichen Definitionen, was die Interaktion verschiedenartiger Agentensysteme in der Praxis schwierig gestaltet. Zugleich bieten sie mehr Flexibilität und, etwa durch die Verwendung von Autonomie und Zielen, weitergehende Möglichkeiten. Folglich wird eine Kombination der beiden Ansätze als außerordentlich vorteilhaft erachtet [25].

Die ähnlichen Einsatzgebiete von Agentensystemen, „klassischen“ SOA-Systemen und BPMS haben bereits zu einer Reihe von Ansätzen geführt, die drei Gebiete zu kombinieren und miteinander zu integrieren. Diese Ansätze beruhen typischerweise darauf, dass entweder:

- Schnittstellen zwischen Agenten und Webservices spezifiziert werden, sodass diese miteinander interagieren können, wozu in der Regel die Anreicherung der Webservices mit semantischen Informationen nötig ist,
- Agenten selbst zur Ausführung von Geschäftsprozessen oder einzelnen Diensten herangezogen werden, und diese auf einer höheren Ebene steuern und koordinieren, womit die Prozesse an Robustheit gewinnen, oder

- Abbildungen zwischen serviceorientierten Architekturen oder Geschäftsprozessen und einzelnen Aspekten von Agentensystemen, etwa Protokollen, definiert werden.

In den folgenden Abschnitten wird zunächst ein Überblick über Ansätze gegeben, die Agenten-Konzepte zusammen mit SOA und/oder BPM in einem System verbinden. Anschließend werden Abbildungen und Transformationen von Prozessmodellen auf Agentensysteme betrachtet. Zuletzt wird auf verschiedene Modellierungsverfahren, Methodologien und Werkzeuge zur Entwicklung von Agentensystemen – teilweise basierend auf den vorherigen Ansätzen – eingegangen.

4.1. Schnittstellen zwischen Agenten und Webservices

Wie eingangs bereits erwähnt erfüllen Agenten viele der an eine SOA gestellten Anforderungen. Um effektiv als (Teil einer) SOA eingesetzt zu werden, müssen die Agenten aber auch bestehende Dienste – insbesondere Webservices – nutzen können, und ihrerseits ihre Fähigkeiten für andere Komponenten als Dienste anbieten.

Ein Überblick über die Interaktionen zwischen Agenten und SOA, und wie Agenten benutzt werden können, um zielorientiert Webservices auszuführen, geben Dickinson und Wooldridge [41]. Sie erörtern hierbei insbesondere die Notwendigkeit semantischer Servicebeschreibungen. Walton [166] beschreibt zudem mögliche Herausforderungen, wenn verschiedene, als Webservices beschriebene Agenten in einem Multiagenten-System zusammenarbeiten sollen.

Es gibt verschiedene Arbeiten, die sich mit der Integration von Agenten und Webservices befassen; häufig in Form von „Gateways“, so haben etwa Greenwood und Calisti ein Modell eines Webservice-Gateways entwickelt, das zwischen FIPA-basierten JADE-Agenten und Webservices vermittelt [56]. Zu diesem Zweck werden Webservices und Dienste der Agenten sowohl an den bekannten UDDI-Verzeichnisdiensten als auch an deren Gegenstücken der Agentenwelt, den Direcory Facilitatoren (DFs), registriert. Dienst-Anfragen werden in beiden Sprachen, WSDL und ACL (Agent Communication Language), verarbeitet. In einem anderen Ansatz sind einzelne „Manager“-Agenten für verschiedene Webservices zuständig, wobei eingehende ACL-Nachrichten entsprechend übersetzt werden [128].

Für das JIAC-Framework existieren ebenfalls Schnittstellen zu WSDL- und REST-Services [96]: Das *Web-Service Gateway* und das *RESTful JIAC* Modul. Mit diesen ist es möglich, einerseits JIAC-Actions als WSDL- bzw. REST-Services verfügbar zu machen, und andererseits vorhandene Webservices in JIAC zu nutzen, als wären es JIAC-Actions.

4.2. Ausführung von Prozessen durch Agenten

Ein verbreiteter Ansatz zur Integration von Agenten und BPM ist, dass Prozessmodelle von Agentensystemen ausgeführt bzw. interpretiert werden. Hierbei spricht man häufig auch von einem *Agentenbasierten Prozessmanagement-System (APMS)*.

So sehen Singh und Huhns Agenten als die nächste Generation der Workflow-Technologie und argumentieren, weshalb Agenten besonders gut für die Ausführung von Workflows geeignet seien [150]. Sie benennen hierbei die Gemeinsamkeiten von Agenten und Prozessen als *Interaction Oriented Programming (IOP)*. Der Fokus ihrer Arbeit liegt dabei auf den Commitments und Meta-Commitments, die die Agenten untereinander machen.

In einer weiteren solchen Arbeit schlagen Buhler und Vidal vor, die heute üblichen starren BPEL-Workflows durch dynamisch orchestrierte und von Agenten ausgeführte Prozesse zu ersetzen [19, 20, 163]. Da eine vollständige Planung der Prozesse durch Agenten zu aufwändig erscheint, sollen BPEL-Prozesse in Petri-Netze überführt und diese von Agenten interpretiert werden, wobei geeignete Dienste per Planung ausgewählt werden sollen. Hierfür sollen die einzelnen Dienste statt mit WSDL mit DAML-S[1] beschrieben werden, sodass diese über eine semantische Beschreibung verfügen. Auf diese Weise kann das System auf den Ausfall einzelner Dienste reagieren und diese durch ähnliche Dienste ersetzen, während der Workflow als Ganzes vorgegeben ist.

Ein sehr ähnlicher Ansatz, der dem gleichen Prinzip folgt, wurde auch von Poggi et al. entwickelt [128]: Hier werden JADE-Agenten für die verteilte Ausführung eines BPEL-Prozesses verwendet, wobei einzelne Webservices dynamisch ausgewählt werden.

Eine weitere Integration von Agenten und Workflows wurde von Wagner und Moldt entwickelt [165]. Hierbei werden die *Agent Activities* oder AgAc in Form hierarchischer High-Level Petri-Netze dargestellt, die wiederum in größeren Workflows eingebettet sind und in einem *Agent Based WFMS* ausgeführt werden [105]. Durch die Verwendung von Petri-Netzen (vgl. Abschnitt 3.1.3) haben die Workflows eine sehr klare Semantik, doch komplexere Prozesse können für Nicht-Experten mitunter schwerer zu durchschauen sein als mit einer anderen Notation, wie etwa BPMN.

Schließlich wurde auch für das Jadex Agentenframework eine Prozess-Engine entwickelt, mit der Jadex-Agenten BPMN-basierte Prozesse ausführen können.[2] Die Prozesse werden hierbei mit dem Eclipse STP-BPMN-Editor erstellt und mit Jadex-spezifischen

[1] DARPA Agent Markup Language for Services, ein Vorläufer von OWL-S
[2] Jadex Processes: `https://www.activecomponents.org/`

Informationen annotiert. Insbesondere wird jedem Task eine Task-Klasse zugeordnet, in der beschrieben ist, was bei der Ausführung des Tasks geschehen soll.

Im Folgenden werden insbesondere drei Ansätze näher betrachtet: Das *ADEPT*-Projekt von Jennings et al. [77], in dem in einem Multiagenten-System Geschäftsprozesse ausgeführt werden, und *GO-BPMN* von Whitestein Technologies [29], in dem über Ziel-Hierarchien verbundene Prozesse von Agenten verknüpft werden, sowie das konzeptionell ähnliche Go4Flex [15].

4.2.1. Das ADEPT-Projekt

Jennings et al. haben im Projekt *ADEPT (Advanced Decision Environment for Process Tasks)* ein agentenbasiertes Prozessmanagement-System entwickelt, in dem hierarchisch angeordnete Agenten für verschiedene Aktivitäten verantwortlich sind [3, 77]. Die einzelnen Aktivitäten werden hierbei nicht zur Designzeit durch einen Workflow miteinander verbunden; stattdessen verwenden die Agenten Reasoning-Techniken, um ihre Aufgaben auszuführen, und Verhandlungsprotokolle, um dabei mit anderen Agenten zusammenzuarbeiten.

Die Agenten-Architektur besteht hierbei aus einer Reihe von Agencies (Agenturen) mit je einem verantwortlichen Agenten, der über einer Reihe von Tasks und möglicherweise weitere, untergeordnete Agencies verfügt. Auf diese Weise lassen sich reale Unternehmensstrukturen nachbilden, in denen die Agenten unterschiedliches Kooperationsverhalten zeigen, je nachdem in welchem Verhältnis sie zum jeweiligen Verhandlungspartner stehen [109]. Dieses Architekturprinzip wird beispielhaft an einem Unternehmensprozess der British Telecom angewendet [76].

Die wichtigsten Vorteile von Agenten gegenüber klassischen, zentralisierten Workflow-Managementsystemen werden hierbei in der natürlichen Abstraktion von realen Organisationsstrukturen durch Agenten sowie in der Anpassungsfähigkeit, Autonomie und Verteiltheit der Agentensysteme gesehen [113]. Als Nachteil des Ansatzes bezeichnen Jennings et al. jedoch die „fragmented view“ auf das Gesamtsystem, die es mitunter schwierig macht, das Verhalten der Agenten nachzuvollziehen und zu überprüfen [77, Seite 151].

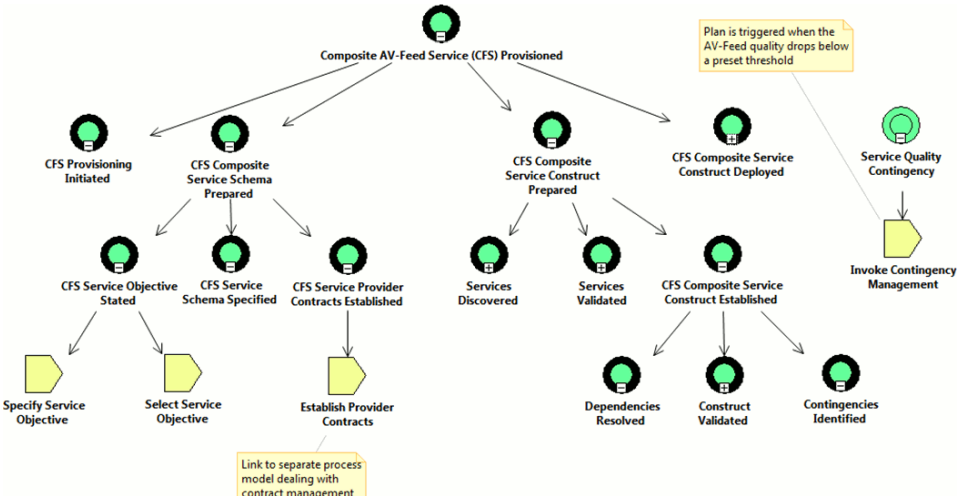


Abbildung 4.1.: GO-BPMN-Beispieldiagramm: Die Kreise stellen die Zielhierarchie dar, und die Fünfecke sind Prozesse, um diese Ziele umzusetzen. [29]

4.2.2. GO-BPMN und Whitestein LS/TS

Whitestein Technologies[3] verfolgen mit *Goal-oriented BPMN (GO-BPMN)* einen Ansatz, bei dem Prozesse und Ziele verknüpft werden [29]. Die Arbeiten entstanden in Kooperation mit Daimler Crysler Research & Development [22].

Die Ziele sind in einem Baum mit teils redundanten, teils alternativen Unterzielen angeordnet, in dem die jeweiligen Prozesse die Blätter darstellen (siehe Abbildung 4.1). Die Prozesse werden hierbei in BPMN beschrieben, wobei nur ein Subset von BPMN verwendet wird; insbesondere beschränken sich die Prozesse auf einen Pool, d. h. es werden keine Nachrichten modelliert. Die Ziele werden von Jadex-Agenten ausgewertet und die entsprechenden Prozesse werden ausgeführt.

Als Vorteile des Systems werden die Möglichkeiten zur Parallelisierung und die hohe Agilität, etwa durch das Aktivieren und Deaktivieren von Zielen auch zur Laufzeit, genannt. Durch die Verwendung unterschiedlicher Technologien ergebe sich jedoch das Problem, dass die einzelnen Prozesse nicht über einen gemeinsamen Kontext verfügen. Auch seien die Systeme schwer zu testen, da Seiteneffekte der Prozesse sich auf andere Ziele auswirken können [21].

Der Ansatz wurde in Whitesteins *Living Systems Technology Suite (LS/TS)* implementiert [57, 139], die in verschiedenen Projekten genutzt wird.

Die verwendeten Ziel-Hierarchien stellen eine gute Möglichkeit dar, die einzelnen Prozesse zu einem Gesamtsystem zu verknüpfen, wobei die BDI-Agenten selbständig unter

[3]Whitestein Technologies: `http://www.whitestein.com`

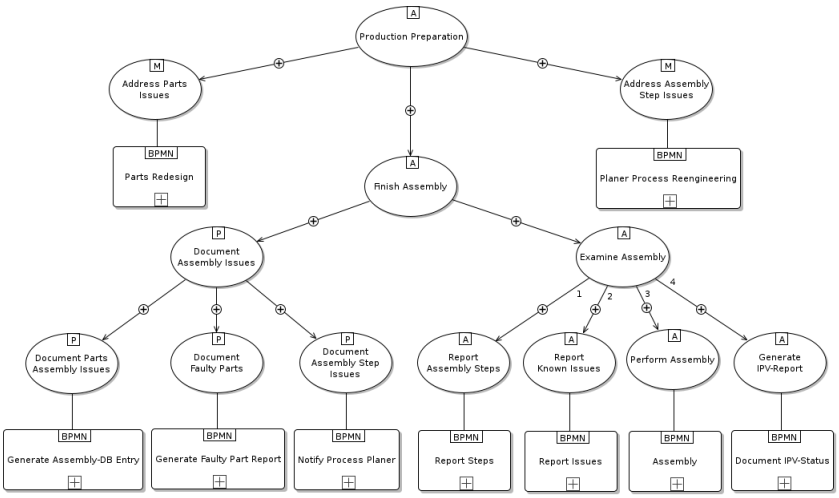


Abbildung 4.2.: GPMN-Beispieldiagramm für einen Fahrzeug-Fertigungsprozess. [73]

den Prozessen auswählen können. Jedoch scheint die Verwendung von BPMN auf zu niedriger Ebene zu erfolgen, sodass die Vorteile des hohen Abstraktionsniveaus der Prozessdiagramme nur eingeschränkt zum Wirken kommt. Insbesondere werden in den Prozessen nur die Abläufe eines einzelnen Agenten dargestellt, und die Stärken der BPMN bei der Modellierung von Kommunikation kommen nicht zum Tragen.

4.2.3. Go4Flex / GPMN

Ein ähnlicher Ansatz ist das ebenfalls in Kooperation mit Daimler entwickelte *Go4Flex* bzw. die darin verwendete *Goal-oriented Process Modeling Notation (GPMN)* [15]. Wie bei GO-BPMN werden hierbei Zielhierarchien aufgebaut, wobei jeder Zweig des Ziel-Baumes am Ende in einen BPMN-Prozess oder Prozessfragment endet. Motiviert wird der Ansatz damit, dass sowohl EPC als auch BPMN allein zu wenig flexibel für viele echte Geschäftsprozesse seien.

Zusätzlich zu den Konzepten, die bereits aus GO-BPMN bekannt sind, wurden „Activation Plans“ eingeführt, mit denen die Ziel-Hierarchien genauer beschrieben werden können. Für die Ziele wird das BDI-Modell [16] verwendet. Abbildung 4.2 zeigt eine in GPMN beschriebene Ziel-Hierarchie. Damit die entstehenden Graphen dennoch übersichtlich bleiben unterstützt der beschriebene Editor zudem verschiedene *levels of detail* [74].

Sowohl die Ziele als auch die Prozesse werden von Jadex-Agenten ausgewertet, die zu diesem Zweck mit einem BPMN-Interpreter ausgestattet sind und dem Referenz-Modell für WfMSs entsprechen [73]. Ein besonderer Fokus liegt zudem auf der Verteilung,

Synchronisation und Persistierung aller für den Prozess relevanter Ereignisse, sodass dieser auch nach einem (teilweisen) Absturz des Systems rekonstruiert werden kann [72].

Zusammen mit der für die einzelnen Pläne verwendeten BPMN stellt GPMN somit eine graphische Notation für Jadex-Agenten dar, die sowohl die Definition der BDI-Ziele als auch der zugrunde liegenden Pläne umfasst.

In einem anderen Paper beschreiben die Autoren zudem, wie FIPA-Interaktionsdiagramme auf BPMN abgebildet [129] und mittels *Jadex Active Components* [132] interpretiert werden können.

4.3. Übersetzungen zwischen Prozessmodellen und Agenten

In diesem Abschnitt werden verschiedene Arbeiten betrachtet, in denen Abbildungen und Modelltransformationen genutzt werden, um eine Verbindung zwischen Agenten und Geschäftsprozessen herzustellen. Die verschiedenen Ansätze umfassen dabei sowohl Abbildungen von Prozessen auf Agenten, als auch solche in der anderen Richtung. Auch der Fokus der Abbildungen ist unterschiedlich: Während in einigen Abbildungen die Interaktionen – bzw. die Modellierung derselben – im Vordergrund steht, werden in anderen Arbeiten Workflows eingesetzt, um das Verhalten einzelner Agenten zu beschreiben.

Chen-Burger et al. benutzen Geschäftsprozesse um Agenten-Interaktionsprotokolle, jedoch keine vollständigen Agentensysteme, zu modellieren [34]. Sie verwenden hierbei eine eigene Notation für die Geschäftsprozesse; zur Formalisierung der Interaktionsprotokolle verwenden sie den *Lightweight Coordination Calculus (LCC)* und ziehen Temporallogik als zugrunde liegende Semantik heran. In einer späteren Arbeit wurde der Ansatz weiter verfeinert und auf WS-BPEL-Prozesse angewandt [59], wobei jedoch nur eine Teilmenge von WS-BPEL unterstützt wird.

Ein ähnlicher Ansatz wird von Urzică et al. verfolgt [155], die BPMN-Prozesse (bzw. eine vereinfachte und vorstrukturierte textuelle Form von BPMN) auf AUML Interaktionsdiagramme (siehe Abschnitt 4.4.1) abbilden, von wo aus sie weiter zu JADE überführt werden können. Die Transformation ist notwendigerweise nur partiell, da beispielsweise BPMN-Aktivitäten und andere „interne“ Elemente keine Entsprechung in den Interaktionsdiagrammen haben.

Eine partielle Abbildung in die andere Richtung, von Agenten auf Prozesse, wurde von Cysneiros et al. untersucht [39]. Hierbei werden zunächst die Abhängigkeiten der

Agenten und ihrer Ziele untereinander mithilfe des i^*-Frameworks [181] modelliert, um aus diesen einen groben BPMN-Prozess abzuleiten.

Casella und Mascardi vertreten den Standpunkt, dass Agenten sowohl die koordinativen Fähigkeiten als auch die entsprechenden Designprinzipien implementieren, die für die Umsetzung von komplexen Anwendungen mittels Webservices nötig sind. Für diesen Zweck definierten sie eine Übersetzung von Interaktionsprotokollen der AUML zu WS-BPEL-Prozessen [30]. Diese BPEL-Prozesse werden dann in Prolog übertragen, und dazu genutzt, dass JADE-Agenten Schlüsse über die Protokolle ziehen können [100].

4.3.1. WADE

Ein weiterer Ansatz, der dieser Arbeit nahe kommt, ist *WADE (Workflows and Agents Development Environment)*: eine Erweiterung des JADE-Agentensystems, mit der einzelne Funktionen eines Softwareagenten in Form einfacher Workflows modelliert werden können [27, 144]. Hierbei wird eine sehr einfache Workflow-Notation verwendet, die im Wesentlichen nur aus Aktivitäten und Transitionen besteht, wobei jede Aktivität entweder eine bestehende Application oder Methode aufruft, oder selbst implementiert werden kann.

Basierend auf den Workflows wird jeweils eine Java-Klasse generiert. Hierbei wird klar zwischen den einzelnen Aktivitäten und dem dazwischen liegenden Kontrollfluss unterschiedenen: So wird eine Methode für den Workflow als Ganzes, also die Verknüpfung der einzelnen Aktivitäten entsprechend der Transitionen, und ein Methodenrumpf für jede Aktivität erzeugt. Dies hat den Vorteil, dass der Code nach einer Änderung des Workflows neu generiert werden kann, ohne dass die Implementierungen für die einzelnen Aktivitäten überschrieben werden. Ein Interessanter Effekt der Verwendung von Klassen ist zudem, dass Workflows voneinander erben können, d. h. ein einfacher Basis-Workflow kann mit zusätzlichen Elementen erweitert werden.

Negativ fällt auf, dass die Workflow-Notation sehr einfach gehalten ist und dementsprechend auch nur sehr einfache Prozesse erlaubt. Durch die Verwendung unterschiedlicher *Guards* an den Transitionen lässt sich der Kontrollfluss in Form von XOR-, OR- und AND-Splits modelliert. Jedoch werden in dieser Version weder Events nach Nachrichten unterstützt, und die Workflows beziehen sich immer nur auf eine einzelne, abgeschlossene Funktion eines Agenten; die Interaktionen zwischen Agenten lassen sich nicht in einem Workflow darstellen, und in den Workflows wird auch kein Gebrauch von typischen Agenten-Konzepten wie etwa Zielorientierung gemacht.

In späteren Versionen wurde WADE und die zugrunde liegende Notation um weitere Konzepte erweitert und einige dieser Schwächen behoben [7]. So bietet WADE mittlerweile Event-Handling (für Zeit, Nachrichten und Signale), Web-Service-Integration, sowie bessere Unterstützung für lange Zeit laufende Prozesse (u. a. Persistenz und Recovery). WADE wird in mehreren Projekten der Telecom Italia eingesetzt [7].

Mit *Wolf* existiert auch ein Eclipse-Plugin, mit dem die Workflow- und die Code-Ansicht synchronisiert werden können [28, 144].

4.4. Notationen, Methodologien und Werkzeuge

Es existiert eine große Zahl verschiedener Methodologien und Notationen für die Modellierung von Multiagenten-Systemen – eine aktuelle Übersicht bieten unter anderem Bordini [12], Dastani [40], Nunes [110] und Cossentino [38]. Anders als bei der objektorientierten Programmierung, bei der es mit der UML eine praktisch universell eingesetzte Notation gibt, verwenden die meisten Methodologien ihre eigenen Diagramme. Außerdem wird bemängelt, dass es eine konzeptionelle Lücke zwischen den verschiedenen Methodologien und den tatsächlichen Agenten-Frameworks gäbe [26]. So würden etwa Begriffe uneinheitlich verwendet, oder die Konzepte der Methodologien passen nicht zu den praktisch implementierten Funktionen und Fähigkeiten.

Es gibt auch Anstrengungen, die zahlreichen unterschiedlichen Agenten-Modelle und Methodologien zu vereinheitlichen. So vergleichen Bernon et al. [8] die Metamodelle von Gaia [179], ADELFE [127] und PASSI [37], und verbinden diese zum *Unified Multi-Agent System Metamodel*, das die Stärken der unterschiedlichen Modelle kombinieren und zugleich mit jeder der drei Methodologien kompatibel sein soll.

Padgham et al. haben mit der *Unified Graphical Notation* einen Vorschlag gemacht, wie sich die Diagramm-Notationen verschiedener Agenten-Methodologien vereinheitlichen lassen [125], wobei sie auf Prinzipien der Entwickler der UML aufbauen [140]. Sie betrachten hierbei nur Diagramme zur Agentenstruktur, nicht zu individuellen Verhaltensweisen, wie etwa Protokolle; hierfür empfehlen sie die Verwendung von beispielsweise *Agent UML* (siehe unten). Ob sich diese oder eine andere einheitliche Notation für Agentensysteme durchsetzen wird, muss sich jedoch erst noch zeigen.

In diesem Abschnitt sollen verschiedene Modellierungstechniken und Methodologien für Agentensysteme vorgestellt werden. Ein Schwerpunkt liegt hierbei auf solchen Ansätzen, die entweder ähnliche Konzepte verfolgen, wie sie in dieser Arbeit angewandt werden sollen, d. h. insbesondere der Einsatz von Prozessmodellierung, oder für ähnliche Aspek-

te der Modellierung besonders geeignet erscheinen, beispielsweise die Modellierung von Interaktionen zwischen Agenten.

4.4.1. Agent UML

Odell und Bauer haben mit *Agent UML (AUML)* die UML für Agentensysteme erweitert [114, 4], wobei Agenten selbst als Erweiterung von *Active Objects* betrachtet werden. Während AUML Erweiterungen für verschiedene Diagrammtypen beschreibt, hat vor allem die Erweiterung der UML Sequenzdiagramme zu *AUML Protokolldiagrammen* Bekanntheit erlangt. Insbesondere wird diese Notation in den Spezifikationen für verschiedene Interaktionsprotokolle der FIPA eingesetzt,[4] und auch verschiedene Agenten-Methodologien nutzen AUML zur Beschreibung von Interaktionen.

Protokolldiagramme sind im Wesentlichen Sequenzdiagramme, wobei die Nachrichtenpfeile nicht Methodenaufrufe und deren Rückgabewerte („Messages“ im Sinne der objektorientierten Programmierung), sondern Sprechakte darstellen. Besonderes Augenmerk wird dabei auf Wiederverwendbarkeit gelegt: So kann etwa ein *Template* für eine Englische Auktion oder ein anderes verbreitetes Protokoll erstellt und in einer konkreten Implementierung genutzt werden. Auch können Interaktionsprotokolle ineinander verschachtelt werden. Ganz im Sinne der UML stellen die Protokolldiagramme dabei nur einen Aspekt des Agentensystems dar – eben die Kommunikation im Rahmen eines bestimmten Protokolls. Um darzustellen, was *zwischen* den einzelnen Interaktionen passiert, müssen die Protokolldiagramme mit anderen Diagrammtypen verknüpft werden, beispielsweise mit UML Aktivitätsdiagrammen.

Im Entwurf von Bauer und Odell [4], welcher auf der Version 1.x der UML aufbaute, können sowohl die Lebenslinien als auch die Kommunikationsflüsse geteilt werden (AND, OR und XOR), wodurch sich die verschiedenen Entwicklungen, die ein Protokoll nehmen kann, sehr gut veranschaulichen lassen. Da mit Version 2.0 der UML Fallunterscheidungen auch in herkömmlichen UML Sequenzdiagrammen eingeführt wurden, wurden die Protokolldiagramme dementsprechend angepasst, d. h. die verschiedenen Möglichkeiten werden nun untereinander in (ggf. geschachtelten) Kästen angeordnet [52], vergleichbar mit der Darstellung von Fallunterscheidungen in textuellen Programmiersprachen. Somit können Protokolldiagramme nun auch mit normalen UML-Werkzeugen erstellt werden und haben eine klarere Semantik, sind jedoch nicht mehr so anschaulich wie zuvor. Abbildung 4.3 zeigt einen direkten Vergleich der alten und neuen Notation.

[4] FIPA Interaktionsprotokolle: `http://www.fipa.org/repository/ips.php3`

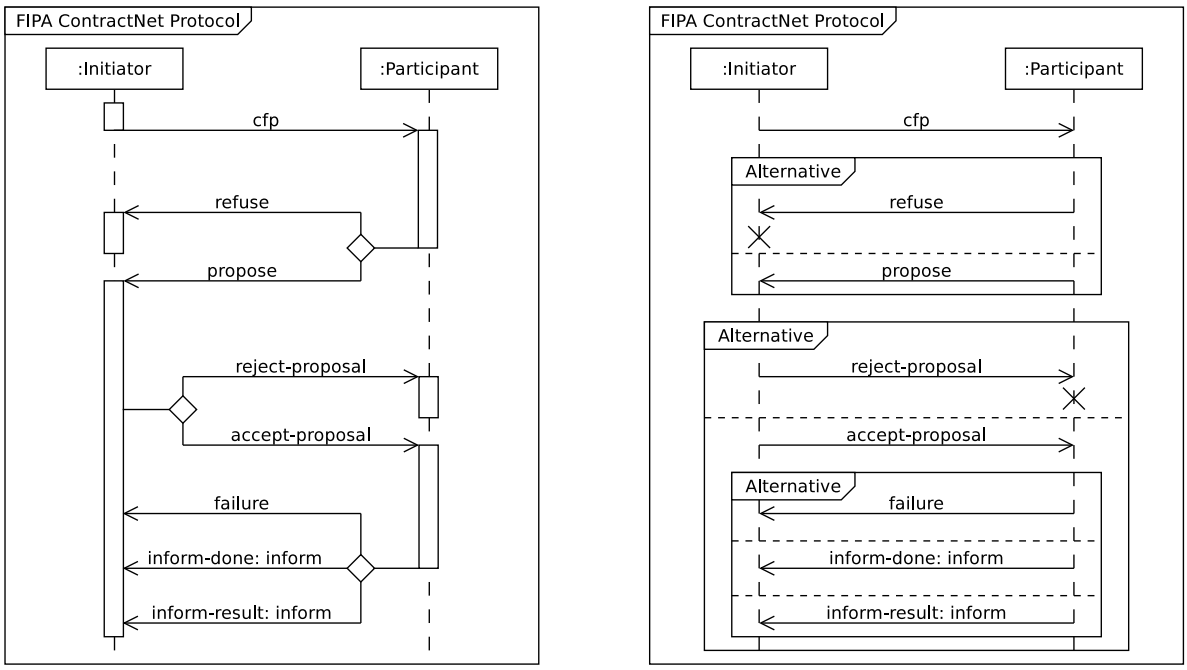


Abbildung 4.3.: Agent UML-Protokolldiagramm „Contract Net“. Links: basierend auf UML 1.x (nach [114]). Rechts: basierend auf UML 2.x (nach [71]).

Des Weiteren definiert die Agent UML Agenten-Diagramme – eine Abwandlung der UML Klassendiagramme – wobei die Agenten, statt über Attribute und Methoden, über ihre Rollen, Protokolle, Ziele, Actions und Services beschrieben werden. Actions wiederum können über einer Erweiterung von UML Aktivitätsdiagrammen beschrieben werden [71].

Seit einigen Jahren gibt es deutlich weniger Aktivität rund um Agent UML.

Seit 2008 gab es unter Mitwirkung von James Odell zudem eine Initiative der Object Management Group (OMG), ein *Agent Metamodel and Profile (AMP)* zu spezifizieren.[5] Bis auf einen *Request for Proposals* und ein *Initial Submission* Dokument [116] wurden jedoch noch keine Ergebnisse, und insbesondere kein finaler Standard veröffentlicht.

4.4.2. AML und ADEM

Cervenka und Trencansky von Whitestein Technologies haben mit der *Agent Modeling Language (AML)* eine weitere Erweiterung von UML geschaffen, die teils mit ähnlichen Konzepten wie Agent UML aufwartet, beispielsweise einer Erweiterung von Sequenzdiagrammen, mit denen sich Agenten-Interaktionsprotokolle modellieren lassen [32, 154].

[5] OMG Agent Platform Special Interest Group: `http://agent.omg.org/`

Eine weitere Besonderheit stellen *Social Interaction Diagrams* dar, in denen beschrieben ist, wie einzelne Agenten zueinander stehen (bspw. *cooparate* oder *manage*). Ferner können einzelne Klassen explizit als *Agent*, *Environment*, oder *Resource* gekennzeichnet werden. Es können auch Ziel-Hierarchien aufgestellt werden, so wie sie auch in GO-BPMN verwendet werden.

In der *Agent-oriented Development Methodology (ADEM)* wird AML in Kombination mit Prozessmodellierung eingesetzt [31]. Hierbei wird das allgemeine Verhalten des Systems mit Prozessen modelliert, und für die einzelnen Agenten, die diese Prozesse ausführen sollen, wird AML verwendet.

4.4.3. PIM4Agents und DSML4MAS

Die Gruppe um Christian Hahn und Klaus Fischer arbeitet an einem plattformunabhängigem Metamodell zur Beschreibung von Multiagenten-Systemen, und darauf aufbauender Werkzeuge und Methodologien, wobei unter anderem auch Prozessmodellierung zum Einsatz kommt. Die auf diese Weise erstellten Agenten-Modelle können in die Agentenframeworks JACK und JADE übertragen werden [61].

Erste Ansätze hierfür wurden bei der Abbildung eines vergleichbaren Metamodells für SOAs (PIM4SOA) auf AgentMM, dem Metamodell des Agentenframeworks JACK, entwickelt [62]. Dieses Metamodell wurde zum *Platform-Independent Metamodel for Agents (PIM4Agents)* weiterentwickelt und generalisiert, in dem insbesondere die Rollen, Fähigkeiten und Verhaltensweisen des Agenten beschrieben sind. Das Modell gliedert sich in mehrere *Views* für verschiedene Aspekte von Multiagenten-Systemen: Die *Multiagent-*, *Agent-*, *Behavioural-*, *Organisation-*, *Role-*, *Interaction-* und *Environment views*.

Darauf aufbauend wurde die *Domain-Specific Modeling Language for Multiagent-Systems (DSML4MAS)* entwickelt [60], in der sich insbesondere die Abhängigkeiten zwischen den verschiedenen Agenten und Gruppen, sowie (als workflowartige Diagramme) das Verhalten einzelner und die Interaktionen mehrerer Agenten darstellen lassen (siehe Abbildung 4.4), wobei unterschiedliche Diagramm-Typen für die verschiedenen Aspekte, oder Views, des PIM4Agents verwendet werden. Auf diese Weise lassen sich Interaktionen zwischen Agenten, wie im Contract-Net-Protokoll, graphisch modellieren und etwa auf das JACK-Agentenframework abbilden [63].

Das *DSML4MAS Development Environment* (DDE), das durch die Erzeugung von „protected regions“ das Weiterarbeiten am generierten Code vereinfacht, wird als Entwicklungsumgebung für JADE-Agenten vorgeschlagen [167].

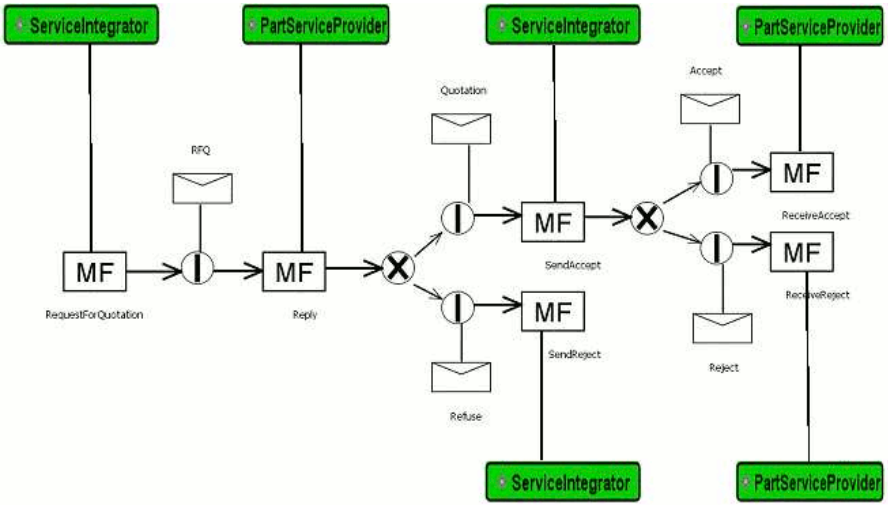


Abbildung 4.4.: PIM4Agents Interaktionsdiagramm. [186]

Zinnikus et al. verwenden das PIM4Agents-Metamodell und die Transformation nach JACK, um agentenbasierte Web-Service-Kompositionen zu erstellen. Als größten Vorteil der Agenten gegenüber der Verknüpfung mittels BPEL schätzen sie dabei die Flexibilität und Robustheit der Agenten ein, etwa um Dienste zur Laufzeit auszuwählen [186].

4.4.4. Gaia

Eine der bekanntesten und vermutlich einflussreichsten Agenten-Methodologien ist *Gaia*, die unter anderem auch im ADEPT-Projekt angewendet wurde (siehe Abschnitt 4.2).

In der ersten Version der Methodik [179] liegt der Schwerpunkt auf Rollen und deren Interaktionen, die in der Analyse-Phase beschrieben und in der Design-Phase zu Agenten und Diensten ausgearbeitet werden. Die Rollen werden hierbei über ihre Schreib- und Leseberechtigungen, zur Verfügung stehende Aktionen und Interaktionen und einen auf diesen gründenden Lifecycle beschrieben. Die Dienste werden in der Design-Phase semantisch beschrieben (Input, Output, Precondition und Effect). Die Organisationsstrukturen und Fähigkeiten sind hierbei statisch, womit nur geschlossene Systeme beschrieben werden können.

In einer späteren Version [183, 184] wurde Gaia für die Beschreibung offener Systeme erweitert, in die Agenten eintreten und unterschiedliche Ziele verfolgen können. Die Methodik wurde hierbei um die Beschreibung von Organisationen und den darin geltenden Regeln ergänzt und die Design-Phase in das *Architectural Design* und das *Detailed Design* aufgespalten (Abbildung 4.5). Anforderungsanalyse und Implementierung

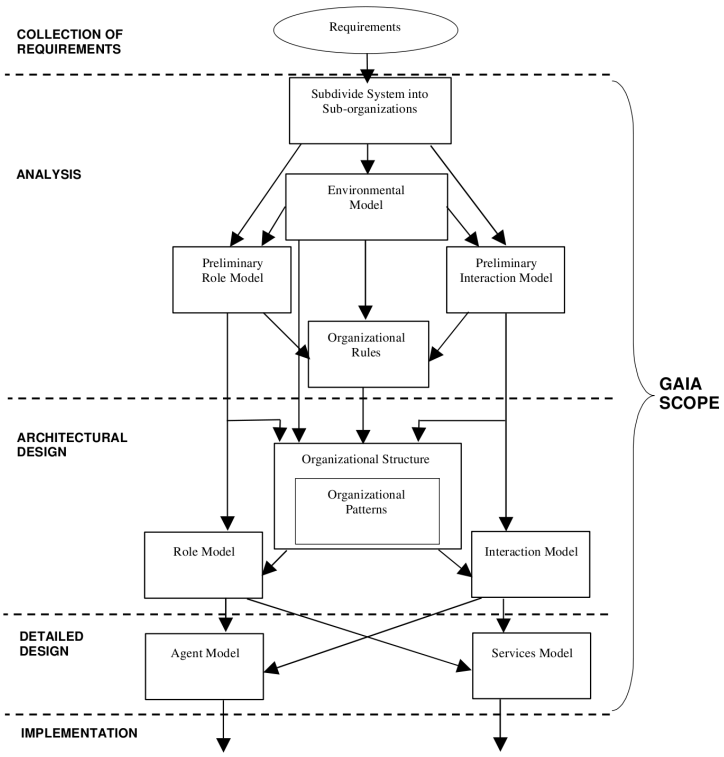


Abbildung 4.5.: Gaia: Prozess und Modelle. [183]

werden nach wie vor nicht betrachtet, und auch Iterationen der einzelnen Schritte sind nicht vorgesehen. Die Methodik wird an mehreren Beispielen veranschaulicht, wobei auch ausführlich auf verschiedene Einflussfaktoren eingegangen wird, die bei der Entwicklung berücksichtigt werden sollten.

Gaia trifft hierbei keinerlei Aussage darüber, wie das beschriebene System implementiert werden soll, oder ob die Agenten beispielsweise FIPA-Nachrichten oder einen BDI-Lifecycle unterstützen müssen. Somit lässt sich Gaia mit verschiedenen Agentenframeworks oder auch klassischen objektorientierten Sprachen anwenden und mit Notationen wie etwa UML und AUML [4] kombinieren. Zugleich bemerken die Autoren jedoch auch, dass es im Bereich der AOSE noch an geeigneten Werkzeugen und Entwicklungsumgebungen fehlt [183]. Ein Weg, anhand der Gaia-Methodik JADE-Agenten zu entwickeln wurde von Moraitis et al. vorgeschlagen [106], hierbei handelt es sich jedoch um einen manuellen Prozess, d. h. ohne automatische Code-Erzeugung.

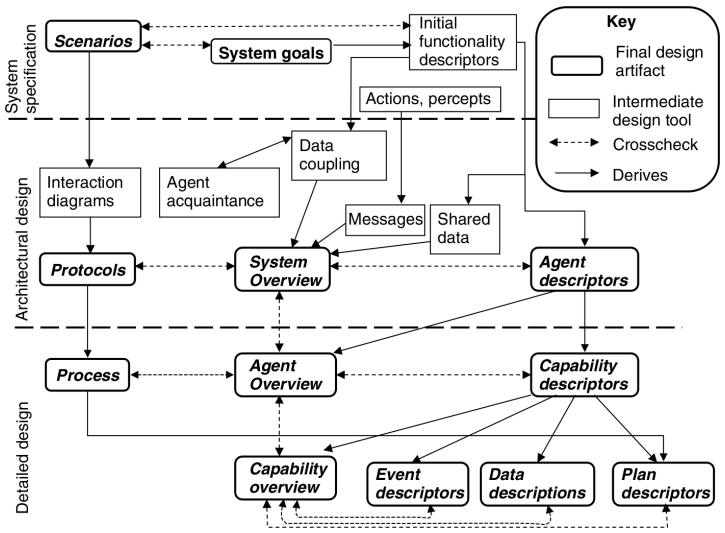


Abbildung 4.6.: Phasen der Prometheus-Methodik. [172]

4.4.5. Prometheus

Eine weitere verbreitete Methodik ist *Prometheus*. Im Vergleich zu Gaia ist Prometheus praxisnäher, behandelt auch Aspekte der Implementierung und unterstützt iterative Entwicklung [124, 172]. Die Methodik unterstützt BDI-Agenten (z. B. Jadex und JACK), kann aber auch für reaktive Agenten angewendet werden. Prometheus wurde erfolgreich in der Lehre und der Industrie eingesetzt.

Ähnlich wie Gaia gliedert sich die Methodik hierbei in drei Phasen: *System Specification*, *Architectural Design* und *Detailled Design* (Abbildung 4.6). In der ersten Phase werden die zu behandelnden Usecases identifiziert und verschiedene Rollen, oder „Functionalities“, mit ihren jeweiligen Perceptions, Actions und Events definiert. In der zweiten Phase, dem Architectural Design, werden die Rollen auf Agenten-Typen aufgeteilt und in einem sogenannten System Overview Diagram dargestellt. Die Interaktionen und die dabei ausgetauschten Datentypen werden ebenfalls beschrieben. Schließlich werden die verschiedenen Agenten-Typen näher beschrieben und anhand ihrer Capabilities, Events und Beliefs spezifiziert. Zur Beschreibung der einzelnen Modelle werden sowohl tabellarische Darstellungen als auch einfach verständliche Diagramme (u. a. AUML) verwendet.

Passend zur Prometheus-Methodik wurde das *Prometheus Design Tool (PDT)* entwickelt [122, 123], mit dem sich die verschiedenen Diagramme und Modelle erstellen lasen. Die Hauptaufgabe des PDT besteht darin, die Konsistenz der verschiedenen Diagramme und Modelle zu überprüfen, es kann aber auch genutzt werden, um Design-Dokumente und Code-Skelette (für JACK und Jadex) zu erzeugen.

In Prometheus werden Protokolle und die sie implementierenden Prozesse getrennt modelliert: Für die Protokolle werden AUML-Interaktionsdiagramme verwendet, und für die Prozesse Activity-Diagrams, die jeweils nur eine Seite der Interaktion zeigen. Um die Interaktionen anschaulicher und zugleich flexibler und robuster zu machen, haben Cheong und Winikoff [35] Prometheus mit der *Hermes*-Methodik kombiniert. Hierbei werden Interaktionen als Hierarchien von „Interaction Goals“ dargestellt, die jeweils mit „Action Maps“ spezifiziert werden, in denen – auf mehrere Swimlanes verteilt – die Aktivitäten aller beteiligter Rollen in einem Diagramm dargestellt sind. Die Aufteilung in Ziel-Hierarchien und Prozessdiagramme hat zudem Ähnlichkeit mit GO-BPMN bzw. Go4Flex (Abschnitt 4.2).

4.4.6. Tropos

Der Schwerpunkt der Methodik *Tropos* liegt auf der Analyse der Requirements, oder der Ziele, die das System erfüllen soll [18]. Die Methodik eignet sich besonders für BDI-Agenten und gliedert sich in fünf Phasen: *Early Requirements*, *Late Requirements*, *Architectural Design*, *Detailled Design* und *Implementation*.

In der ersten Phase werden die Stakeholder und deren Ziele identifiziert, wobei auch *i** [181] zum Modellieren sozialen Verhaltens und der Interaktionen verwendet [182] wird. Als nächstes wird ein zusätzlicher System-Akteur eingeführt, der seinerseits über Ziele verfügt, um die Ziele der Stakeholder zu erfüllen. Im nächsten Schritt wird der System-Actor in mehrere Sub-Akteure und die Ziele in Ziel-Hierarchien aufgebrochen, für die dann verschiedene Capabilities – Pläne und Protokolle – definiert werden. Hierbei werden UML-Aktivitätsdiagramme und AUML-Interaktionsdiagramme verwendet. Für die Implementierung wird ein BDI-Agentenframework wie JACK empfohlen.

4.5. Fazit

In den verschiedenen Arbeiten zur Integration von Agenten mit klassischen SOA- oder BPM-Systemen wird als größten Vorteil der Agenten deren Robustheit eingeschätzt: So können sie etwa im Falle eines Fehlschlagens selbständig einen alternativen Dienst auswählen, ohne dass dies explizit im Code beschrieben sein muss. Zugleich wird an vielen Stellen aber auch Kritik laut, dass die agentenbasierten Systeme schwerer zu durchschauen und zu überblicken seien, so etwa im ADEPT-Projekt und bei GO-BPMN.

Ansätze, dieser Unüberschaubarkeit durch geeignete Modellierungstechniken zu begegnen, gibt es viele, doch schafft es keine, das Agentensystem als Ganzes darzustellen: Die Interaktionsdiagramme von Agent UML sind gut geeignet, einzelne Protokolle übersichtlich darzustellen, doch müssen sie, wie in der UML üblich, zusammen mit zahlreichen anderen Diagrammen unterschiedlicher Art eingesetzt werden. Ein unüberschaubares Agentensystem wird damit nur hinter einem ähnlich unüberschaubaren Dickicht aus Diagrammen versteckt.

Wo Prozessmodellierung eingesetzt wird, da geschieht dies oft in einer Weise, auf die viele Vorteile der Prozessmodellierung abhanden kommen: So setzen sowohl GO-BPMN als auch WADE Prozessdiagramme nur auf sehr niedrigem Abstraktionsniveau ein. Insbesondere werden sowohl bei GO-BPMN als auch bei WADE nur die Prozesse einzelner Agenten modelliert, nicht jedoch die Interaktionen von Agenten, obwohl gerade dieser Aspekt stark von der Modellierung über eine Prozessnotation wie BPMN profitieren könnte.

Im nächsten Kapitel werden die Stärken und Schwächen der verwandten Arbeiten genauer betrachtet, um daraus einen Ansatz für die vorliegende Arbeit abzuleiten.

Teil II.

Konzept und Ausarbeitung

Kapitel 5.

Lösungsansatz und Konzept

Trotz der großen Zahl an bestehenden Arbeiten, die Verbindungen zwischen Agenten und Geschäftsprozessen schaffen – sowohl seitens der Modellierung als auch seitens der Ausführung –, ist dem Autor kein Verfahren bekannt, über welches sich das Verhalten von Multiagenten-Systeme, insbesondere die dort stattfindende Kommunikation, auf intuitive Weise nach Art der Geschäftsprozessdiagramme darstellen lässt.

In den folgenden Abschnitten soll zunächst erneut ein Blick auf den im vorherigen Kapitel vorgestellten Stand der Forschung geworfen und herausgestellt werden, welche Aspekte der Integration von Agenten und Prozessen in diesen Arbeiten besonders gut geglückt sind, und an welchen Stellen Möglichkeiten zur Verbesserung gesehen werden. Darauf basierend werden die Ziele beschrieben, die in dieser Arbeit erreicht werden sollen, und es wird ein Überblick darüber gegeben, wie diese Ziele in den folgenden Teilen der Arbeit erreicht werden sollen.

5.1. Auswertung des Stands der Forschung

Die Konzepte der agentenorientierten Softwareentwicklung decken sich sehr gut mit denen der Prozessmodellierung – nicht zuletzt deshalb, weil Agenten eine Abstraktion von Programmen darstellen, die sich an Menschen und dem menschlichen Miteinander orientiert, und Geschäftsprozessdiagramme in erster Linie dafür gemacht sind, Abläufe und Interaktionen von Menschen zu beschreiben. Insbesondere haben in beiden Welten Konzepte wie Rollen und Kommunikation zwischen diesen Rollen eine große Bedeutung.

Genau diese Aspekte sind es jedoch, die sowohl in GO-BPMN [21] als auch in WADE [27] wenig Beachtung finden, da in beiden nur einzelne Agenten betrachtet werden; Kommunikation zwischen Agenten wird nicht modelliert. In anderen Arbeiten, wie Agent UML [4] und DSML4MAS [61], gibt es sehr gute Lösungen für einzelne Aspekte, wie die

Modellierung eben der Kommunikation oder der Rollen, jedoch ist eine große Anzahl an einzelnen Diagrammen notwendig, um alle Aspekte eines Agentensystems abzudecken.

In anderen Punkten gibt es jedoch auch beträchtliche Unterschiede zwischen Agenten und Prozessen: Geschäftsprozesse sind dafür gedacht, bestehende Abläufe möglichst genau abzubilden oder Verfahrensanweisungen für neue Prozesse darzustellen, und sind dementsprechend sehr präzise, aber auch starr formuliert. Demgegenüber werden bei Agenten Konzepte wie Ziele und Planung verwendet, womit sich sehr viel dynamischeres und flexibleres Verhalten beschreiben lässt.

In GO-BPMN wird diesem Problem damit begegnet, dass das Verhalten der Agenten auf oberster Ebene durch Zielhierarchien definiert ist [21]; erst darunter, als verschiedene Möglichkeiten, einzelne Ziele umzusetzen, kommen die Prozessmodelle zum Einsatz. Gleichzeitig ergibt sich hieraus das Problem, dass das Verhalten der Agenten mitunter schwer nachzuvollziehen ist. Die Prozesse werden nur auf einem mittleren Abstraktionsniveau eingesetzt; das große Ganze, weswegen die einzelnen Prozesse aufgeführt werden, ist nur indirekt über die Ziele beschrieben. Ähnliches wurde auch im ADEPT-Projekt erkannt [77].

Bei Go4Flex und GPMN [73] zeigt sich ein ähnliches Bild, wobei der Ansatz alles in allem einen sehr ausgereiften Eindruck macht. Wie bei GO-BPMN werden Prozesse nur auf mittlerem Abstraktionsniveau eingesetzt, während die Abläufe und Zusammenhänge des Gesamtprozesses über Zielhierarchien beschrieben werden. Aus Sicht einer Agentensprache wie Jadex, die ebenfalls mit Zielen und Zielhierarchien arbeitet, hat dies den Vorteil, dass die Modellierung sehr nah an dem entsprechenden Agenten-Code ist, und für einen erfahrenen Entwickler entsprechend intuitiv benutzbar ist. Für Anwender aus der Prozessdomäne, die es gewohnt sind einen Gesamtprozess zu sehen, erfordert dies jedoch ein Umlernen.

In WADE wiederum stellt jede Aktivität einen Methodenaufruf dar, und der Prozessgraph wird dazu genutzt, diese übersichtlich und auf einer hohen Abstraktionsebene miteinander zu verbinden [27]. Auf diese Weise ist es insbesondere möglich, Details der Implementierung im Code anzupassen, oder die einzelnen Methoden im Prozess neu anzuordnen. Ursprünglich war das Prozessmodell in WADE jedoch sehr einfach gehalten; insbesondere gab es keine graphischen Elemente für Kommunikation, Parallelität oder die Behandlung von Ereignissen. Erst in späteren Versionen wurde die Notation vielschichtiger und ausdrucksstärker.

Prozessmodelle, insbesondere solche der BPMN, bieten zahlreiche vielversprechende Möglichkeiten, von denen viele direkt auf Agentensysteme abgebildet werden können;

insbesondere die Kommunikation zwischen den verschiedenen Rollen lässt sich sehr anschaulich darstellen und mit dem weiteren Verhalten der Agenten in Verbindung bringen. Alle genannten Arbeiten enthalten interessante Ideen und gute Konzepte, jedoch ist dem Autor kein Ansatz bekannt, in dem das Verhalten eines Multiagenten-Systems vollständig über ein Prozessmodell beschrieben werden kann. Insbesondere werden in vielen Ansätzen eigene, in der Geschäftspraxis wenig geläufige Notationen verwendet, oder etablierte Standards werden in erheblichem Maße um agentenspezifische Elemente erweitert.

5.2. Ziel dieser Arbeit

In dieser Arbeit soll untersucht werden, inwiefern sich ein Multiagenten-System mit Hilfe etablierter Methoden der Prozessmodellierung beschrieben werden, und sich im Umkehrschluss für deren Ausführung einsetzen lässt. Um bestmöglichen Gebrauch von den Prozessmodellen zu machen, sollen diese hierbei für die Beschreibung des Multiagenten-Systems als ganzen eingesetzt werden, nicht nur für Abläufe einzelner Agenten. Das Ziel ist, die Entwicklung verteilter Systeme zu vereinfachen und zu beschleunigen und für eine weitere Nutzergruppe zugänglich zu machen.

Hierzu soll ein Verfahren entwickelt werden, mit dem sich das Verhalten und die Interaktionen von Agenten über Prozessmodelle darstellen und im Sinne des model-driven Engineering entwickeln bzw. automatisch daraus ableiten lassen:

> Das Verhalten von Multiagenten-Systemen soll *vollständig* über graphische Prozessmodelle beschrieben werden können. Hierbei sollen die Prozessmodelle so eingesetzt werden, dass ihre Stärken zur Geltung kommen und diese *effektiv und effizient* für die Modellierung eingesetzt werden können. Insbesondere bei komplexen Interaktionen, der Verarbeitung verschiedener Ereignisse, oder nebenläufigen Abläufen hat die Darstellung sämtlicher Rollen in einem Prozessdiagramm großen Nutzen. Um alle *Aspekte des Verhaltens von Agenten* darstellen zu können müssen die Prozessnotationen gegebenenfalls um zusätzliche Elemente erweitert werden, etwa für komplexere Kommunikationsformen oder für Konzepte wie die Planung basierend auf Zielen. Für systemnahe Algorithmen und ähnliche „Implementierungsdetails“ hingegen sind Prozessnotationen weniger geeignet, sodass es an dieser Stelle möglich sein muss, *ergänzende Modelle* und/oder Quellcodeteile einzubinden.

Im Folgenden werden einige Aspekte und Teilziele näher erläutert:

Vollständig, effektiv und effizient: Die Prozessmodelle sollen nicht nur einen Teilaspekt des Verhaltens darstellen, beispielsweise die Kommunikation zwischen den Agenten, während für andere Aspekte andere Modelle verwendet werden müssen. Stattdessen soll jedes Verhaltensmuster, oder jeder Usecase, mit allen beteiligten Akteuren in einem Prozessdiagramm dargestellt werden. Das Verhalten komplexer Agentensysteme kann und soll auf mehrere solcher Diagramme aufgeteilt werden; das Ziel ist jedoch, dass jedes der Diagramme für sich betrachtet in sich abgeschlossen ist und jeden Aspekt des Verhaltens aufzeigt, von den das Verhalten auslösenden Ereignissen über die Kommunikation mit beteiligten Akteuren bis hin zu den internen Abläufen.

Insbesondere verfügt die BPMN über eigene Mechanismen um anzuzeigen, wann ein Prozess ausgeführt werden soll: den Start-Events. Würden auf oberster Ebene beispielsweise Ziele eingesetzt werden, könnte dieser Mechanismus nicht genutzt werden.

Auf diese Art lassen sich komplexe Prozesse und Interaktionen mitsamt ihrer jeweiligen auslösende Ereignisse effektiv modellieren und in entsprechende Multiagenten-Systeme übertragen, und das auf eine Weise, die auch Domänenexperten, die wenig Erfahrung mit den Konzepten agentenorientierter Softwareentwicklung haben, vertraut ist.

Die Stärken von Prozessbeschreibungssprachen wie BPMN liegen dabei insbesondere in der Darstellung von *abstrakten* Prozessen bzw. von Abläufen auf einem hohen Abstraktionsniveau: Insbesondere bei der Modellierung von Interaktionen zwischen verschiedenen Rollen oder bei die Behandlung von Ereignissen bietet die graphische Notationen einen echten Mehrwert gegenüber textuellen Programmiersprachen, in denen jede Rolle des Systems in einer anderen Datei beschrieben wird.

Aspekte des Verhaltens von Agenten: Zugleich zeigen die Prozessbeschreibungssprachen hier einige Einschränkungen: Zwar ließe sich Kommunikation zwischen Agenten beispielsweise in der BPMN sehr gut über Nachrichtenflüsse darstellen, jedoch wurde die BPMN, ebenso wie viele andere Notationen, primär für rein serviceorientierte Sprachen wie BPEL konzipiert. Folglich ist in BPMN einzig ein Nachrichtenaustausch über Services und deren Rückgabewerte vorgesehen. Um andere Arten von Kommunikation abzubilden, etwa Nachrichtengruppen oder auch asynchrone Nachrichten an bestimmte Agenten, muss die zu verwendende Notation entsprechend erweitert werden.

Ein weiterer Aspekt von Multiagenten-Systemen, der in klassischen Prozessbeschreibungssprachen oft nicht berücksichtigt wird, ist die Autonomie und Proaktivität der

Agenten: das Aufstellen und Verfolgen von Zielen, die Auswahl geeigneter Aktionen mittels Planung, und ähnliche „dynamische“ Aspekte. Während in diesem Ansatz nicht jeder Prozess über Ziele oder Zielhierarchien gestartet werden soll, sollen Ziele und Planung durchaus als Bestandteil der Prozesse eingesetzt werden können. BPMN enthält in Ansätzen Elemente die für solche Zwecke genutzt werden könnten, etwa Ad-Hoc-Subprozesse, jedoch mangelt es nicht zuletzt an der semantischen Beschreibung der verfügbaren Aktivitäten, ohne die diese nicht automatisch zu Plänen verbunden werden können, sodass diese Elemente in ausführbaren Prozessen bislang keine Rolle spielen und hauptsächlich in rein dokumentierenden Prozessen Verwendung finden.

Mit diesen Erweiterungen stellen Prozessnotationen und insbesondere BPMN eine interessante Möglichkeit dar, Agentensysteme graphisch zu modellieren. Hierbei wird vorausgesetzt, dass ein Agentenframework verwendet wird, welches das beschriebene Verhalten auf geeignete Weise umsetzen kann: Es soll nicht notwendig sein, die Interna von Planungsalgorithmen, der Kommunikation oder des Lifecycle-Managements in den Prozessen darzustellen; solche Funktionen müssen von dem jeweiligen Agentenframework bereit gestellt werden.

Ergänzende Modelle und Methoden: Für Details der Implementierung, etwa spezifische Algorithmen, Datentransformationen, etc., sind graphische Notationen weniger geeignet. Diese lassen sich im Quellcode einfacher und übersichtlicher beschreiben als im Prozessdiagramm. Dementsprechend sollte es möglich sein, Programmcode-Fragmente („Snippets“) direkt in das Prozessdiagramm einzubetten und/oder diese später zu ergänzen. Ebenso soll es möglich sein, die als Prozesse modellierten Agentensysteme mit auf herkömmliche Art und Weise erstellten Komponenten zu kombinieren.

Ferner sollen Aspekte wie Datenmodelle/Ontologien oder auch Benutzerschnittstellen *nicht* Teil der Prozessmodelle sein. Für diese Aspekte gibt es geeignetere Modelle und Notationen, wie etwa UML Klassendiagramme oder OWL. Es muss jedoch Möglichkeiten geben, diese zu referenzieren und beispielsweise für Variablen zu verwenden.

Hierfür soll eine Entwicklungsmethode konzipiert werden, mit der sich die prozessbasierte Modellierung der Multiagenten-Systeme nahtlos in die weiteren Entwicklungsschritte eingliedern lässt.

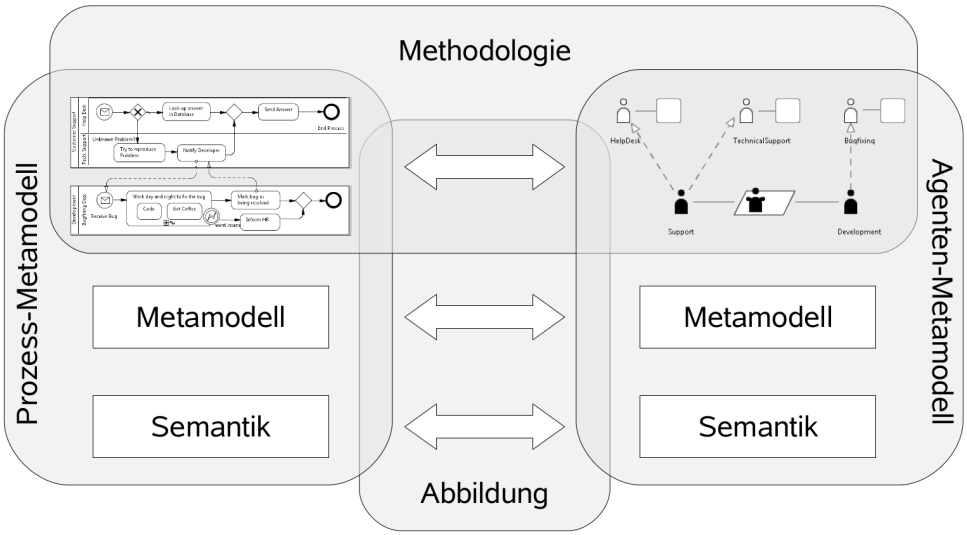


Abbildung 5.1.: Übersicht über den Ansatz und die Arbeitspakete.

5.3. Ansatz

Um diese Ziele zu erreichen, soll im Rahmen dieser Arbeit eine Abbildung von Geschäftsprozessmodellen auf Agentensysteme erarbeitet, prototypisch umgesetzt und evaluiert werden. Hierzu sollen zudem entsprechende Metamodelle, sowohl für Prozesse als auch für Agenten, entworfen werden. Schließlich soll die Abbildung in eine Entwicklungsmethodik eingebettet und prototypisch implementiert werden. Diese verschiedenen Aspekte und deren Zusammenhänge bzw. Berührungspunkte sind in Abbildung 5.1 dargestellt.

Im folgenden Teil dieser Arbeit werden zunächst die verwendeten Prozess- und Agenten-Modelle beschrieben, die als Domäne und Co-Domäne der Abbildung dienen werden (Kapitel 6).

Das Agentenmodell soll hierbei vergleichsweise abstrakt und allgemein gehalten werden, sodass die Abbildung auf unterschiedliche Agentenframeworks übertragbar ist, solange diese bestimmte Eigenschaften und Funktionen aufweisen. Insbesondere sollen verschiedene Arten der Kommunikation unterstützt werden, etwa Unicast und Multicast-Messages, sodass etwa komplexe Interaktionsprotokolle modelliert werden können. Ferner soll reaktives oder ereignisbasiertes ebenso wie deliberatives Verhalten unterstützt werden.

Als Grundlage für das Prozess-Modell soll BPMN dienen. Diese Notation verfügt über ein breites Repertoire an verschiedenen Elementen, die gut für die Modellierung von verteilten Systemen verwendet werden können. Um sämtliche Aspekte von Multiagenten-

Systemen zu beschreiben sind Erweiterungen nötig, etwa für vielfältigere Nachrichtentypen, semantische Dienstbeschreibungen, Ziele und Planung.

Mit diesen Modellen kann sodann eine Abbildung spezifiziert werden (Kapitel 7), wie Elemente des Prozess-Modells auf Agentensysteme übertragen werden, bzw. wie sich ein bestimmtes Verhalten eines Agenten als Prozess ausdrücken lässt. Der Fokus soll hierbei insbesondere auf dem Verhalten der Agenten und der Interaktion der Agenten untereinander liegen, sowie auf der Abbildung der Prozessstruktur auf äquivalente Agenten-Programme.

Anschließend wird beschrieben, wie sich diese Abbildung praktisch umsetzen lässt. Hierfür gibt es verschiedene Möglichkeiten, beispielsweise kann basierend auf den Prozessmodellen Quellcode für die Agenten generiert werden, oder die Prozesse können von einem Agenten direkt interpretiert werden. Mehrere solcher Ansätze sollen prototypisch umgesetzt und ihre jeweilige Vor- und Nachteile diskutiert werden.

Je nachdem, wie genau die Abbildung umgesetzt wird, lässt sich diese auf unterschiedliche Weise in den Agenten-Entwicklungsprozess einbeziehen. In jedem Fall wird es Aspekte geben, die sich nicht für die Modellierung über Prozesse eignen, und die vor und/oder nach der eigentlichen Abbildung geschehen müssen. Das Zusammenspiel der Prozessmodellierung mit diesen verschiedenen anderen Techniken soll in einer Methode zur prozessorientierten Entwicklung von Agentensystemen beschrieben werden (Kapitel 8).

Schließlich soll die Abbildung von Geschäftsprozessen auf Multiagenten-Systeme und die damit verbundene Methodik und die Implementierungen anhand praxisnaher Beispiele und im Rahmen tatsächlicher Projekte evaluiert werden (Kapitel 9). Hierbei soll der Frage nachgegangen werden, inwieweit die Verwendung von Prozessmodellen im Allgemeinen und die hier beschriebene Abbildung im Besonderen beim Erstellen von Multiagenten-Systemen hilft, welche Vor- und Nachteile sich hierbei zeigen, und in welchen Fällen sie den größten Nutzen bringt.

In den folgenden Kapiteln wird jeder dieser Aspekte – die Metamodelle für Agenten und Prozesse, die Abbildung zwischen den Modellen, sowie eine entsprechende Entwicklungsmethodik und Implementierung – erarbeitet und detailliert beschreiben.

Kapitel 6.

Beschreibung der Metamodelle für Agenten und Prozesse

In diesem Kapitel sollen die beiden Metamodelle für Agenten und für Prozesse beschrieben werden, die die Grundlage der im nächsten Kapitel beschriebenen Abbildung bilden.

Wie im vorherigen Abschnitt erwähnt, kann der größte Nutzen von Prozessmodellen für die Modellierung von Multiagenten-Systemen in der übersichtlichen Darstellung von Verhaltensweisen auf hoher Abstraktionsebene und insbesondere für die Kommunikation und Interaktion zwischen den einzelnen am Prozess beteiligten Agenten gesehen werden. Dementsprechend sollen sich auch die hier beschriebenen Metamodelle insbesondere auf die folgenden Aspekte konzentrieren:

- *Services:* Ursprünglich als Notation zur Beschreibung von BPEL-Webservices gedacht, eignet sich BPMN besonders gut für die Modellierung von Services, und auch in manchen Agentenframeworks spielen Services zunehmend eine Rolle [56, 67].
- *Nachrichten:* Komplementär zu Services ist für komplexe Interaktionsprotokolle mit mehreren Partnern nachrichtenbasierte Kommunikation unerlässlich.
- *Prozesse* Neben der Interaktion der Agenten untereinander sollen insbesondere auch die Abläufe innerhalb der einzelnen Agenten modelliert werden können.
- *Ereignisse:* Verschiedene Arten von Ereignissen, die sich mittels der BPMN gut modellieren lassen, spielen insbesondere für reaktive Agenten eine wichtige Rolle.
- *Semantik und Planung:* Die Aktionsplanung basierend auf semantischen Beschreibungen ist in der Agentenwelt sehr viel verbreiteter als in der Prozessmodellierung, doch mit *Ad-Hoc*-Prozessen bietet BPMN auch hierfür ein geeignetes Konzept an.

Im Folgenden werden zunächst das Agenten-, und anschließend das Prozess-Metamodell beschrieben. Hierbei wird eine einfache, semi-formale Notation verwendet, mit der sich die Elemente, ihre Relationen und ihre Attribute vergleichsweise klar und kompakt darstellen lassen. Anschließend werden die Metamodelle anhand eines einfachen Beispiels verdeutlicht.

Wie in den vorherigen Kapiteln dargelegt wurde, existiert bereits eine Vielzahl an Metamodellen sowohl für Multiagenten-Systeme (vgl. [8]) als auch für Geschäftsprozesse. Ziel dieses Kapitels soll es nicht sein, diesen Listen ein weiteres Metamodell hinzuzufügen, noch erheben die hier beschriebenen Modelle den Anspruch auf Vollständigkeit in dem Sinne, als dass sie sämtliche Konzepte der existierenden Modelle in sich vereinen. Entsprechend dem Ziel dieser Arbeit – aufzuzeigen, wie Multiagenten-Systeme mit Prozessen modelliert werden können – sollen in diesem Kapitel zwei minimale, für die in dieser Arbeit beschriebene Abbildung und deren Formalisierung nutzbare Metamodelle entwickelt werden. Dementsprechend konzentrieren sich die Modelle auf diejenigen Elemente, die in der Abbildung von Bedeutung sind.

In der verwendeten Notation werden die verschiedenen Elemente als Tupel beschrieben. Hierbei werden englische Begriffe für die Elemente des Modells und deutsche Begriffe für die dahinter stehenden Konzepte verwendet. Bezeichner mit kleinem Anfangsbuchstaben stehen für einzelne Elemente und solche mit großem Anfangsbuchstaben für Mengen oder Listen. Kapitälchen stehen für Elemente einer Aufzählung. Ferner werden die üblichen Symbole $\varnothing$ für die leere Menge, [] für die leere Liste, $\top$ und $\bot$ für Wahr und Falsch, und ε für das leere Element, oder „null“, verwenden.

6.1. Ausarbeitung des Agenten-Metamodells

Ein generelles Problem bei der Arbeit mit Multiagenten-Systemen ist, dass es keine einheitliche Definition gibt, was genau einen „Agenten“ ausmacht, und aus welchen Teilen er besteht (siehe Kapitel 2, insb. [53]). In Anbetracht der Vielzahl an konkurrierenden Agenten-Metamodellen wird für diese Arbeit daher eine eigene Definition von Agenten und Multiagenten-Systemen verwendet, die speziell an den Anforderungen für die formale Spezifikation der im nächsten Kapitel vorgestellten Abbildung ausgerichtet ist. Dementsprechend orientiert sich das Modell auch vorwiegend an denjenigen Aspekten, die für die Abbildung relevant sind.

Das Modell wird hierbei vergleichsweise abstrakt gehalten, ohne eine konkrete Implementierung vorzugeben. Im späteren Kapitel 8 und in der darin beschriebenen Imple-

mentierung wird das JIAC-Framework [98] als Referenzimplementierung verwendet.

6.1.1. Konzeption und Zusammenstellung

Das Agenten-Metamodell konzentriert sich vornehmlich auf die eingangs aufgeführten Aspekte von Agenten, für deren Modellierung Prozessmodelle besonders geeignet scheinen: Kommunikation und Interaktion basierend auf Nachrichten und Diensten, Ereignisse, Semantik und Planung.

Hierfür werden im nächsten Abschnitt die folgenden Aspekte näher beschrieben:

- Den Aufbau von Multiagenten-Systemen aus einzelnen Agenten und deren Zusammensetzung aus verschiedenen Rollen, die sich wiederum durch verschiedene Pläne, sowie Regeln und Ziele zum Ausführen dieser Pläne definieren.
- Die Elemente aus denen die Agenten-Pläne zusammengesetzt sind, um grundlegende Funktionen wie das Versenden und Empfangen von Nachrichten oder das Aufrufen von Diensten zu ermöglichen, sowie verschiedene Kontrollfluss-Konstrukte.
- Anforderungen an die zur Beschreibung von Ausdrücken und semantischen Beschreibungen verwendeten Sprache.
- Die Repräsentation des Wissens und der Daten des Agenten, und insbesondere das Nachrichtenformat.

Mit diesen Konzepten lassen sich die Agenten eines Multiagenten-Systems abstrakt beschreiben, ohne zu strikte Anforderungen zu stellen, wie diese Funktionen von einem konkreten Agentenframework implementiert sein müssen.

Zugleich muss vorausgesetzt werden, dass gewisse Funktionen von dem jeweiligen als Ziel der Abbildung verwendeten Agentenframework bereits unterstützt werden und genutzt werden können. Hierzu zählt etwa der Execution-Cycle des Agenten, die Infrastruktur zum Versenden und Empfangen von Nachrichten, Regel-Engines zum Überprüfen von Bedingungen, oder die Handlungsplanung. Diese Mechanismen und Algorithmen sind weder Teil dieser Arbeit, noch sollen sie mittels der Prozessdiagramme modelliert werden müssen.

6.1.2. Formale Beschreibung

Im Folgenden sollen die verschiedenen Aspekte des Multiagenten-Modells formell beschrieben werden (siehe auch [92]). Die Beschreibung gliedert sich hierbei in die Zusam-

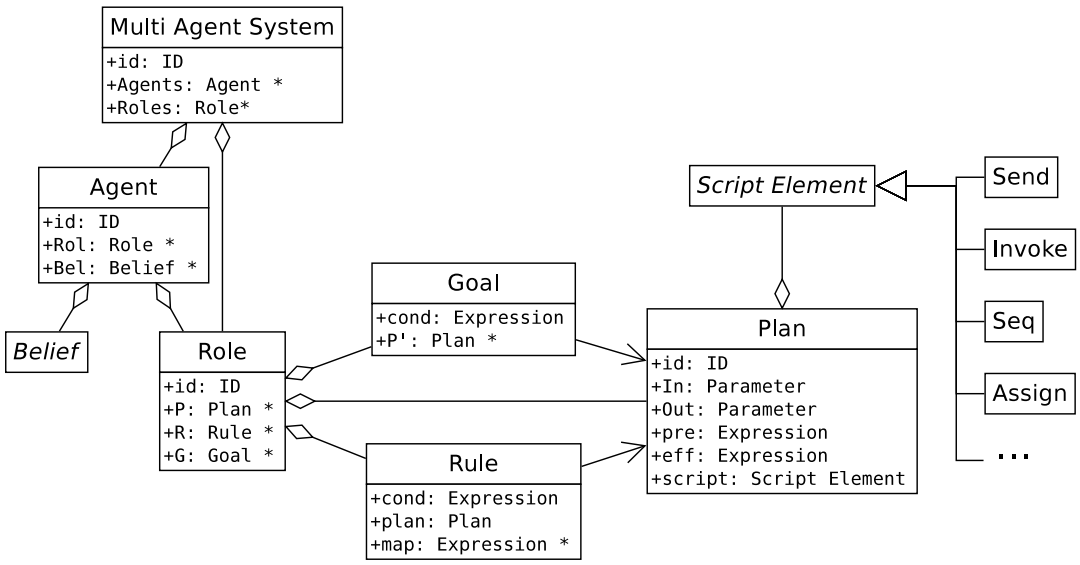


Abbildung 6.1.: Klassendiagramm des Agenten-Metamodells, geringfügig vereinfacht. [92]

mensetzung der Agenten, die für die Agenten-Pläne notwendigen Konstrukte, und Hilfskonstrukte wie etwa Daten, Regeln und Nachrichten. Abbildung 6.1 zeigt das Agenten-Metamodell, geringfügig vereinfacht, in Form eines Klassendiagramms.

Aufbau und Zusammensetzung der Agenten

Das Multiagenten-System als Ganzes ist im Wesentlichen nur eine Menge von Agenten. In der Praxis enthält es neben den eigentlichen Agenten auch gewisse Infrastruktur-Elemente, etwa ein Verzeichnis mit bekannten Agenten und Diensten, oder Kommunikations-Infrastruktur. Diese Elemente sollen jedoch nicht über die Prozessmodelle modelliert werden, sondern werden als gegeben vorausgesetzt und werden infolgedessen hier nicht weiter betrachtet.

Multiagenten-System: Ein Multiagenten-System *mas* ist ein Tupel (*id*, *Agents*, *Roles*). Hierbei ist *Agents* eine Menge von Agenten und *Roles* die Menge aller im System vertretenen Agentenrollen. *id* ist der eindeutige Bezeichner des Multiagenten-Systems.

Ein Agent ist in erster Linie über die Rollen definiert, die er implementiert. Außerdem verfügt jeder Agent über eine Menge von Beliefs, seinen *Memory*, auf die alle Rollen gleichermaßen zugreifen können. Die Art, wie diese Beliefs repräsentiert werden – ob als

Objekte, als Prädikate, oder mittels Temporallogik – ist für die vorliegende Arbeit nicht von Bedeutung.

Agent: Sei (id, $Agents$, $Roles$) ein MAS. Ein Agent $a \in Agents$ ist ein Tupel (id, Rls, Bel). Hierbei ist $Rls \subseteq Roles$ eine Menge von Rollen, die der Agent implementiert, und Bel die Menge der aktuellen Beliefs des Agenten. id ist der eindeutiger Bezeichner des Agenten.

Rollen definieren das Verhalten des Agenten, wobei jeder Agent eine oder mehrere Rollen annehmen kann. Jede Rolle setzt sich hierbei aus Plänen zusammen, sowie Regeln und Zielen nach denen diese Pläne ausgewählt und ausgeführt werden. Bei rein reaktiven Agenten ist die Menge der Ziele leer.

Rolle: Eine Rolle r ist ein Tupel (id, P, R, G). Hierbei ist P eine Menge von Plänen, R eine Menge von Reaktionsregeln, und G eine Menge von Zielen, die mit der Rolle einher gehen. id ist der eindeutiger Bezeichner der Rolle.

Pläne beschreiben die einzelnen „atomaren“ Verhaltensweisen und Fähigkeiten einer Rolle. Zwar kann ein Plan wiederum aus mehreren Skript-Elementen zusammengesetzt sein, diese können jedoch nicht einzeln, sondern nur in einen Plan zusammengefasst ausgeführt werden. Die Signatur des Plans setzt sich hierbei aus einem eindeutigen Namen sowie dem Input und Output des Plans und seiner Precondition und seines Effektes zusammen (IOPE). Der Rumpf des Plans ist über ein (komplexes) Skript-Element definiert (siehe nächster Abschnitt).

Plan: Ein Plan p ist ein Tupel (id, In, Out, pre, eff, $script$). Hierbei ist In die Menge der Eingabeparameter und Out die Menge der Rückgabewerte. pre ist eine Bedingung, die vor der Ausführung des Plans erfüllt sein muss, und eff ist eine Bedingung, die durch die Ausführung des Plans erreicht werden soll, wobei pre Elemente aus In, und eff Elemente aus In und Out als Variablen enthalten darf. $script$ ist der Rumpf des Plans, der beschreibt, was bei der Ausführung des Plans geschieht, und sich aus verschiedenen Skript-Elementen zusammensetzt. id ist der eindeutige Bezeichner des Plans.

Regeln geben Bedingungen an, unter denen ein bestimmter Plan ausgeführt werden kann. Dies kann etwa ein bestimmter Zeitpunkt, eine eingehende Nachricht, oder ein bestimmter Zustand der Welt oder des Agenten sein. Diese Regeln können freie Variablen enthalten, die gegen die aktuellen Beliefs des Agenten abgeglichen und dem Plan

als Parameter mitgegeben werden können. Auf welche Weise die Regeln beschrieben und implementiert sind, spielt hierbei keine entscheidende Rolle und wird nicht genauer spezifiziert.

Regel: Sei $(R,\ G,\ P)$ eine Rolle. Eine Regel $r \in R$ ist ein Tupel $(cond,\ plan,\ map)$. Hierbei ist *cond* eine Bedingung, die gegen die Beliefs des Agenten geprüft wird, und freie Variablen enthalten kann. $plan \in P$ ist der Plan, der bei Eintreffen der Bedingung ausgeführt werden soll, und *map* eine Relation, die sämtliche Eingabeparameter des Plans auf einen Wert oder eine der freien Variablen der Bedingung abbildet.

Ein weiterer Mechanismus um Pläne auszuführen sind Ziele. Ein Ziel definiert sich in ersten Linie durch eine Zielbedingung, die es zu erfüllen gilt. Hierfür können die Pläne der Rolle (oder eine Teilmenge derselben) angewendet werden. Welche Pläne verwendet und mit welchen Parametern sie aufgerufen werden kann über verschiedene Planungsalgorithmen bestimmt werden, die jedoch ebenso wie die zur Beschreibung der Zielbedingung verwendete Notation nicht Bestandteil dieser Arbeit sind.

Ziel: Sei $(R,\ G,\ P)$ eine Rolle. Ein Ziel $g \in G$ ist ein Tupel $(cond,\ P')$. Hierbei ist *cond* die Zielbedingung, die erfüllt werden soll, und P' eine Menge von Plänen, die für die Erfüllung des Zieles verwendet werden können.

Es gibt verschiedene Arten von Zielen. Interessant sind insbesondere die *Achieve-Goals*, bei denen ein bestimmter Zustand erreicht, und die *Maintain-Goals*, bei denen ein bestimmter Zustand aufrechterhalten werden soll [162]. Diese Arbeit beschränkt sich jedoch auf die Achieve-Goals. Maintain-Goals lassen sich auf einfache Art über Achieve-Goals und Regeln emulieren: Hierzu überprüft eine Regel die Goal-Condition des Maintain-Goals, und wenn diese nicht mehr erfüllt ist wird ein Plan aufgerufen, der ein Achieve-Goal mit derselben Goal-Condition erzeugt.

Agenten-Skript-Elemente

Die einzelnen Pläne der Agenten setzen sich aus Skript-Elementen zusammen. Diese können grob in zwei Kategorien unterteilt werden: atomare Aktionen und komplexe Skript-Elemente für den Kontrollfluss. Die Elemente des Kontrollflusses orientieren sich hierbei an typischen blockstrukturierten Sprachen.

Die Umsetzung in einem konkreten Agentenframework spielt für diese Arbeit nur eine untergeordnete Rolle: Die Skripte können direkt in Java-Code implementiert sein, oder

in einer höheren, agentenspezifischen Skriptsprache. In jedem Fall sollten die folgenden Elemente enthalten sein:

- *send*(m) Versendet eine Nachricht m mit gegebenem Inhalt und Empfänger.
- *receive*(m) Empfängt eine Nachricht, die dem Template m entspricht und speichert diese in m.
- *invoke*($p,\ i,\ o$) Ruft Plan mit ID p auf, mit Input i, und speichert den Output des Plans in o. i und o müssen den In- und Out-Listen des Plans entsprechen.
- *assign*($x,\ y$) Wertet den Ausdruck y aus und bindet das Ergebnis an die Variable x. Wenn x noch nicht existiert wird es erstellt.
- *achieve*(g) Fügt das Ziel g den Zielen des Agenten hinzu und wartet bis das Ziel erfüllt wurde. Nach Erfüllung wird das Ziel entfernt.
- *nop* Führt keine Aktion aus („No Operation“).

Des Weiteren werden mindestens die folgenden Kontrollfluss-Elemente benötigt:

- *seq*($s_1,\ \ldots,\ s_n$) Führt die Skript-Elemente $s_1,\ \ldots,\ s_n$ sequenziell aus.
- *par*($s_1,\ \ldots,\ s_n$) Führt die Skript-Elemente $s_1,\ \ldots,\ s_n$ parallel aus.
- *cond*($c,\ x,\ y$) Führt x aus, wenn Bedingung c erfüllt ist, andernfalls y.
- *while*($c,\ s$) Führt Skript-Element s aus, solange Bedingung c erfüllt wird.

Weitere Operationen zur Modellierung von Nebenläufigkeit, die für die Abbildung von Prozessmodellen interessant, jedoch nicht zwingend erforderlich sind:[1]

- *fork*($id,\ x$) Führt das Skript x in einem nebenläufigen Thread mit ID id aus.
- *join*(id) Wartet, bis der Thread mit ID id beendet ist.
- *stop*(id) Unterbricht den Thread mit ID id, sodass dieser sofort beendet.
- *wait*(t) Die Ausführung wird für den Zeitraum t ausgesetzt.

[1] Wie genau Nebenläufigkeit in dem Agenten umgesetzt wird – ob als echte Parallelität oder Pseudo-Parallelität – spielt für die Abbildung hierbei keine entscheidende Rolle.

Ausdrücke, Daten und Kommunikation

An verschiedenen Stellen im Agenten-Metamodell werden Ausdrücke benötigt, etwa für Zuweisungen, Bedingungen und Effekte, oder für Ziele und Regeln. In dem Metamodell wird keine spezielle Sprache für diese Ausdrücke festgelegt; stattdessen werden die Ausdrücke nur informell beschrieben. Eine Abbildung dieser Ausdrücke auf die in der Praxis gebräuchlichen Sprachen ist in der Regel ohne weiteres intuitiv möglich. Die Ausdruckssprache sollte natürlich die üblichen mathematischen und logischen Operationen unterstützen sowie Zugriff auf die Beliefs des Agenten und die Inputs und Outputs des aktuellen Plans ermöglichen. Die Beliefs sind hierbei einfache Tupel in der Form *id*, *value*), wobei die *id* Zugriff auf den entsprechenden *value* gibt.

Ein weiterer wichtiger Aspekt von Multiagenten-Systemen im Allgemeinen und für die Modellierung mittels Prozessen im Besonderen sind Nachrichten. Diese müssen neben dem eigentlichen Nachrichteninhalt insbesondere Informationen über die Absender und Empfänger der Nachricht enthalten.

Nachricht: Eine Nachricht m ist ein Tupel (*sender*, *receiver*, *content*). Hierbei ist *sender* die *id* des sendenden Agenten, *receiver* die *id* des empfangenden Agenten, und *content* der Inhalt der Nachricht, ein Datum.

Die *sender*-, *receiver*- und *content*-Attribute können hierbei ebenfalls in Ausdrücken verwendet werden, insbesondere in Zuweisungen. Der Typ des Inhalts unterliegt hierbei keinen Einschränkungen. Es kann sich ebenso um einen FIPA-Sprechakt handeln, wie um ein beliebiges anderes Datenobjekt. Absender und Empfänger der Nachricht sind Agenten-IDs; um auch Multicast-Nachrichten zuzulassen, kann der Empfänger der Nachricht alternativ auch der Name einer Nachrichtengruppe sein. In diesem Fall kann jeder Agent, der Nachrichten für die entsprechende Gruppe empfängt, die Nachricht erhalten.

6.2. Ausarbeitung des Prozess-Metamodells

Die Problematik einer fehlenden, allgemein anerkannten Definition der zentralen Abstraktion besteht bei Prozess-Modellen nicht: Zwar gibt es eine Vielzahl an möglichen Modellen und Notationen, und diese haben alle mehr oder weniger klar definierte Metamodelle, die als Orientierung dienen können. Dennoch wird an dieser Stelle ein eigenes, an BPMN angelehntes Prozess-Metamodell definiert. Zum einen bildet dieses zusammen mit dem Agenten-Metamodell eine einheitliche Basis für die Beschreibung der Abbil-

dung. Zum anderen wird das Modell geringfügig an die Möglichkeiten und Bedürfnisse der Abbildung auf Multiagenten-Systeme angepasst.

Die Grundlage des Prozess-Metamodells bildet das Business Process Model and Notation [118], das in Abschnitt 3.2 als die geeignetste Notation festgestellt wurde.

- BPMN ist eine standardisierte Notation mit hoher Verbreitung in der Praxis,
- BPMN ermöglicht die Modellierung auf hohem Abstraktionsniveau, und ist zugleich detailliert genug um daraus ausführbare Programme zu generieren, und
- BPMN verfügt über zahlreiche Sprachelemente, die für die Modellierung von Multiagenten-Systemen besonders relevant sind, beispielsweise Kommunikation.

Während bereits Arbeiten zur Formalisierung von BPMN durchgeführt wurden, etwa von Dijkman [42] und Ouyang [121], decken diese nur eine Teilmenge von BPMN ab; so fehlen etwa Aspekte wie Event Handler ebenso wie die Kommunikation zwischen Pools. Die BPMN-Spezifikation [118] wiederum ist in ihren Beschreibungen für die Belange dieser Arbeit zu nah an der Implementierung gehalten und enthält zahlreiche Elemente und Attribute, die für diese Arbeit nicht relevant sind.

Diese Arbeit beschränkt sich stattdessen auf diejenigen Elemente, die für die Abbildung auf Multiagenten-Systeme als am wichtigsten erachten werden können. Hierzu zählen auch einige Erweiterungen gegenüber der BPMN-Spezifikation, um etwa komplexere Kommunikation und dynamisches Verhalten modellieren zu können.

6.2.1. Konzeption und Zusammenstellung

Während BPMN viele Konzepte mit Multiagenten-Systemen gemeinsam hat und sich somit gut für deren Modellierung eignet, wurde sie doch für andere Zwecke entworfen: Einerseits zur Modellierung von Web-Service-Orchestrierungen mittels BPEL, und andererseits zur informellen Beschreibung von teil-automatisierten oder auch gänzlich manuellen Unternehmensprozessen.

Beides schlägt sich in der BPMN nieder: Zum einen sind einige Attribute stark auf die Beschreibung von Web-Services hin ausgerichtet; zum anderen ist die Notation mitunter nicht sehr streng und erlaubt verschiedene Möglichkeiten, bestimmte Sachverhalte zu modellieren, von denen in der Praxis nur wenige regelmäßig genutzt werden [187]. Hierdurch ergeben sich einige Elemente, die für die Abbildung auf Agenten nicht benötigt werden und die im Folgenden daher nicht betrachten werden, ohne dass die Ausdruckskraft hierunter leidet:

- Es werden nur *Collaboration Diagrams* betrachtet (die ursprüngliche, aus Version 1.x bekannte Form der BPMN-Diagramme), wobei das Gesamtsystem mehrere solcher Diagramme umfassen kann. *Conversation* und *Choreography Diagrams* stellen nur eine andere Ansicht dieser Prozesse dar und werden nicht betrachtet.
- Es wird angenommen, dass jeder Pool genau eine Lane hat; eine Unterteilung in mehrere Lanes wird nicht betrachtet.
- Aktivitäten und Ereignisse dürfen nur maximal einen eingehenden und ausgehenden Sequence Flow haben; d. h. für Verzweigungen und Zusammenführungen muss immer ein Gateway verwendet werden.
- Es wird eine klarere Trennung zwischen Tasks und Subprozessen verwendet. So sollen etwa nur Tasks über Assignments verfügen, während nur Subprozesse mit Event-Handlern versehen werden können. Die Ausdruckskraft wird hierdurch nicht eingeschränkt, da Tasks einfach in Subprozesse geschachtelt werden können.
- Activity-Looping wird nicht betrachtet; der gleiche Effekt kann über Kontrollfluss-Loops erzielt werden.
- Verschiedene Web-Service-spezifische Attribute werden nicht betrachtet.

Der Zweck dieser Einschränkungen liegt darin, sowohl das Metamodell als auch die sich daraus ergebende Abbildungsregeln zu vereinfachen; die Ausdruckskraft der Prozesse leidet hierunter nicht wesentlich (siehe auch Abschnitt 7.2 für mehr Hintergründe zu den Designentscheidungen). Prozessdiagramme, die den obigen Einschränkungen nicht genügen, können entsprechend „normalisiert“ werden, wofür in Abschnitt 8.2.3 entsprechende Transformationsregeln vorgestellt werden.

Zugleich gibt es einige Aspekte, die in Hinblick auf die beabsichtigte Abbildung auf Multiagenten-Systeme in dem Prozess-Metamodell hinzukommen, oder auf die mehr Gewicht gelegt wird als in anderen BPMN-Implementierungen üblich:

- Komplementär zu den üblichen Dienst-Aufrufen und Rückgabewerten wird das Nachrichten-Konzept um Nachrichten an einzelne Akteure sowie an Nachrichtengruppen erweitert. Auf diese Weise können komplexere Interaktionsprotokolle modelliert werden als mit Diensten allein.
- Aktivitäten und insbesondere Services sollen über semantische Informationen zu ihren *Inputs*, *Outputs*, *Preconditions* und *Effects* verfügen (IOPE). Diese können etwa zur semantischen Dienstauswahl und Planung verwendet werden.

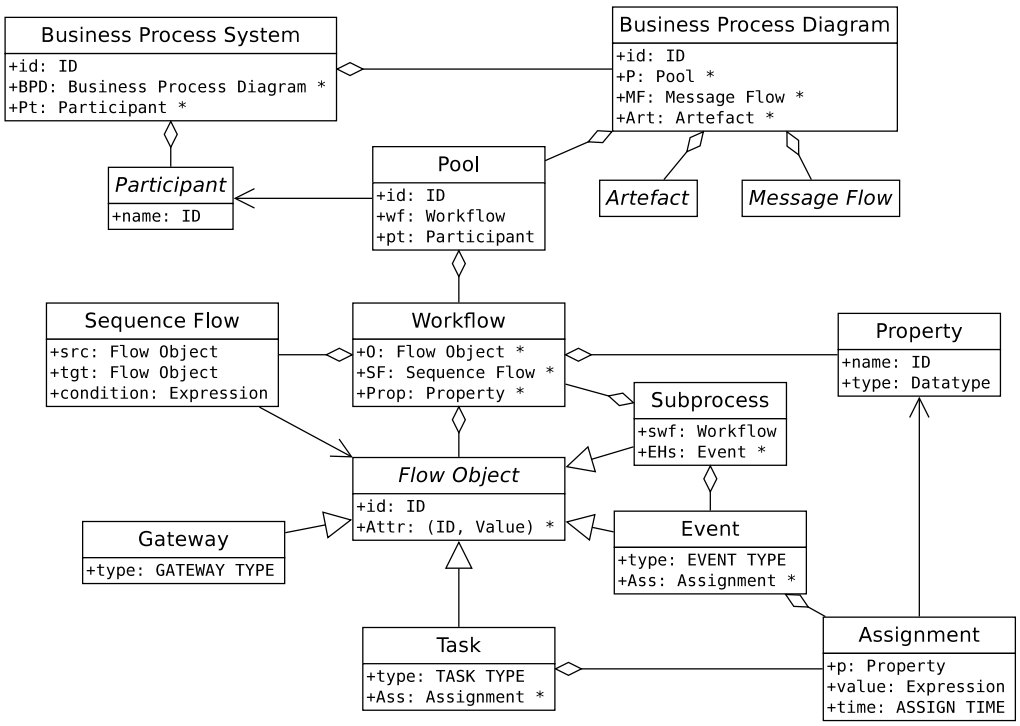


Abbildung 6.2.: Klassendiagramm des Prozess-Metamodells, geringfügig vereinfacht. [92]

6.2.2. Formale Beschreibung

Im Folgenden werden die verschiedenen Elemente des Prozess-Modells beschrieben [92]. Wie zuvor beim Agenten-Modell gliedert sich die Beschreibung in drei Teile: Die eigentlichen Prozess-Diagramme und deren Zusammenhang, die verschiedenen Elemente, aus denen die Prozesse zusammengesetzt sind, und Hilfskonstrukte für Daten, Nachrichten und ähnliches. Abbildung 6.2 zeigt das Prozess-Metamodell, geringfügig vereinfacht, in Form eines Klassendiagramms.

Prozesse, Pools und Participants

Zunächst wird ein Element für das gesamte Prozess System (BPS) definiert. Dieses System fasst die verschiedenen Prozessdiagramme und die Participants, die in diesen eine Rolle spielen, zusammen.

Business Process System: Ein BPS *bps* ist ein Tupel $(id,\ BPD,\ Pt)$. Hierbei ist BPD eine Menge von Business Process Diagrams und Pt eine Menge von Participants. id ist der eindeutige Bezeichner des BPS.

Diese Art von übergeordnetem Prozesssystem ist den Usecase-Diagrammen der UML ähnlich, wobei jeder Participant die Bezeichnung eines Actors und jedes Prozessdiagramm einem Usecase entspricht [93]. Mit Version 2 der BPMN wurden Conversation-Diagramme eingeführt, die einen ähnlichen Zweck erfüllen [118, Abschnitt 9.6].

Das Kernelement ist das Prozessdiagramm (Business Process Diagram, BPD). Ein Prozessdiagramm enthält eine Reihe von Pools und Nachrichtenflüssen (Message Flows). Ferner kann das BPD optional verschiedene Artifacts enthalten.

Business Process Diagram: Ein BPD *bpd* ist ein Tupel $(id,\ P,\ MF,\ Art)$. Hierbei ist P eine Menge von Pools, MF eine Menge von Message Flows und Art eine Menge von Artefakten. id ist der eindeutige Bezeichner des BPD.

Sowohl Artefakte als auch Message Flows haben hierbei keine eigenständige Bedeutung sondern dienen allein der Visualisierung. Die eigentliche Semantik ist in Attributen, beispielsweise der Send-Tasks, enthalten.

Pools sind definiert durch einen Workflow, sowie einen Participant, d. h. einen Verantwortlichen, der den im Pool beschriebenen Workflow ausführt. Auf eine weitere Unterteilung der Pools in verschiedene Lanes wird an dieser Stelle verzichtet.

Pool: Sei $bps = (id_1,\ BPD,\ Pt)$ ein BPS und sei $bpd = (id_2,\ P,\ MF,\ Art)$ ein BPD, sodass $bpd \in BPD$. Ein Pool $p \in P$ ist ein Tupel $(id,\ wf,\ pt)$. Hierbei ist wf ein Workflow und $pt \in Pt$ der Participant, der für die Ausführung des Workflows verantwortlich ist. id ist der eindeutige Bezeichner des Pools.

Der *Participant* definiert sich hierbei lediglich durch seine id und hat ansonsten keine weiteren Attribute. Der $Workflow$ definiert sich wie folgt:

Workflow: Ein Workflow wf ist ein Tupel $(O,\ SF,\ Prop)$. Hierbei ist O die Menge der Flow Objects, die direkt in dem Workflow enthalten sind, und $SF \subset O \times O \times (expressions \cup \{\varepsilon\})$ ist die Menge der Sequence Flows, über die die verschiedenen Flow Objects verbunden sind, wobei jedem Sequence Flow eine Bedingung (oder ε, d. h. keine Bedingung) zugeordnet wird. $Prop$ ist eine Menge von Properties (Variablen), die im Gültigkeitsbereich des Workflows deklariert werden.

Die Menge der Flow Objects O ist hierbei Unterteilt in die disjunkten Teilmengen T, S, E und G der Tasks, Subprocesses, Events und Gateways, deren Elemente im Folgenden näher beschrieben werden.

Workflow-Elemente: Flow-Objects

Die Workflow-Elemente der Pools, oder „Flow Objects“, setzen sich aus Aktivitäten (*Tasks* und *Subprocesses*), *Events* und *Gateways* zusammen. Die meisten dieser Elemente haben die Form (id, $type$, Ass, $Attr$), wobei Ass eine Menge von Zuweisungen, über die Properties im Gültigkeitsbereich des Workflows mit Werten belegt werden können, und $Attr$ eine Menge von zusätzlichen Attributen ist, die vom jeweiligen Typen $type$ abhängen. id ist ein eindeutiger Bezeichner.

Die wichtigsten Flow Objects sind die Aktivitäten, die sich wiederum in Tasks und Subprozesse unterteilen lassen. Tasks stehen hierbei für „atomare“ Aktivitäten, wie das Versenden einer Nachricht, oder einen Service aufzurufen.

Task: Sei $wf = (O, SF, Prop)$ ein Workflow. Ein Task $t \in O$ ist ein Tupel (id, $type$, Ass, $Attr$). Hierbei ist $type \in \{\text{SERVICE}, \text{SEND}, \text{RECEIVE}, \text{SCRIPT}, \dots\}$ der Typ des Tasks, $Ass \subset Prop \times expressions \times \{\text{BEFORE}, \text{AFTER}\}$ eine Menge von Assignments (Zuweisungen), und $Attr \subset ID \times expression$ eine Menge von typabhängigen Attributen. id ist der eindeutige Bezeichner des Tasks.

Subprozesse fassen mehrere Tasks und andere Flow-Objects zu komplexeren Strukturen zusammen. Neben der Gruppierung zusammengehörender Elemente bieten diese insbesondere die Möglichkeit, mit Event Handlern versehen zu werden, womit diese beim Eintreffen bestimmter Ereignisse unterbrochen und der Prozess an anderer Stelle fortgesetzt werden kann.

Subprocess: Sei $wf = (O, SF, Prop)$ ein Workflow. Ein Subprocess $sub \in O$ ist ein Tupel (id, swf, EH, $succ_{EH}$, $Attr$). Hierbei ist swf der Workflow innerhalb des Subprocess. EH ist eine Menge von Events, durch die der Subprocess unterbrochen wird (Event Handler), $succ_{EH} \subset EH \times O$ eine Relation, über die jedem der Events in EH ein Nachfolgeelement aus O zugewiesen wird, und $Attr \subset ID \times expression$ eine Menge von zusätzlichen Attributen. id ist der eindeutige Bezeichner des Subprocess.

Neben den Aktivitäten gibt es zudem verschiedene Typen von *Events*.

Event: Sei $wf = (O, SF, Prop)$ ein Workflow. Ein Event $e \in O$ ist ein Tupel (id, $type$, Ass, $Attr$). Hierbei ist $type \in \{\text{MESSAGE}, \text{TIMER}, \text{RULE}, \dots\}$ der Typ des Events, $Ass \subset Prop \times expressions \times \{\text{BEFORE}, \text{AFTER}\}$ eine Menge von Assignments (Zuweisungen), und $Attr \subset ID \times expression$ eine Menge von typabhängigen Attributen. id ist der eindeutiger Bezeichner des Events.

Intuitiv sind Aktivitäten „aktiv“ und Events „passiv“, doch diese Unterscheidung trifft nicht immer zu. So gibt es etwa auch *throwing* Events, die eine Nachricht verschicken, oder einen Fehler auslösen. Insbesondere beim Versenden und Empfangen von Nachrichten können meist sowohl Events als auch Aktivitäten verwendet werden. Als Faustregel sollten Events verwendet werden, wenn ein Prozess durch eine Nachricht ausgelöst wird oder es sich um die Ein- und Ausgaben eines Services handelt, sowie für Event-Handler und Event-based XOR-Gateways, und ansonsten Aktivitäten.

Schließlich gibt es noch die *Gateways*, mit denen der Workflow aufgespalten, umgelenkt und zusammengeführt werden kann. In der Praxis sind Gateways oft optional, helfen aber dabei, die Struktur des Workflows visuell zu erfassen. In dieser Arbeit wird daher vorausgesetzt, dass Gateways die einzigen Elemente mit mehreren ein- oder ausgehenden Sequence Flows sind. Dies vereinfacht die Abbildung, ohne die Ausdruckskraft einzuschränken.[2]

Gateway: Sei $wf = (O,\ SF,\ Prop)$ ein Workflow. Ein Gateway $g \in O$ ist ein Tupel $(id,\ type,\ Attr)$. Hierbei ist $type \in \{\text{XOR}, \text{OR}, \text{AND}, \text{EVENT}, \text{COMPLEX}\}$ der Typ des Gateways, und $Attr \subset ID \times expression$ eine Menge von typabhängigen Attributen. id ist der eindeutiger Bezeichner des Gateways.

Gateways können weiter in *verzweigende* und *zusammenführende* Gateways unterschieden werden; dies ergibt sich implizit aus der Anzahl der ein- und ausgehenden Sequence Flows.

Ausdrücke, Daten und Kommunikation

Analog zum Agenten-Metamodell wird auch für das Prozess-Metamodell keine spezielle formale Ausdruckssprache festgelegt; stattdessen werden Ausdrücke, etwa für Zuweisungen und Bedingungen, informell und intuitiv verständlich beschrieben. Wie schon zuvor soll die Sprache die üblichen mathematischen und logischen Operationen unterstützen und – über deren eindeutige ID – den Zugriff auf die verschiedenen Prozess-Properties (Variablen) erlauben.

Darüber hinaus gibt es zwei wichtige Hilfskonstrukte: Services und Nachrichtenkanäle. Nachrichtenkanäle definieren sich hierbei ganz analog zum Agenten-Metamodell.

[2] Ein Workflow, bei dem dies nicht der Fall ist, lässt sich auf einfache Weise in eine gültige Form überführen; siehe Abschnitt 8.2.3.

Nachricht: Eine Nachricht m ist ein Tupel (*sender*, *receiver*, *content*). Hierbei ist *sender* die *id* des sendenden Partners, *receiver* die *id* des empfangenden Partners, und *content* der Inhalt der Nachricht.

Die Attribute *sender*, *receiver* und *content* können hierbei ebenso wie andere Properties auch in Ausdrücken und Zuweisungen verwendet werden, um etwa zur Laufzeit den Absender einer Nachricht auszulesen oder deren Adressaten zu bestimmen.

Neben den Nachrichten wird außerdem noch eine Beschreibung für Dienste, oder Services, benötigt. Im Agenten-Metamodell erfüllen die Pläne diese Aufgabe; im Prozess-Metamodell muss jedoch nicht jeder Pool einem Dienst entsprechen. Diese Zuordnung ergibt sich erst durch die Attribute der jeweiligen Start-Ereignisse.

Service: Ein Service s ist ein Tupel (*id*, *provider*, *Input*, *Output*, *pre*, *eff*). Hierbei ist *provider* der Participant, der den Service anbietet und *input* und *output* die Ein- und Ausgabeparameter des Dienstes. Optional können Services auch semantische Beschreibungen ihrer Vorbedingungen *pre* und Effekte *eff* haben. *id* ist der eindeutige Bezeichner des Dienstes.

Ebenso wie bei den Nachrichten bleibt die Implementierung der Services hierbei offen. Services wie auch Nachrichten können über verschiedene Technologien realisiert werden. Bei der Abbildung auf Agenten muss ggf. eine entsprechende Brückentechnologie genutzt werden.

6.3. Beispiel

In diesem Abschnitt sollen die Metamodelle anhand eines Beispiels veranschaulicht werden. Hierzu wird ein einfaches Prozessdiagramm skizziert und das diesem entsprechende Prozess-Modell beschrieben. Anschließend wird das äquivalente Agenten-Modell aufgezeigt. Hierbei werden implizit bereits Teile der Abbildung von Prozessen auf Agenten vorweg genommen, die im nächsten Kapitel ausführlich beschrieben wird.

Der Beispielprozess besteht aus einem einzelnen Prozessdiagramm (siehe Abbildung 6.3), das eine Interaktion zwischen zwei Participants darstellt, von denen der eine die Interaktion initiiert und der andere auf die erste Nachricht reagiert und darauf antwortet. Das Beispiel ist der Übersichtlichkeit und Verständlichkeit halber bewusst sehr einfach gehalten, enthält jedoch die wichtigsten Elemente.

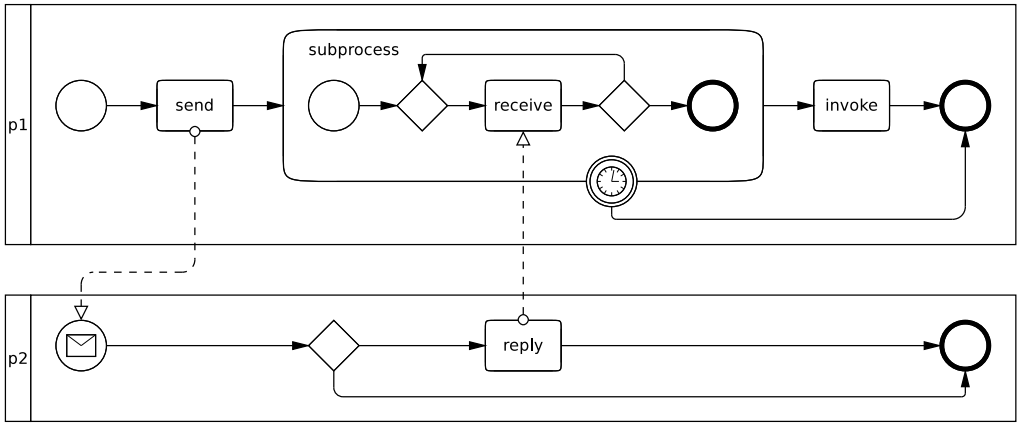


Abbildung 6.3.: BPMN-Beispieldiagramm zur Illustration der Modelle.

6.3.1. Prozess-Modell

Das gesamte Business Process System *bps* besteht nur aus einem einzelnen Prozessdiagramm *bpd* mit zwei Pools p_1 und p_2 die von den zwei am Prozess beteiligten Participants pt_1 und pt_2 ausgeführt werden. Ein weiterer Participant, pt_3, ist für die Ausführung des aufgerufenen Services zuständig und im Prozessdiagramm nicht dargestellt.

- $bps = (\text{'bps'},\ \{bpd\},\ \{pt_1,\ pt_2,\ pt_3\})$
- $bpd = (\text{'bpd'},\ \{p_1,\ p_2\},\ \{mf_1,\ mf_2\},\ \varnothing)$
- $p_1 = (\text{'p1'},\ wf_1,\ pt_1),\ \ p_2 = (\text{'p2'},\ wf_2,\ pt_2)$

Im oberen Pool p_1 wird zunächst eine Nachricht an den zweiten Participant geschickt. Danach betritt der Prozess einen Subprozess, in dem in einer Endlosschleife Antworten auf die Nachricht empfangen werden, bis ein Timeout zuschlägt.[3] Anschließend wird noch ein Dienst aufgerufen.

- $wf_1 = (\{s_1,\ send,\ sub,\ inv,\ e_1\},\ \{(s_1,\ send,\ \varepsilon),\ \ldots,\ (inv,\ e_1,\ \varepsilon)\},\ props_1)$, wobei $props_1 = \{best\}$ und $sub = (\text{'sub'},\ wf_{sub},\ \{timer\},\ \{(timer,\ inv)\},\ \varnothing)$
- $wf_{sub} = (\{s_{sub},\ g_1,\ receive,\ g_2,\ e_{sub}\},\ \{\ldots,\ (g_2,\ g_1,\ \top),\ \ldots\},\ \varnothing)$

Die eigentliche Semantik des Prozesses ist in den Flow Objects enthalten, von denen die interessantesten *send*, *receive*, *inv* und *timer* sind. Jedes dieser Elemente ist ein Tupel in der Form $(id,\ type,\ Ass,\ Attr)$.

[3] Diese Schleife ließe sich auch kompakter darstellen, mit einem Looping Subprocess; das vereinfachte Modell in dieser Arbeit beschränkt sich jedoch auf Kontrollfluss-Schleifen.

- $send =$ ('send', SEND, ass_{send}, {('impl', msg_1)}),
  wobei $ass_{send} = \{(msg1_{cnt},$ 'auftrag', BEFORE), $(msg1_{rec},$ 'msg_grp', BEFORE)$\}$
- $receive =$ ('receive', RECEIVE, ass_{rec}, {('impl', msg_2)}),
  wobei $ass_{rec} = \{(best,\ max(msg2_{cnt},\ best),$ AFTER$)\}$
- $inv =$ ('invoke', SERVICE, $\{(service_{in1},\ best,$ BEFORE$)\}$, {('impl', $service$)})
- $timer =$ ('timer', TIME, $\varnothing$, {('timeout', $3s$)})

Der untere Pool p_2 wird durch das Empfangen einer Nachricht gestartet. Daraufhin wird die Nachricht überprüft und gegebenenfalls eine Antwort geschickt.

- $wf_2 = (\{s_2,\ g_3,\ reply,\ e_2\},\ \{\ldots,\ (g_3,\ reply,\ is_ok(order)),\ \ldots\},\ props_2)$,
  wobei $props_2 = \{sender,\ order\}$
- $s_2 =$ ('s2', MESSAGE, ass_{s2}, {('impl', msg_1)}) ,
  wobei $ass_{s2} = \{(order,\ msg1_{cnt},$ AFTER$),\ (sender,\ msg1_{snd},$ AFTER$)\}$
- $reply =$ ('reply', SEND, ass_{reply}, {('impl', msg_2)}),
  wobei $ass_{reply} = \{(msg2_{rec},\ sender,$ BEFORE$),\ (msg2_{cnt},$ 'angebot', BEFORE$)\}$

Schließlich die Elemente zur Beschreibung der Nachrichten und des Services:

- $msg_1 = (msg1_{snd},\ msg1_{rec},\ msg1_{cnt}),\ msg_2 = (msg2_{snd},\ msg2_{rec},\ msg2_{cnt})$
- $service =$ ('service', pt_3, $\{service_{in1}\}$, $\varnothing$, ε, ε)

6.3.2. Agenten-Modell

Ein diesem Prozessdiagramm äquivalentes Multiagenten-System könnte wie folgt aussehen: Jeder Participant entspricht eine Rolle, die von je einem Agenten implementiert wird. Die Pools und die darin enthaltenen Prozesse entsprechen jeweils einem Plan dieser Rollen, die miteinander interagieren. Das Nachrichten-Start-Event des zweiten Pools kann über eine entsprechende Reaktionsregel beschrieben werden. Die weiteren Attribute – Beliefs und Goals – sind initial leer.

- $mas =$ ('mas', {('agent1', $\{role_1\}$, $\varnothing$), ('agent2', $\{role_2\}$, $\varnothing$)}, $\{role_1,\ role_2\}$)
- $role_1 =$ ('role1', $\{plan_1\}$, $\{rule_1\}$, $\varnothing$), $role_2 =$ ('role2', $\{plan_2\}$, $\{rule_2\}$, $\varnothing$)

Genau wie im Pool p_1 beginnt der Plan $plan_1$ damit, dass eine Nachricht versandt wird. Das Verhalten des Subprozesses kann über das $fork$ Konstrukt umgesetzt werden: Sowohl der Subprozess selbst als auch das Timer-Ereignis werden vom Haupt-Thread abgezweigt; wenn der Timer abgelaufen oder der Subprozess beendet ist wird der jeweils andere Thread unterbrochen. Abschließend wird wie im Pool ein Dienst aufgerufen.

- $rule_1 = (\text{‘rule1’}, \text{‘Agent wurde gestartet’}, plan_1, \varnothing)$
- $plan_1 = (\text{‘plan1’}, \varnothing, \varnothing, \varepsilon, \varepsilon, script_1)$
- $script_1 = seq(snd, sub, timer, join(\text{‘sub’}), stop(\text{‘timer’}), invk)$, wobei
  - $snd = send(m)$, wobei $m = (self, role_2, \text{‘angebot’})$
  - $sub = fork(\text{‘sub’}, while(\top, receive(m), assign(best, max(m_{cnt}, best))))$, wobei $m = (*, self, \text{‘offer’})$
  - $timer = fork(\text{‘timer’}, seq(wait(\text{‘3s’}), stop(\text{‘sub’})))$
  - $invk = invoke(\text{‘service’}, [best], [])$

Das Verhalten des Plans $plan_2$ ist abermals deutlich einfacher: Nachdem der Plan durch eine entsprechende Reaktionsregel aufgerufen wurde überprüft dieser lediglich die Bedingung und verschickt gegebenenfalls die Antwortnachricht.

- $rule_2 = (\text{‘rule2’}, \text{‘empfange } m = (*, self, \text{‘angebot’})\text{’}, plan_2, \{(m, msg)\})$
- $plan_2 = (\text{‘plan2’}, [msg], [], \varepsilon, \varepsilon, script_2)$
- $script_2 = cond(is_ok(msg_{cnt}), send((self, msg_{snd}, \text{‘angebot’})), nop)$

In diesem Beispiel wurden bereits Teile der Abbildung von Prozessmodellen auf Multiagenten-Systeme angewendet, die Thema dieser Arbeit ist. Im nächsten Kapitel soll diese Abbildung vollständig und detailliert beschreiben werden.

Kapitel 7.

Konzeption und Ausarbeitung der Abbildung von Prozessen auf Agenten

Die Abbildung von BPMN auf JIAC-Agenten wurde über einen längeren Zeitraum und im Rahmen verschiedener Projekte entwickelt [45, 46, 86, 88, 89, 90, 91, 92, 93], wobei sukzessive weitere Konzepte hinzu kamen.

Ein erster Schritt bestand darin, die bestehende Abbildung von BPMN auf BPEL auf die Skriptsprache *JADL* der damaligen Version 4 des JIAC-Agentenframeworks zu übertragen [86]. Auch gab es erste Versuche, die Abbildung zu formalisieren [45, 46], wobei jedoch nur ein kleiner Teil der BPMN betrachtet wurde, insbesondere kein Event-Handling. Kurze Zeit später wurde JIAC IV jedoch eingestellt, wodurch diese Version der Abbildung nie in der Praxis zum Einsatz kam.

Stattdessen wurde die Abbildung auf die neue Version 5 des JIAC-Frameworks und dessen Skriptsprache *JADL++* übertragen [93]. Im Zuge dessen wurde die Abbildung auch um zusätzliche Konzepte erweitert. Insbesondere konnten nun nicht mehr nur einzelne BPMN-Diagramme, sondern Systeme mehrerer zusammenhängender Prozesse abgebildet werden, wobei die resultierenden Pläne den jeweiligen Rollen zugeordnet werden. Außerdem wurden nun auch Start-Events auf Regeln abgebildet, um die entsprechenden Pläne beim Eintreten eines Ereignisses auszulösen, etwa bei einer eingehenden Nachricht oder zu einem bestimmter Zeitpunkt.

Später entstand eine neue Implementierung der Abbildung, die statt *JADL++*-Skripten direkt JIAC Agenten-Beans erzeugt [153]. Aufgrund der eher geringen Verbreitung von JADL++ avancierte diese bald zur neuen Standard-Implementierung der Abbildung [89, 90]. Im Rahmen dieser Version kamen neben einem verbesserten Event-Handling, das in dieser Form in JADL++ zuvor nicht möglich gewesen wäre, auch weitere Möglichkeiten zum Versenden und Empfangen von Nachrichten hinzu, wodurch nun auch komplexere Interaktionsprotokolle modelliert und abgebildet werden können.

Zuletzt wurde die Abbildung in Form einer JIAC-AgentBean umgesetzt, mit der die BPMN-Prozesse direkt interpretiert werden können [164]. Diese Implementierung unterscheidet sich insofern von den anderen, als dass die Prozesse nicht strukturiert werden müssen; stattdessen folgt der Interpreter direkt der durch die Sequence Flows festgelegten Ausführungsreihenfolge [90]. Ein weiterer Unterschied ist, dass die Prozesse sämtliche zur Ausführung nötigen Informationen enthalten müssen, da es keinen generierten Quellcode gibt, der angepasst oder ergänzt werden könnte.

Schließlich wurde die Abbildung, die diesen verschiedenen Implementierungen zugrunde liegt, sowie die im vorigen Kapitel beschriebenen Metamodelle für Agenten und Prozesse formal beschrieben [91, 92], mit einem Schwerpunkt auf der Übertragung der Prozessstrukturen in entsprechende Agenten-Programme.

Im Folgenden wird die Abbildung von Prozessmodellen auf Multiagenten-Systeme zunächst auf relativ abstrakte und einfach verständliche Art beschrieben. Anschließend werden verschiedene Aspekte der Abbildung und die dahinter stehenden Designentscheidungen näher erläutert sowie mögliche Erweiterungen diskutiert. Abschließend wird die Abbildung anhand der im vorigen Kapitel beschriebenen Metamodelle für Prozesse und Agenten formal beschrieben.

7.1. Überblick und Konzeption

In diesem Abschnitt soll die Abbildung von BPMN auf Multiagenten-Systeme auf konzeptioneller Ebene beschrieben werden (siehe [93]). Abbildung 7.1 stellt eine Übersicht über die verschiedenen Konzepte dar.

- Die Pools in den verschiedenen BPMN-Diagrammen werden auf Agenten-Pläne abgebildet; die Tasks und Events innerhalb des Pools werden auf entsprechende Skript- und Kontrollfluss-Elemente abgebildet.
- Die Start-Events werden auf Regeln und Mechanismen zum Ausführen der jeweiligen Pläne in den entsprechenden Situationen abgebildet.
- BPMN-Participants werden auf Agentenrollen abgebildet, deren Verhalten durch die Pläne und Regeln bestimmt ist, die aus den entsprechenden Pools abgeleitet wurden.

In den folgenden Unterabschnitten wird jeder dieser Aspekte näher erläutert. Die Abbildung ist hierbei weitgehend unabhängig von der verwendeten Implementierung und

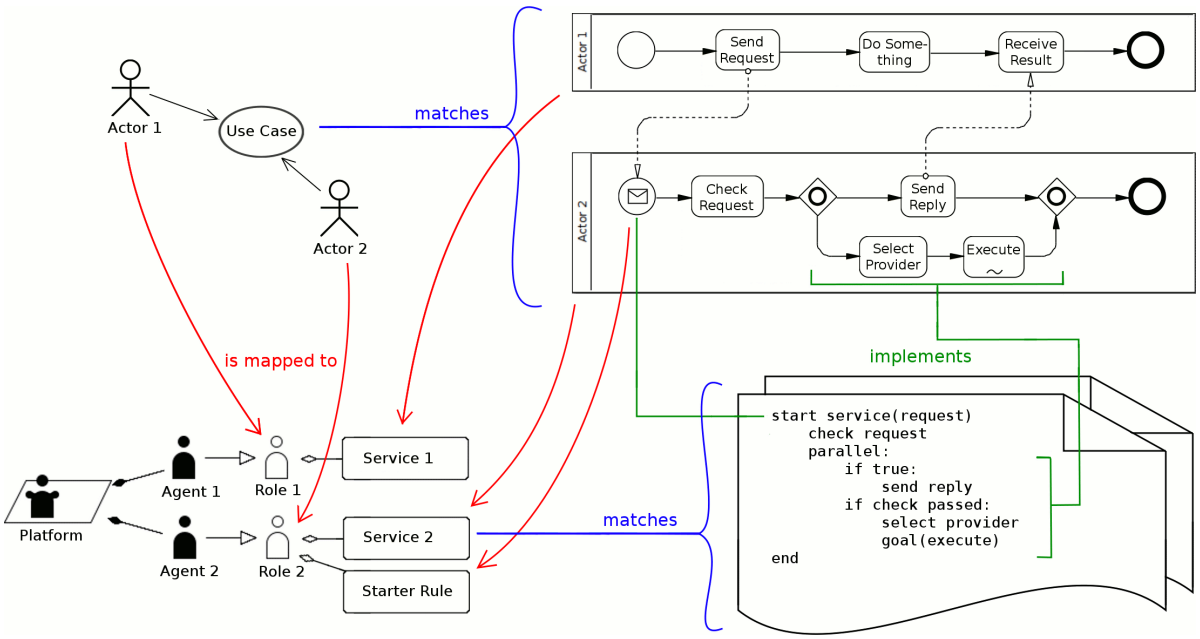


Abbildung 7.1.: Überblick über die Abbildung vom Prozessmodell auf Agenten [93]: Participant zu Rolle, Prozess zu Plan, Start-Ereignis zu Regel.

kann für verschiedene Agenten-Frameworks und als Code-Generierung oder als Interpreter umgesetzt werden. Mehrere Referenzimplementierungen mit individuellen Vor- und Nachteilen werden in Kapitel 8.3 vorgestellt.

7.1.1. Von Prozessen zu Agenten-Verhalten

Den Kernaspekt stellt die Abbildung der BPMN-Prozessdiagramme auf das Verhalten der Agenten dar: die einzelnen Pläne der verschiedenen Agenten. Im Prinzip ist es möglich, die gesamte Logik der Pläne in den BPMN-Diagrammen darzustellen; praktikabler ist es jedoch, sich auf die wichtigsten Aspekte des Kontrollflusses und vor allem die Kommunikation zu beschränken. Hier bietet die Verwendung von BPMN den größten Nutzen, während die Modellierung aller Details eines Algorithmus schnell zu übergroßen und unübersichtlichen Prozessdiagrammen führen würde. Diese Aspekte können stattdessen per Hand in den generierten Plänen ergänzt oder in Form von bestehenden Plänen, Diensten, oder Bibliotheksfunktionen eingebunden werden.

Kontrollfluss innerhalb der Pools

Die einzelnen Pools innerhalb eines Prozessdiagramms werden auf Pläne abgebildet, die von der jeweiligen Rolle angeboten werden. Hierbei wird für jeden Pool ein Plan erstellt.[1] Properties des Pools werden auf Variablen abgebildet, die für den Scope des gesamten Plans definiert sind. Der Workflow innerhalb des Pools wird auf die eigentliche Logik im Rumpf des Plans abgebildet. Eine mögliche weitere Unterteilung in Lanes wird hierbei nicht beachtet.

Vor der eigentlichen Abbildung einzelner Elemente des Workflows innerhalb des Pools auf Elemente innerhalb eines Plans muss zunächst die Struktur des Workflow in die der Plan-Beschreibungssprache übertragen werden. Während die Details dieser Transformation von der Art der Implementierung abhängen, bedeutet dies in der Regel, dass der Workflow, welcher ein gerichteter Graph ist, in eine äquivalente Blockstruktur übertragen werden muss. Dies kann beispielsweise über eine Reihe von Graphtransformationsregeln geschehen, und ist im Wesentlichen äquivalent zu dem Vorgehen bei der Transformation von BPMN nach BPEL, wie es in Abschnitt 3.3 ausführlich beschrieben ist. Die in dieser Arbeit verwendete Struktur-Transformation wird im Abschnitt 8.2.3 beschrieben.

Die eigentlichen Workflow-Elemente unterteilen sich nun in Aktivitäten (Tasks und Subprocesses), Ereignisse (Events) und Verzweigungen (Gateways). Gateways dienen der Steuerung des Kontrollflusses und werden bei der Abbildung der Struktur des Workflows auf die des Plans in Form von Sequenzen, bedingten Verzweigungen, Schleifen und parallel ausgeführte Segmente umgesetzt. Was bleibt sind die Aktivitäten und Ereignisse, deren Abbildung in den folgenden Abschnitten erläutert wird.

Aktivitäten

Über Aktivitäten ist beschrieben, was der Agent in Laufe des Prozesses *tut*. Sie unterteilen sich in atomare *Tasks* und komplexe *Subprocesses*, wobei Tasks weiter nach Typen unterschieden werden können. Die wichtigsten Typen sind *Send*-, *Receive*-, *Service*- und *Script*-Tasks.

Service-Tasks werden auf Dienstaufrufe übertragen, und Send- und Receive-Tasks werden naturgemäß auf die Fähigkeit des Agenten, Nachrichten zu versenden und zu empfangen abgebildet; mehr hierzu weiter unten im Abschnitt zur Kommunikation. Über

[1] Genauer: Es wird ein Plan für jedes Start-Event des Pools erzeugt, wobei ein Pool i. d. R. genau ein Start Event hat. Der Plan deckt denjenigen Teil des Prozesses ab, der beginnend ab diesem Start-Event erreichbar ist. Verfügt ein Pool über mehrere Start-Events, so können sich Teile der erzeugten Pläne überschneiden.

den Script-Task lassen sich kurze Code-Fragmente angeben, die direkt in den Plan integriert werden können; er stellt somit eine Art „Joker“ dar, über den sich andernfalls nicht unterstützte Funktionen ansprechen lassen.

Bei der Abbildung der Subprozesse bieten sich zwei Möglichkeiten an: Entweder, der Inhalt des Subprozesses wird direkt in den umgebenden Plan eingefügt, oder der Subprozess wird auf eine Subroutine abgebildet, die an der entsprechenden Stelle des Plans aufgerufen wird. Die Auslagerung in eine Subroutine erscheint als die sauberere Alternative, insbesondere wenn der Subprozess über eine Referenz an anderer Stelle noch einmal verwendet wird. Es ist jedoch notwendig, dass der Subprozess vollen Zugriff – lesend wie schreibend – auf die Properties (die Variablen) des ihn umgebenden Prozesses haben muss.

Ist eine Aktivität mit einem *Event Handler* ausgestattet, so muss auch dies entsprechend auf die Logik des Plans übertragen werden, sofern ein solches Verhalten von der Beschreibungssprache unterstützt wird. Der einfachste Fall eines Error-Event-Handlers kann in der Regel einfach auf Strukturen zum Exception-Handling abgebildet werden. Für andere Arten von Ereignissen, etwa eintreffende Nachrichten, müssen spezielle Observer genutzt werden, die das jeweilige Ereignis überwachen und den Hauptprozess ggf. unterbrechen und/oder umleiten.

Ereignisse

Als Ereignisse, oder Events, werden Dinge bezeichnet, die während des Prozesses *passieren* können. Sie können zur Laufzeit des Systems auftreten und einzelne Prozesse auslösen oder deren Verlauf ändern, können aber auch von den Prozessen selbst erzeugt werden. Von den verschiedenen in der BPMN spezifizierten Event Typen werden von der hier vorgestellten Abbildung nur einige unterstützt, die zugleich jedoch auch die interessantesten Ereignisse darstellen: Das *Message Event*, das *Timer Event* und das *Rule Event*.

Wie weiter oben bereits erwähnt wird für jedes Start Event, über das der Workflow in einem Pool angestoßen werden kann, ein separater Plan erzeugt, dessen Logik den Teil des Workflows *ab diesem Start Event* entspricht. (Sollten die von zwei Start Events ausgehenden Workflows an einem späteren Punkt zusammenlaufen, kann der gemeinsame Teil der Workflows in eine gemeinsam genutzte Subroutine ausgelagert werden, doch dies ist ein Implementierungsdetail.)

Das Start Event bestimmt hierbei auch die Parameter des entsprechenden Plans. Bei einem Message-Start-Event entsprechen die Eingabeparameter etwa dem Payload der

Message, also der mit der Nachricht verschickten Information. Im Falle eines Rule-Start-Events werden alle Properties des Prozesses, die als Variablen im Regel-Ausdruck verwendet werden, zu Eingabeparametern. Pläne, die durch ein Timer-Start-Event angestoßen werden, ebenso wie solche mit untypisiertem Start-Event, haben keine Eingabe-Parameter. Analog zum Message-Start-Event bestimmt ein Message-End-Event, sofern vorhanden, die Rückgabewerte des Plans.

Als Intermediate-Event eingesetzt, wird ein Message-Event äquivalent zu einem Send- bzw. Receive-Task abgebildet, je nachdem, ob es sich um eine eingehende oder ausgehende Nachricht handelt. Ein Intermediate-Timer-Event unterbricht den Plan für die angegebenen Zeit, und ein Intermediate-Rule-Event bis die angegebene Regel als *wahr* ausgewertet wird.

Schließlich kann ein Event auch als Event-Handler an eine Aktivität angefügt werden, wodurch die Aktivität – üblicherweise ein Subprozess – bei Eintreffen des Ereignisses unterbrochen und der Kontrollfluss an anderer Stelle fortgeführt wird. Dieses Konzept ist für die Abbildung besonders wertvoll, da sich hiermit Verhaltensweisen modellieren lassen, die auf traditionelle Weise oft nur schwer umzusetzen sind. Solche Verhaltensweisen werden auf Observer abgebildet, die parallel zum eigentlichen Subprozess ausgeführt werden und diesen ggf. unterbrechen.

Kommunikation

Der größte Vorteil von BPMN gegenüber vielen anderen Prozessbeschreibungssprachen ist die explizite Darstellung der Kommunikation zwischen den verschiedenen Akteuren. Diese kann sehr gut auf die Kommunikation zwischen den Agenten abgebildet werden.

Durch die Nähe zu BPEL und Webservices stellt in BPMN jeder Kommunikationspfeil einen Dienstaufruf dar. Für die Abbildung auf Agenten wurde das Metamodell um ein weiteres Element ergänzt: den Nachrichtenkanal, definiert durch einen Empfänger (ein Agent oder ein Nachrichtenkanal) und Absender sowie die zu verschickenden Objekte. Mit diesen Nachrichtenkanälen ist die Kommunikation nicht auf Dienstaufrufe und deren Rückgabewerte beschränkt.

Für Service-Tasks werden wie gehabt die ursprünglichen Dienstbeschreibungen der BPMN-Spezifikation verwendet, um von anderen Agenten angebotene Pläne oder Aktionen aufzurufen und deren Resultate entgegen zu nehmen. Send- und Receive-Tasks hingegen werden mit einem Nachrichtenkanal assoziiert, auf dem diese Nachrichten senden bzw. empfangen sollen. Message-Events können sowohl mit Diensten als auch mit Nachrichtenkanälen assoziiert werden: Je nachdem, ob die Message-Start- und -End-

Events, die den Anfang und das Ende eines Prozesses markieren, mit einem Dienst oder einem Nachrichtenkanal assoziiert sind, wird der Prozess über einen Dienstaufruf oder durch eine Nachricht auf dem entsprechenden Nachrichtenkanal gestartet.

Eine weitere Besonderheit von BPMN stellt das Event-basierte XOR-Gateway dar, das benutzt werden kann, um auf verschiedenen Nachrichtenkanälen zugleich auf eingehende Nachrichten zu warten und in Abhängigkeit davon, welche Nachricht zuerst eintritt, dem einen oder dem anderen Pfad zu folgen – auch in Verbindung mit einem Timeout. Abgebildet auf das Verhalten der Agenten können auf diese Weise komplexe Interaktionsprotokolle modelliert werden, beispielsweise für Auktionen.

Daten und Zuweisungen

Auf den ersten Blick spielen Variablen und Zuweisungen in BPMN keine große Rolle. Es gibt keine eigenen graphischen Elemente für Variablen und Zuweisungen; stattdessen kann jede Aktivität und jedes Event beliebig viele Zuweisungen enthalten, die zusammen mit der Aktivität bzw. dem Event ausgeführt werden. Variablen (in BPMN: Properties) können für einen Prozess als ganzes oder für einzelne Aktivitäten definiert werden. Während die eigentliche Logik des Prozesses so in den Diagrammen versteckt ist, bleiben auf diese Weise auch komplexe Abläufe übersichtlich. Auch wird auf diese Weise explizit gemacht, zu welchen Aktivitäten verschiedene Gruppen von Zuweisungen gehören, beispielsweise als Vor- und Nachbereitung eines Dienstaufrufs.

In der Abbildung auf Pläne und Aktionen von Agenten werden die Prozess-Variablen auf Variablen im Namensraum des Plans und ggf. auf Parameter und Rückgabewerte abgebildet.

Dynamik und Planung

Ein besonderes Charakteristikum von Agenten und Multiagenten-Systemen sind Aspekte wie zielgerichtetes und deliberatives Verhalten und Planung. Naturgemäß spielt dieses in Prozessdiagrammen keine wesentliche Rolle, schließlich besteht der Zweck einer Prozessbeschreibung gerade darin, den *Weg* zu beschreiben, *wie* ein bestimmtes Ziel erreicht wird, und welche Schritte in welcher Reihenfolge dazu ausgeführt werden müssen.

Dennoch gibt es in BPMN ein Element, das dem Aufstellen eines Zieles nicht unähnlich ist: Die Aktivitäten in einem *Ad-hoc Subprozess* sind nicht in einer vorgegebenen Reihenfolge durch Sequence Flows verknüpft, sondern werden so lange in beliebiger Reihenfolge ausgeführt, bis die *Completion Condition* des Subprozesses erfüllt ist. Darüber, wie dies erreicht wird, schweigt die Spezifikation, und Literatur zu diesem Thema ist

selten. In einer Arbeit wurde Unterstützung für Ad-Hoc Subprozesse für den *Oryx* Editor implementiert [85], doch in der Praxis dürfte der Ad-hoc Subprozess hauptsächlich in manuellen Prozessen Anwendung finden. Es ist aber auch denkbar, ihn für die Modellierung deliberativer Agenten einzusetzen.

Während der Großteil der Prozesse in BPMN vorgegeben und in genau dieser Weise auszuführen sind, ist es somit möglich, temporäre Ziele aufzustellen und zu verfolgen, bis sie erfüllt sind. Für die Erfüllung dieser Ziele stehen dabei nur die Aktivitäten zur Verfügung, die innerhalb des Ad-hoc Subprozesses definiert sind. Dies stellt zwar eine Einschränkung dar, macht die verfügbaren Pläne aber zugleich explizit und schränkt den Suchraum auf die relevanten Pläne ein.

7.1.2. Von Start-Ereignissen zu Reaktionsregeln

Bisher wurden durch die Abbildung lediglich Aktionen und Pläne des Agenten definiert, jedoch nicht, wann der Agent diese ausführen soll. Natürlich kann jeder der Pläne explizit aufgerufen werden, doch geben die Start-Ereignisse der verschiedenen Pools bereits vor, in welchen Situationen die verschiedenen Prozesse angestoßen werden sollen. Um dieses Verhalten in den Agenten umzusetzen muss für jedes dieser Start-Ereignisse eine Reaktionsregel erstellt werden, die den Plan beim Eintreten der jeweiligen Situation mit passenden Parametern aufruft.

Der *Effekt* der Regeln ist hierbei immer gleich: Der Plan, dessen Anfang das entsprechende Start-Ereignis markiert, wird aufgerufen, bzw. es wird eine entsprechende Intention erstellt. Die *Bedingung* der Regel, ebenso wie die Parameter, mit denen der Plan aufgerufen wird, sind abhängig vom Typ des jeweiligen Start-Events.

- Bei einem *Regel*-Start-Event kann die Bedingung, unter welcher der Plan ausgeführt werden soll, direkt angegeben werden. Prozess-Variablen, die in dieser Bedingung Verwendung finden, werden zu Parametern des Plans.

- Bei einem *Message*-Start-Event ist die Bedingung, dass der Agent auf dem mit dem Ereignis assoziierten Nachrichtenkanal eine Nachricht passenden Typs empfängt. Diese Nachricht wird sodann als Parameter für den Plan verwendet. Wird das Message-Event hingegen mit einem Dienstaufruf assoziiert, wird *keine* Reaktionsregel erstellt, da diese explizit aufgerufen werden.

- Bei einem *Timer*-Start-Event ergibt sich die Bedingung aus dem angegebenen Zeitpunkt bzw. dem Intervall. Hierfür benötigt der Agent ein Konzept von Zeit,

welches auch für die Auswertung der Regeln zur Verfügung stehen muss, sowie Wissen darüber, ob und wann ein Plan zuletzt ausgeführt wurde. Der entsprechende Plan wird ohne Parameter aufgerufen.

- Bei einem *None*-Start-Event, d. h. wenn kein spezieller Event-Typ angegeben ist, wird der entsprechende Plan einmalig beim Start des Agenten ausgeführt.

Andere Arten von Start-Ereignissen werden nicht betrachtet, bzw. für solche werden keine Reaktionsregeln erstellt.

7.1.3. Von Participants und Use Cases zum Agentenmodell

Das Ergebnis der vorherigen Schritte ist eine Menge von Plänen, oder Aktionen, die den einzelnen Prozessen entsprechen, sowie Regeln zum Ausführen der Pläne entsprechend der jeweiligen Start-Ereignisse. In diesem Schritt werden diese Verhaltensweisen nun verschiedenen *Rollen* zugeordnet, aus denen sich schließlich das Multiagenten-System zusammensetzen wird. Zu diesem Zweck wird das *Business Process System* betrachtet, in dem die einzelnen *Business Process Diagrams* zu einem Gesamtsystem verbunden werden. Wie im vorigen Kapitel erläutert kann dieses Gesamtsystem beispielsweise als BPMN-Conversation-Diagram oder als UML-Usecase-Diagramm dargestellt werden.

Jeder der Participants des Systems (oder ein Akteur im Usecase-Diagramm) wird hierbei auf eine Agenten-Rolle abgebildet, welche die Pläne anbietet und über die jeweiligen Reaktionsregeln verfügt, die den Prozessen entsprechen, in die der Participant in den verschiedenen Business Process Diagrams involviert ist.

Konkrete Agenten werden in dieser Abbildung nicht erzeugt, da die Prozessdiagramme keine Informationen darüber liefern, in welcher Anzahl die einzelnen Agenten auftreten und wie diese im Multiagenten-System verteilt sind. Die Zuweisung von Rollen zu konkreten Agenten bleibt somit dem Entwickler überlassen.

7.2. Diskussion und mögliche Erweiterungen

Die im vorigen Abschnitt beschriebene Abbildung deckt die wichtigsten Eigenschaften von Prozessen und Agenten ab und kann auf verschiedene Arten umgesetzt und für die prozessorientierte Modellierung von Multiagenten-Systemen eingesetzt werden – siehe hierzu Kapitel 8 für eine prototypische Umsetzung.

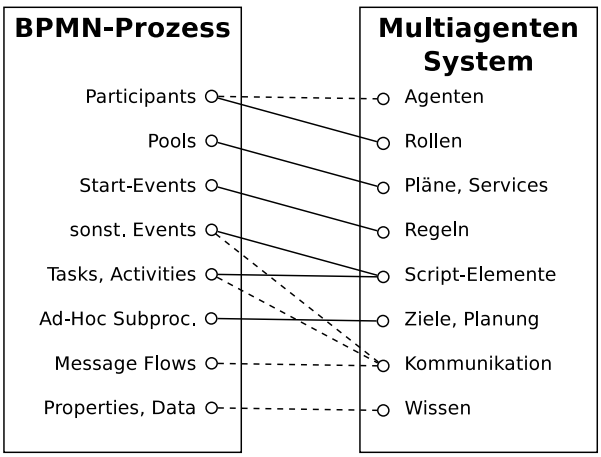


Abbildung 7.2.: Abdeckung der Konzepte seitens BPMN und Agenten.

In diesem Abschnitt sollen nun verschiedene Aspekte und zugrunde liegende Designentscheidungen näher erläutert werden: Weswegen die Abbildung in dieser Art konzipiert wurde, was für Elemente bei der Abbildung nicht berücksichtigt werden, und was für Erweiterungen für die Zukunft noch vorstellbar wären.

Ein grober Überblick über die aktuell abgedeckten Konzepte von Prozessen und Agenten und deren Beziehung zueinander ist in Abbildung 7.2 dargestellt.

7.2.1. Rollen und Agenten

Auf oberster Ebene werden Participants auf Rollen, und die diesen Participants zugeordneten Pools auf Verhaltensweisen, oder Pläne, dieser Rollen abzubilden. Im Folgenden sollen einige Überlegungen und Designentscheidungen, die in diesem Rahmen getroffen wurden, erläutert werden.

Einteilung in verschiedene Prozessdiagramme

Während jeder Pool auf einen Plan abgebildet wird, haben die Prozessdiagramme als solche keine Entsprechung im Agenten-System und werden in der Abbildung nicht betrachtet. Prinzipiell wäre es möglich, auf diese Ebene zu verzichten und sämtliche Pools in einem einzigen Prozessdiagramm darzustellen. Der Übersichtlichkeit halber bietet es sich jedoch an, unterschiedliche Prozessdiagramme für die verschiedene Szenarien, oder Use Cases zu erstellen und in diesen die jeweils zu diesem Szenario gehörigen Pools zusammenzufassen – in der Regel all jene Pools, die untereinander interagieren.

Seit Version 2 der BPMN gibt es zu diesem Zweck die *Conversation Diagrams*, die einen Überblick der verschiedenen Participants und deren Interaktionen darstellen [118]. Jede dieser Interaktionen entspricht hierbei einem *Collaboration Diagram*, welches dem „klassischen“ BPMN Diagramm aus Version 1 des Standards entspricht.

Rollen, Agenten und Multiagenten-System

Indem Participants auf Rollen und Pools auf deren Verhaltensweisen abgebildet werden, bleiben keine geeigneten Elemente, auf denen basierend konkrete Agenten und Multiagenten-Systeme abgeleitet werden könnten. Dies hat jedoch den Vorteil, dass eine Rolle von mehreren Agenten implementiert, oder dass sich die Agenten auf verschiedene Plattformen verteilen können. Entspräche etwa jeder Pool dem Verhalten eines Agenten, wäre es deutlich schwerer, Verhaltensweisen für mehrere Agenten zu modellieren.

Stattdessen müssen die Rollen in einem nachgelagerten Schritt zu Agenten und diese zu einem Multiagenten-System aggregiert werden (siehe Abschnitt 8.1). Eine mögliche Alternative hierzu bestünde darin, die Aufteilung auf Agenten und Plattformen über spezielle Artefakte direkt in die Prozessdiagramme aufzunehmen. So könnten etwa die Participants im BPS zu verschiedenen Agenten-Artefakten, und diese wiederum zu Plattform-Artefakten verbunden werden.

Betrachtung von Lanes als Rollen

Lanes ermöglichen es, genaue Zuständigkeiten für die verschiedenen Aktivitäten eines Pools zu definieren, etwa ein Unternehmen (den Pool) in Abteilungen (Lanes) zu unterteilen. Zugleich ist die Bedeutung von Lanes in der BPMN allerdings sehr vage bzw. gänzlich dem Modellierer überlassen [118, Abschnitt 10.8].

Im Rahmen einer Abbildung auf Multiagenten-Systeme könnten Pools auf Agenten und Lanes innerhalb der Pools auf Rollen des Agenten abgebildet werden; oder auch Lanes auf individuelle Agenten, und Pools auf Organisationen kooperierender Agenten. Auf den ersten Blick mag diese Abbildung natürlicher erscheinen: Jede Aktivität könnte einem Plan der jeweiligen Rolle entsprechen, Properties auf Ebene des Pools dem Faktenwissen des Agenten (oder einem gemeinsam genutzten Blackboard der Organisation), und der Workflow innerhalb der Pools würde darauf abgebildet werden, wie die verschiedenen Pläne hintereinander aufgerufen werden und sich bedingen.

In der Umsetzung stellt dies jedoch einige Schwierigkeiten dar. So könnte etwa jeder Plan (jede Aktivität) am Ende der Ausführung den Plan für die jeweils nächste Aktivität

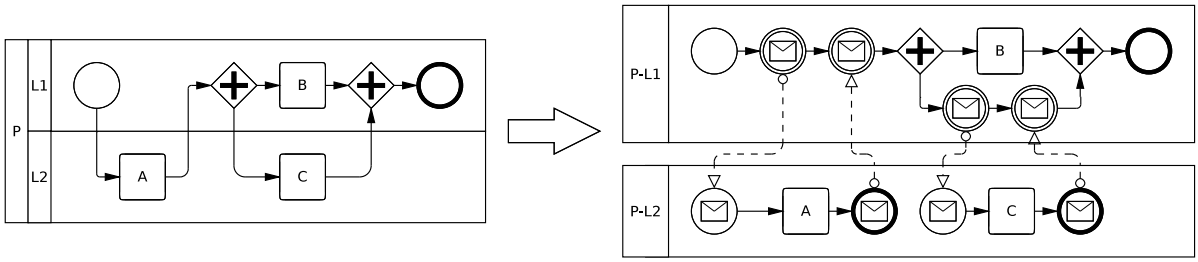


Abbildung 7.3.: Möglicher Vorverarbeitungsschritt zum Aufspalten von Lanes zu kommunizierenden Pools (nicht umgesetzt).

aufrufen. Alternativ könnten die verschiedenen Lanes auf ein eigenständiges Interaktionsprotokoll innerhalb der Pools abgebildet werden. Letzteres könnte auch als ein Vorverarbeitungsschritt vor der eigentlichen Abbildung ausgeführt werden, um Pools mit mehreren Lanes in mehrere kommunizierende Pools aufzuspalten (siehe Abbildung 7.3), sodass die in dieser Arbeit beschriebene Abbildung auch auf diesen Fall anwendbar wäre. Dies stellt jedoch zusätzliche Anforderungen an die Struktur des Prozesses, etwa dass zusammengehörige Gateways (Verzweigung und Zusammenführung) in derselben Lane liegen, damit die Synchronisierung der Workflows zwischen den Lanes gelingt. Zudem ergibt sich abermals das Problem, dass auf diese Weise die in einer Lane beschriebene Rolle nicht in einem anderem/einem weiteren Agenten als dem, der dem umgebenden Pool entspricht, genutzt werden kann, was die Wiederverwendbarkeit stark einschränken würde.

Lanes werden daher nicht betrachtet. Stattdessen wird die Ausdruckskraft der Prozessdiagramme vollständig für die Beschreibung der Pläne genutzt, und die Zusammenhänge zwischen den Plänen nur indirekt, über die Start-Ereignisse und die entsprechenden Reaktionsregeln festgelegt.

7.2.2. Pläne und Regeln

Die einzelnen Pläne, oder Verhaltensweisen, setzen sich aus den jeweiligen Aktivitäten, Tasks, Subprozessen und Events zusammen, und in welcher Reihenfolge diese im Workflow des Pools verbunden sind.

Start-Ereignisse und Reaktionsregeln

Eine Sonderstellung nehmen hierbei die Start-Events ein: Diese sind zwar Teil des Workflows, zugleich aber dafür verantwortlich, den Prozess auszulösen. Sie können daher

keinen Teil der Agenten-Pläne darstellen, sondern müssen diesen vorgelagert sein: Sie werden auf entsprechende Reaktionsregeln abgebildet.

Dennoch bleiben die Start-Events Teil des Workflows, und können Einfluss auf den Ablauf haben. Daher wird das auslösende Ereignis, etwa eine eintreffende Nachricht, als Parameter an den Plan übergeben, sodass über die Zuweisungen des Start-Ereignisses Informationen daraus ausgelesen und verarbeitet werden können.

Abhängig von der Implementierung können die Reaktionsregeln auf unterschiedliche Art und Weise umgesetzt werden, etwa als tatsächliche Regeln einer integrierten Regel-Engine, oder in Form verschiedener Observer (siehe Abschnitt 8.3).

Prozessstruktur

Ein grundsätzliches Problem bei der Abbildung von Prozessdiagrammen in Form gerichteter Graphen auf ausführbare Programme ist die Abbildung des Kontrollflusses. Diese Problematik wurde in Abschnitt 3.3 bereits erläutert und verschiedene Methoden zur Strukturierung von Prozessen wurden vorgestellt. In Abhängigkeit der jeweiligen Zielsprache bieten sich hier verschiedene mehr oder weniger direkte Verfahren an:

- Der Prozess kann direkt interpretiert werden, wobei der aktuelle Schritt im Prozess im internen Zustand des Interpreters gespeichert und die jeweils möglichen Folgeaktivitäten aus dem Prozessdiagramm ausgelesen werden. Da der Prozess im Voraus nicht verändert oder erweitert werden kann, muss jeder Aspekt des Systems direkt im Prozessdiagramm beschrieben oder in Komponenten oder Dienste ausgelagert werden, die im Prozess nur noch aufgerufen werden müssen.

- Eine Mischform aus Interpreter und Code-Generierung findet sich bei WADE [27]: Hier wird für jede Aktivität im Prozess eine Methode generiert, während die Nachfolger-Relation des Workflows in einer speziellen Datenstruktur codiert wird. Zur Laufzeit wird die als nächstes auszuführende Methode in dieser Datenstruktur nachgeschlagen. Im generierten Code ist der Prozessablauf nicht mehr direkt ersichtlich.

- In einer Sprache mit `goto`-artigen Befehlen kann der Workflow direkt abgebildet werden. Gegebenenfalls können auch andere Sprachelemente zweckentfremdet werden, so wie etwa Ouyang Workflows auf sich gegenseitig auslösende Event-Handler in BPEL abbildet [120]. Der auf diese Weise generierte Code ist für den Entwickler jedoch nur schwer zu verstehen und zu warten.

- Der Prozess kann über Mustererkennungs-Regeln in eine entsprechende strukturierte Form übertragen werden [103]. Zwar lassen sich nicht alle Prozesse auf diese Weise strukturieren, zugleich werden jedoch auch viele strukturelle Probleme von vornherein ausgeschlossen, etwa Deadlocks. Durch die vertraute Verwendung der syntaktischen Elemente der Zielsprache, wie etwa *if/else*-Blöcke und Schleifen, ist so generierter Quellcode sehr viel einfacher zu lesen und zu warten.

Welches dieser Verfahren Anwendung finden soll hängt zum einen von der Zielsprache, zum anderen vom Anwendungsfall ab, da jede Methode Vor- und Nachteile hat. Zugleich ist dieser Aspekt für die Abbildung als solche weniger relevant: Solange die einzelnen Aktionen am Ende in der korrekten Reihenfolge ausgeführt werden, spielt die Abbildung der Prozessstruktur nur eine untergeordnete Rolle. Dieses Kapitel befasst sich daher hauptsächlich mit der Abbildung der verschiedenen Aktivitäten und mit den agentenspezifischen Besonderheiten. Die tatsächliche Implementierung der Abbildung der Prozessstruktur wird in den Abschnitten 8.2 und 8.3 beschrieben.

Betrachtete Workflow-Elemente

Während BPMN eine große Vielfalt an Aktivitäts- und Ereignis-Typen aufweist, sind in der Praxis – für die Beschreibung von Geschäftsprozessen ebenso wie für ausführbare Systeme – nur wenige davon wirklich von Bedeutung [187]. Diese Arbeit konzentriert sich daher auf diejenigen Elemente, die für die Abbildung auf Agenten als am relevantesten angesehen werden (siehe Tabelle 7.1).

Tabelle 7.1.: Abdeckung von Task-, Event- und Gateways-Typen

Typ	**unterstützt**	**nicht unterstützt**
Task	None, Service, Send, Receive, User, Script, Call	Manual, Business Rule
Subprocess	Embedded Subprocess, Ad-Hoc Subprocess	Event Subprocess, Transaction
Event	None, Message, Timer, Rule, Link, Error, Terminate, Boundary-Events	Signal, Multiple, Escalate, Cancel, Compensate
Gateway	Exclusive-OR, Inclusive-OR, Parallel, Event-based	Complex

Hierzu zählen insbesondere die *Message-Events* und *Send-* und *Receive-Tasks* zum Senden und Empfangen von Nachrichten und der *Service-Task* zum Aufrufen anderer Pläne und Dienste, sowie *Timer-* und *Rule-Events* für reaktives Verhalten. Mit diesen

Elementen lassen sich die meisten Abläufe und Interaktionen gut modellieren. Verschiedene andere Verhaltensweisen können zudem über den *Script-Task* realisiert werden. Andere Typen, wie etwa der *Manual Task* oder das *Escalate Event*, sind für die hier beschriebene Abbildung weniger interessant und werden daher nicht betrachtet.

7.2.3. Prozessdaten und Datenfluss

Die Prozessdaten bzw. der Fluss dieser Daten durch den Prozess und die darin verwendeten Aktivitäten und Dienste kann in BPMN auf zweierlei Arten abgebildet werden: Zum einen über Properties und Assignments, und zum anderen über *Data Objects* und Assoziationen derselben mit den entsprechenden Aktivitäten. Während erstere im Diagramm selbst nicht sichtbar sind, erlauben letztere es, den Datenfluss anschaulich darzustellen. Zugleich kann ein Prozessdiagramm, werden alle Daten und Datenflüsse über Data Objects und Associations dargestellt, sehr schnell überladen werden, was der Lesbarkeit wiederum abträglich ist.

Während Data Objects in Geschäftsprozessen für die Dokumentation realer Unternehmensabläufe gut geeignet sind, stellen sie für die Beschreibung ausführbarer Service-Orchestrierungen mitsamt derer jeweiliger Ein- und Ausgabeparameter keine sinnvolle Option dar. Allerdings können Data Objects für die Modellierung *spezieller* Datenflüsse verwendet werden, insbesondere für die Persistenz in Datenbanken oder für das Lesen und Schreiben externer Dokumente und Ressourcen. Im Kontext von Agenten könnten sie etwa für die Zugriff auf das prozessübergreifende Weltwissen der Agenten eingesetzt werden. In der in dieser Arbeit beschriebenen Abbildung wird jedoch nur die Modellierung von Prozessdaten über Properties und Assignments betrachtet.

7.2.4. Agentenspezifische Besonderheiten

Während sich die meisten Aspekte von Multiagenten-Systemen gut mit der BPMN modellieren lassen, gibt es auch Aspekte, für die es unter den üblichen Elementen keine geeignete Entsprechung gibt, oder für die die Notation geringfügig erweitert werden musste. Hierzu zählt zum einen die Darstellung komplexer Interaktionsprotokolle, und zum anderen Aspekte wie das Aufstellen von Zielen und die Planung zur Laufzeit.

Komplexe Interaktionsprotokolle

Aufgrund ihrer Nähe zu BPEL sind *Messages* in BPMN normalerweise Webservice-Aufrufe oder deren Rückgabewerte. Um auch komplexere Interaktionen zu ermöglichen,

insbesondere Nachrichten, die nicht Teil eines Dienstes sind, sowie Nachrichten an Agentengruppen, wurde im vorigen Kapitel komplementär zum *Service* die *Nachricht* als mögliche Implementierung für BPMN-Messages eingeführt (siehe Abschnitt 6.2).

Neben dem eigentlichen Nachrichteninhalt verfügen diese Nachrichten über zwei weitere Attribute: den Absender und den Empfänger der Nachricht. Werden diese zur Laufzeit entsprechend gesetzt, ermöglichen sie etwa, dass der Absender einer Nachricht in einer Prozess-Property gespeichert und als Empfänger einer späteren Nachricht eingesetzt wird. Wird statt einer Agenten-ID der Name einer Nachrichtengruppe angegeben sind somit auch Multicast-Nachrichten möglich.

Hiermit ist es möglich, komplexe Interaktionsprotokolle zu modellieren, beispielsweise für Auktionen oder Varianten von *Contract Net* [152]. Es gibt hierbei keine expliziten unterschiedlichen Nachrichtentypen für verschiedene Sprechakte, wie *Inform* oder *Request*. Der Entwickler kann dennoch Repräsentationen solcher Sprechakte verschicken und im Prozess entsprechend darauf reagieren.

Abhängig davon, um was für eine *Message* es sich handelt – Serviceaufruf oder Nachricht – werden die jeweiligen Elemente auf unterschiedliche Agenten-Konzepte abgebildet. Möchte man restriktiver sein, kann man Services nur in *Service-Tasks*, und Nachrichten nur in *Send-* und *Receive-Tasks* zulassen.

Ziele, Planung und Semantik

Wie im vorigen Abschnitt beschrieben, können für die Modellierung von Zielen Ad-hoc-Subprozesse verwendet werden. Deren *Ad-hoc Completion Condition* entspricht hierbei dem Ziel, und die im Subprozess enthaltenen Aktivitäten werden so lange in nicht festgelegter Reihenfolge ausgeführt, bis die Bedingung, bzw. das Ziel, erreicht ist. Die Zielbedingung kann hierbei Variablen (Properties) im Scope des Subprozesses verwenden.

Das heißt jedoch, dass das Ziel nicht auf allen Plänen des Agenten operiert (die aus den verschiedenen Pools abgeleitet werden), sondern auf den Aktivitäten innerhalb des Subprozesses. Diese müssen daher ebenfalls semantisch beschrieben und im Agentensystem als eigenständige Pläne repräsentiert werden.[2]

Im einfachsten Fall ist jede dieser Aktivitäten ein *Service*-Task ohne Zuweisungen, sodass die Aktivität genau dem über diesen Service aufgerufene Plan entspricht. Sollen auch andere Arten von Tasks oder Tasks mit Zuweisungen unterstützt werden, müssen für diese entsprechende Pseudo-Pläne erzeugt werden, mit denen der eigentliche Planer

[2]Es wäre auch möglich, in einem leeren Ad-hoc Subprozess *ohne* eingebettete Aktivitäten implizit *alle* Pläne zuzulassen; dies würde jedoch nicht der Spezifikation entsprechen.

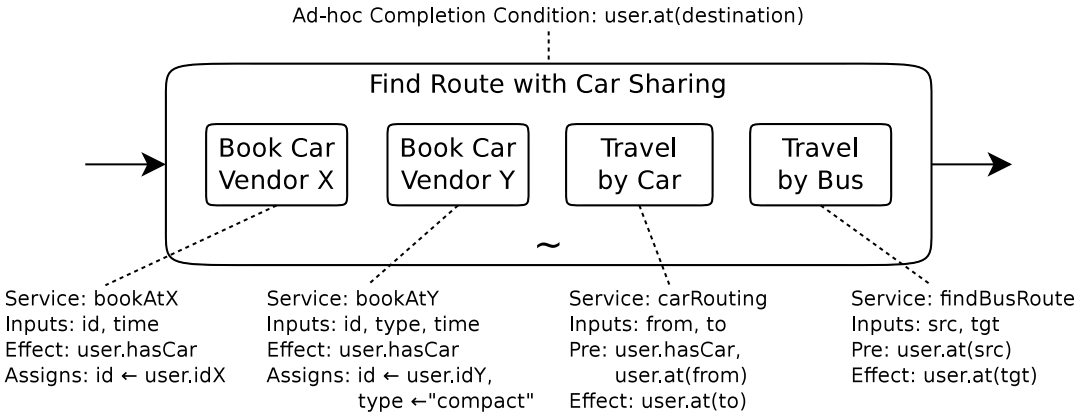


Abbildung 7.4.: Ein Beispiel für die Verwendung des Ad-hoc-Subprozesses. Für die Bearbeitung des Ziels stehen vier Service-Tasks zur Verfügung.

arbeiten kann. Dies kann im Falle von Service-Tasks etwa dazu genutzt werden, dass einige oder alle der Service-Parameter durch entsprechende Zuweisungen vorgegeben werden können, womit der Suchraum für die Planung beträchtlich eingeschränkt werden kann. Die nicht auf diese Weise belegten Service-Parameter müssen weiterhin von einer Planungskomponente so mit Werten belegt werden, dass die Zielbedingung erfüllt wird. Sowohl für die Zielbedingung als auch für die Parameter-Belegung stehen hierbei die Properties im Scope des Subprozesses zur Verfügung.

Abbildung 7.4 zeigt ein einfaches Beispiel für die Verwendung der Ad-Hoc-Subprozesse um Aspekte wie Planung abzubilden. Die *Ad-Hoc Completion Condition* sieht hierbei vor, dass der Nutzer an seinen Zielort gelangt. Hierfür kann er bei verschiedenen Anbietern einen Mietwagen buchen und dann mit diesem fahren, oder den Bus nehmen. Jeder der Tasks entspricht dabei einem (semantisch beschriebenen) Service bzw. einem Plan, wobei teilweise einzelne Input-Parameter vorgegeben (*grounded*) sind, etwa mit der entsprechenden Kundennummer des Nutzers oder die Wahl des Fahrzeugs.

Dieses Konzept ließe sich auch erweitern: Indem der Ad-hoc Subprozess in einer Schleife ausgeführt wird, wird etwa aus dem *achieve*- ein *maintain goal*. Des Weiteren ließen sich an den Subprozess angehängte Event-Handler dazu nutzen, die Verfolgung des Zieles vorzeitig abzubrechen. Diese Arbeit beschränkt sich in dieser Hinsicht jedoch auf den Basisfall ohne Event-Handler. Der eigentliche Planer und die darin verwendeten Algorithmen werden in dieser Arbeit nicht betrachtet und werden als gegeben, bzw. als im jeweiligen Agentenframework vorhanden angenommen.

7.2.5. Offene Punkte und mögliche Erweiterungen

Vorausgesetzt, dass das Prozessdiagramm hinreichend detailliert ist, lässt sich mit dieser Abbildung ein ausführbares Agentensystem erstellen, dessen Verhalten dem modellierten BPMN-Prozess entspricht. Dennoch gibt es auf beiden Seiten – BPMN-Prozess und Agentensystem – Aspekte, die von dieser Abbildung nicht abgedeckt werden.

Verschiedene Agenten-Frameworks legen den Schwerpunkt auf unterschiedliche Aspekte. Diese Arbeit konzentriert sich auf Aspekte, die denen der Prozessdiagramme ähnlich sind, da diese sich natürlicherweise am besten über Prozessmodelle beschreiben lassen. Andere Aspekte, wie etwa *norms and obligations*, werden an dieser Stelle nicht betrachtet. Ein weiterer Gesichtspunkt, der nicht berücksichtigt wird, ist die physische Verteilung der Agenten auf konkrete Computer, ein technische Aspekt, der in den Prozessmodellen nicht beschrieben wird. Solche Aspekte könnten in den Prozessen durch die Einführung spezieller *Artefacts* beschrieben werden, es erscheint jedoch praktikabler, diese aus den Prozessmodellen heraus zu halten und in einem nachgelagerten Schritt einzustellen.

Seitens der BPMN gibt es, wie in den vorigen Abschnitten erläutert, einige Elemente, die von der Abbildung nicht unterstützt werden, wie etwa Lanes, oder auch einzelne Task- und Event-Typen. Während auf die Verwendung von Lanes bewusst verzichtet wird (diese zugleich aber mit dem vormals beschriebenen Vorverarbeitungsschritt auch abgebildet werden könnten), bieten die Tasks und Events Potential für zukünftige Erweiterungen etwa für transaktionales Verhalten. Weiterhin würde auch die Verwendung von Data Objects für den Zugriff auf das planübergreifende Weltwissen des Agenten eine sinnvolle Erweiterung der Modellierungsmöglichkeiten darstellen.

7.3. Formale Beschreibung der Abbildungsregeln

In diesem Abschnitt soll die Abbildung von Prozessmodellen auf Agenten und Multiagenten-Systeme unter Verwendung der im vorigen Kapitel definierten Agenten- und Prozess-Metamodelle formal beschrieben werden [92]. Hierbei werden zwei Teile betrachtet:

- Die Agenten-Architektur, d. h. die Rollen, deren Pläne und Regeln, und wie diese in einem Multiagenten-System zusammengefasst werden.
- Das Agenten-Verhaltens, d. h. die Abläufe innerhalb der Pläne der Agenten.

Im Folgenden sollen diese zwei Aspekte der Abbildung von Prozessen auf Agenten kurz beschrieben werden. Zunächst wird die Abbildung der Pools auf Pläne, Rollen und Reaktionsregeln beschrieben, und danach die der eigentlichen Workflows, welche die Grundlage des Verhaltens der Agenten darstellen.

In den Abbildungsregeln wird die Notation $x \Longrightarrow z$ verwendet um anzuzeigen, dass das Prozess-Element x auf das Agenten-Element z abgebildet wird. Analog hierzu bedeutet $(x,\ y) \Longrightarrow z$, dass der Teil des Prozessgraphs zwischen x und y auf z abgebildet wird. Hierbei gilt: Wenn $x \Longrightarrow z$, dann auch $(x,\ x) \Longrightarrow z$. Dieselbe Notation wird verwendet, um innerhalb einer Regel anzuzeigen, wie bestimmte Elemente durch andere, frühere Regelanwendungen abgebildet wurden.

7.3.1. Abbildung der Agenten-Architektur

Das Business Process System als Ganzes wird auf ein Multiagenten-System abgebildet. Wie in den vorigen Abschnitten geschildert, werden hierbei keine konkreten Agenten, sondern nur Rollen abgeleitet – eine für jeden Participant.

BPS $\Longrightarrow$ MAS: Sei $bps = (id,\ BPD,\ Pt)$ ein Business Process System. Dann gilt:

$$\begin{aligned} bps \implies\ & mas = (id,\ Agents,\ Roles), \text{ wobei} \\ & Agents = \varnothing \\ & Roles = \{role \mid \exists p \in Pt : p \Longrightarrow role\} \end{aligned}$$

Die Rollen sind hierbei zunächst allein durch ihre Pläne und Regeln bestimmt. In der initialen Konfiguration gibt es keine Ziele, diese können aber zur Laufzeit aufgestellt werden. Es wird für jeden Pool des Participants ein Plan, und für jedes Start-Event innerhalb der Pools eine Reaktionsregel erstellt.

Participant $\Longrightarrow$ Rolle: Sei $bps = (id_1,\ BPD,\ Pt)$ ein Business Process System und $pt \in Pt \subset ID$ ein Participant. Dann gilt:

$$\begin{aligned} pt \implies\ & role = (pt,\ P,\ R,\ G), \text{ wobei} \\ & P = \{plan \mid \exists p = (id_2,\ wf,\ pt) : p \Longrightarrow plan\} \\ & R = \{rule \mid \exists p = (id_2,\ (O,\ SF,\ Prop),\ pt), \exists e_s \in O : e_s \Longrightarrow rule\} \\ & G = \varnothing \end{aligned}$$

Jeder Pool wird auf einen Plan abgebildet. Die Inputs und Outputs des Plans, ebenso wie die Precondition und der Effekt, sind zunächst unbestimmt. Hat der Pool mehrere Start-Events, so können ggf. mehrere Pläne abgeleitet werden. Der Inhalt des Plans ergibt sich aus der Abbildung der Strukturen und Workflow-Elemente vom jeweiligen Start- bis zum End-Event.

Pool $\Longrightarrow$ Plan: Sei $bpd = (id_1,\ Pls,\ MF,\ Art)$ ein BPD, $pool = (id_2,\ wf,\ pt)$ ein Pool, sodass $pool \in Pls$, und $wf = (O,\ SF,\ Prop)$ der Workflow dieses Pools, wobei $e_s,\ e_e \in O$ Start- bzw. End-Events sind, und $\nexists w \in O$, sodass $(w,\ e_s,\ c) \in SF$ oder $(e_e,\ w,\ c) \in SF$. Sei Z ein Skript-Element, sodass $(e_s,\ e_e) \Longrightarrow Z$. Dann gilt:

$$\begin{aligned} pool \implies\ & plan = (id_1 id_2,\ In,\ Out,\ pre,\ \mathit{eff},\ Z), \text{ wobei} \\ & In = Out = \varnothing \\ & pre = \mathit{eff} = \varepsilon \end{aligned}$$

Die Start-Events der Pools werden auf Reaktionsregeln abgebildet, welche die entsprechenden Pläne bei Eintreffen der jeweiligen Ereignisse auslösen. Die Regeln selbst werden an dieser Stelle nicht weiter detailliert, da dies von der verwendeten Regel-Notation abhinge. Kommen in diesem Regel-Ausdruck Variablen vor, die in Assignments des Start-Events verwendet werden, so werden diese zu den Input- Parametern des Plans hinzugefügt und über die *map* entsprechend belegt.

Start Event $\Longrightarrow$ Regel: Sei $pool = (id_1,\ wf,\ pt)$ ein Pool, $wf = (O,\ SF,\ Prop)$ der Workflow dieses Pools, und $e_s = (id_2,\ type,\ Ass,\ Attr)$ ein Start-Event, sodass $e_s \in O$, $\nexists w, c :\ (e_s,\ w,\ c) \in SF$ und $type \neq$ SERVICE. Dann gilt:

$$\begin{aligned} e_s \implies\ & rule = (cond,\ plan,\ map), \text{ wobei} \\ & cond = [\text{ Regel-Ausdruck, abhängig von } type\ ] \\ & plan = p \text{ , sodass } pool \Longrightarrow p \\ & map = \{(x,\ x) \mid assign(x,\ y) \in Ass\} \\ & In_{plan} \leftarrow In_{plan} \circ \{y \mid (x,\ y) \in map\} \end{aligned}$$

Eine Ausnahme stellen Service Start-Events dar (d. h. Message Start-Events, bei denen die Nachricht ein Serviceaufruf ist). Für diese wird keine Regel erzeugt, da ein solcher

Plan explizit aufgerufen werden soll.[3] Stattdessen werden die semantischen Attribute aus der im Prozess hinterlegten Service-Beschreibung auf den Plan übertragen.

Start Event $\Longrightarrow$ Plan: Sei $pool = (id_1,\ wf,\ pt)$ ein Pool, $wf = (O,\ SF,\ Prop)$ der Workflow dieses Pools, und $e_s = (id_2,\ type,\ Ass_s,\ Attr_s)$ ein Start-Event, sodass $e_s \in O$, $\nexists w, c:\ (e_s,\ w,\ c) \in SF$, $type =$ SERVICE und ('impl', s) $\in Attr_s$ mit $s = (id_s,\ prov,\ In_s,\ Out_s,\ pre,\ eff)$. Sei (optional) $e_e = (id_3,\ type,\ Ass_e,\ Attr_e)$ ein End-Event, sodass $e_e \in O$, $\nexists w, c:\ (w,\ e_e,\ c) \in SF$ und ('impl', s) $\in Attr_e$. Dann gilt:

$$
\begin{aligned}
e_s \quad \Longrightarrow \quad & plan = p \text{ , sodass } pool \Longrightarrow p \\
& In_{plan} \leftarrow In_{plan} \circ In_s \\
& Out_{plan} \leftarrow Out_{plan} \circ Out_s \\
& Pre_{plan} \leftarrow Pre_{plan} \vee Pre_s \\
& Eff_{plan} \leftarrow Eff_{plan} \vee Eff_s
\end{aligned}
$$

7.3.2. Abbildung der Prozesse und Verhaltensweisen

Im Folgenden wird beschrieben, wie die einzelnen Elemente der Prozess-Diagramme, die im vorigen Abschnitt auf Agenten-Pläne abgebildet wurden, auf die Skript-Elemente dieser Pläne übertragen werden. Hierbei können noch einmal zwei Kategorien von Abbildungsregeln unterschieden werden: solche, welche die Struktur des Prozesses betreffen, d. h. wie die verschiedenen Tasks und Events nacheinander ausgeführt werden, und solche, welche die einzelnen Tasks und Events selbst betreffen.

Prozessstruktur

Zunächst werden die verschiedenen Prozessstrukturen betrachtet, und wie diese in korrespondierende Skript-Elemente des Agenten-Plans überführt werden können. Dieser Vorgang, einen (Prozess-)Graphen in ein entsprechendes strukturiertes Programm zu überführen, ist sehr komplex – siehe hierzu Abschnitt 3.3. Die an dieser Stelle aufgeführten Regeln decken nur die wichtigsten Aspekte dieser Transformation ab. Diese und ergänzend hierzu weitere Regeln wurden in der Implementierung in Form eines Graphtransformationssystems umgesetzt (siehe 8.2.3).

[3] Im Falle, dass das Agentenframework über kein Service-Konzept verfügt, muss eine Reaktionsregel analog zu denen für Message Start-Events erstellt werden, die den Plan bei Eintreffen eines entsprechenden Dienstaufrufs in Form einer Nachricht ausführt.

Die Abbildung ist hierbei ein iterativer Prozess der von innen nach außen durchgeführt wird: Zunächst werden die atomaren Elemente, die Tasks und Events abgebildet, deren Abbildungsregeln weiter unten beschrieben werden. Danach werden diese sukzessive zu Sequenzen, Bedingungen und Schleifen zusammengefügt [91]. Die Regeln beziehen sich jeweils auf die Elemente eines Pools $p = (id,\ wf,\ pt)$ oder eines Subprocesses $sp = (id,\ wf,\ EH,\ succ_{EH},\ Attr)$ mit $wf = (O,\ SF,\ Prop)$.

Abbildung 7.5 zeigt eine Übersicht über die Bedingungen der verschiedenen in diesem Abschnitt beschriebenen Regeln, d. h. über die Strukturen, die mittels dieser Regeln abgebildet werden. Hierbei werden die gleichen Bezeichner wie in den Regeln selbst verwendet. Gruppen von Elementen, die in früheren Regeln auf Skript-Elemente abgebildet wurden, sind grau hinterlegt.

Die einfachste und zugleich wichtigste Struktur ist die Sequenz (Abbildung 7.5 a): Ist eine beliebige Anzahl Flow Objects im Prozessgraphen durch unbedingte Sequence Flows in Reihe und ohne weitere Abzweigungen verbunden, so sind deren Entsprechungen im Agentenskript Teil einer Sequenz.

Kontrollfluss $\Longrightarrow$ Sequence: Seien $x_i,\ y_i \in O\ (i \leq n)$, sodass $\forall i < n:\ (y_i,\ x_{i+1},\ \varepsilon) \in SF$, und $\nexists w \in O \setminus \{x_i,\ y_i \mid i \leq n\}$, sodass $(w,\ x_k,\ c) \in SF$ oder $(y_k,\ w,\ c) \in SF$ $(1 < k < n)$. Seien ferner z_i Skript-Elemente, sodass $(x_i,\ y_i) \Longrightarrow z_i\ (i \leq n)$. Dann gilt:

$$(x_1,\ y_n) \implies seq(z_1,\ \ldots,\ z_n)$$

Die Grenzen von Blöcken sind durch Gateways markiert. Die verschiedenen Gateway-Typen entsprechen hierbei unterschiedlichen Arten von Block-Strukturen.

XOR-Gateways entsprechen einem einfachen *if/else* (Abbildung 7.5 b), d. h. einem *cond*-Element im Agenten-Skript. An dieser Stelle werden nur Blöcke mit genau zwei Verzweigungen betrachtet.[4] Die auszuwertende Bedingung ergibt sich aus der Bedingung eines der Sequence Flows, und der Rumpf aus den Skript-Elementen, die der jeweiligen Verzweigung entsprechen (bspw. einer Sequenz).

Kontrollfluss $\Longrightarrow$ Condition: Seien $g_1,\ g_2,\ x_1,\ y_1,\ x_2,\ y_2 \in O$, wobei $g_1 = (id_1,\ \text{XOR},\ \varnothing)$ und $g_2 = (id_2,\ \text{XOR},\ \varnothing)$ Gateways sind. Seien $(g_1,\ x_1,\ c)$, $(g_1,\ x_2,\ \varepsilon)$, $(y_1,\ g_2,\ \varepsilon)$, $(y_2,\ g_2,\ \varepsilon) \in SF$, und seien z_1 und z_2 Skript-Elemente, sodass $(x_1,\ y_1) \Longrightarrow z_1$ und

[4] Blöcke mit mehr als zwei Verzweigungen können o. B. d. A. zu verschachtelten Blöcken mit jeweils nur zwei Verzweigungen überführt werden.

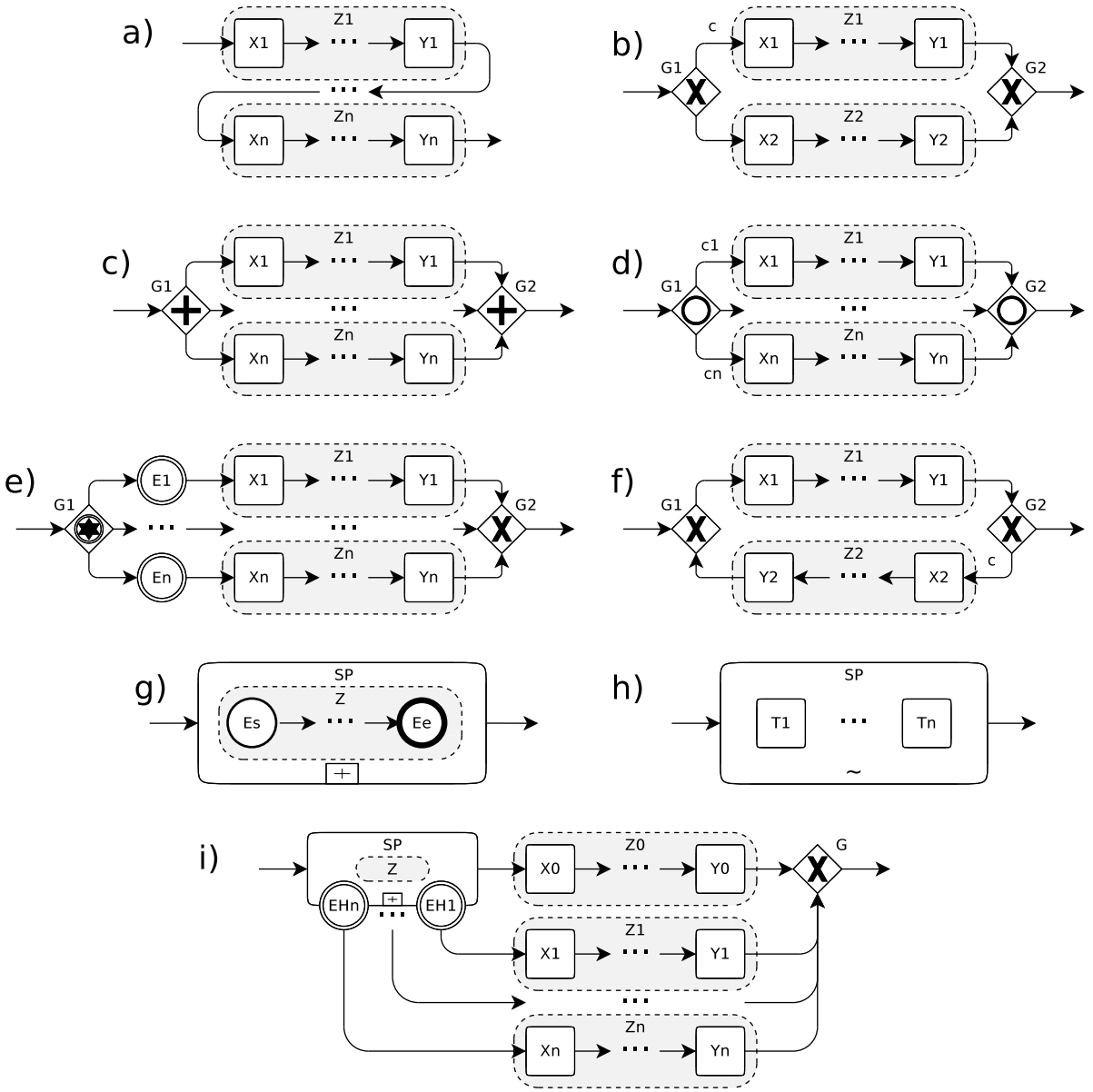


Abbildung 7.5.: Abbildung der Prozessstruktur und des Kontrollflusses: a) Sequence, b) Condition, c) Parallel, d) Parallel-Conditional, e) Event-based Condition, b) While-Loop, g) einfacher Subprocess, h) Ad-hoc Subprocess, i) Subprocess mit Event-Handler. [91]

$(x_2,\ y_2) \Longrightarrow z_2$. Dann gilt:

$$(g_1,\ g_2) \implies cond(c,\ z_1,\ z_2)$$

AND-Gateways werden auf parallele Ausführung abgebildet (Abbildung 7.5 c). In diesem Fall ist die Anzahl der Verzweigungen zwischen den Gateways jedoch nicht beschränkt: Das Ergebnis der Abbildung jedes einzelnen dieser Zweige kann gleichermaßen in das resultierende *par*-Element aufgenommen werden.

Kontrollfluss $\Longrightarrow$ Parallel: Seien $g_1,\ g_2,\ x_i,\ y_i \in O\ (i \leq n)$, wobei $g_1 = (id_1,\ \text{AND},\ \varnothing)$ und $g_2 = (id_2,\ \text{AND},\ \varnothing)$ Gateways sind. Seien $(g_1,\ x_i,\ \varepsilon),\ (y_i,\ g_2,\ \varepsilon) \in SF$, und seien z_i Skript-Elemente, sodass $(x_i,\ y_i) \Longrightarrow z_i\ (i \leq n)$. Dann gilt:

$$(g_1,\ g_2) \implies par(z_1,\ \ldots,\ z_n)$$

Ein OR-Gateway (*inclusive-or*, Abbildung 7.5 d) wird auf eine Kombination aus paralleler und bedingter Ausführung abgebildet: Jeder Zweig wird in ein eigenes *cond*-Element eingepackt, die alle zusammen unabhängig und parallel zueinander überprüft und ausgeführt werden.[5]

Kontrollfluss $\Longrightarrow$ Parallel-Conditional: Seien $g_1,\ g_2,\ x_i,\ y_i \in O\ (i \leq n)$, wobei $g_1 = (id_1,\ \text{OR},\ \varnothing)$ und $g_2 = (id_2,\ \text{OR},\ \varnothing)$ Gateways sind. Seien $(g_1,\ x_i,\ c_i),\ (y_i,\ g_2,\ \varepsilon) \in SF$, und seien z_i Skript-Elemente, sodass $(x_i,\ y_i) \Longrightarrow z_i\ (i \leq n)$. Dann gilt:

$$(g_1,\ g_2) \implies par(cond(c_1,\ z_1,\ nop),\ \ldots,\ cond(c_n,\ z_n,\ nop))$$

Ein EVENT-BASED XOR-Gateway entspricht einem Verhalten, bei dem die Ausführung des Plans unterbrochen wird, bis ein bestimmtes Ereignis eingetroffen ist (Abbildung 7.5 e). Dies kann mit den grundlegenden Agenten-Skript-Elementen umgesetzt werden, indem für jedes Ereignis ein eigener Ausführungsthread abgespalten wird. Ein weiterer Thread wartet eine unbestimmte Zeit, bis er durch einen der Ereignis-Threads unterbrochen wird. Die Abbildung der Ereignisse selbst wird im nächsten Abschnitt beschrieben. Je nachdem, welches Ereignis eingetroffen ist, wird der Plan unterschiedlich fortgesetzt.

Kontrollfluss $\Longrightarrow$ Event-Handler-Threads: Seien $g_1,\ g_2,\ e_1\ ,...,\ e_n,\ x_1,\ ...,\ x_n,\ y_1,\ ...,\ y_n \in O$, wobei $g_1 = (id_{g1},\ \text{EVENT},\ \varnothing)$ und $g_2 = (id_{g2},\ \text{XOR},\ \varnothing)$ Gateways und

[5] Ein OR-Gateway kann auch einen Default-Fall haben; der Einfachheit halber wird dieser Fall an dieser Stelle jedoch nicht betrachtet.

$e_i = (id_{ei},\ type_i,\ attr_i)$ Events sind. Seien $(g_1,\ e_i,\ \varepsilon)$, $(e_i,\ x_i,\ \varepsilon)$, $(y_i,\ g_2,\ \varepsilon) \in SF$, und seien $z_1,\ ...,\ z_n,\ X_1,\ ...,\ X_n$ Skript-Elemente, sodass $(x_i,\ y_i) \Longrightarrow z_i$ und $e_i \Longrightarrow X_i$ $(1 \leq i \leq n)$. Dann gilt:

$$
\begin{aligned}
(g_1,\ g_2) \implies\ & seq(sub,\ ...,\ eh_i,\ ...,\ join(id_{g1}),\ ...,\ stop(\text{`h}_\text{i}\text{'}),\ ...,\ branch_i,\ ...) \\
& sub = fork(id_{g1},\ while(\top,\ nop)) \\
& eh_i = fork(\text{`h}_\text{i}\text{'},\ seq(assign(\text{`t}_\text{i}\text{'}, \bot),\ X_i,\ assign(\text{`t}_\text{i}\text{'}, \top),\ stop(id_{g1}))) \\
& branch_i = cond(\text{`t}_\text{i}\text{'},\ z_i,\ nop)
\end{aligned}
$$

Schließlich können XOR-Gateways auch die Grenzen einer *while*-Schleife markieren (Abbildung 7.5 f). Der (optionale) erste Zweig wird hierbei zweimal ausgeführt: Zum einen bevor die Bedingung zum ersten mal geprüft wird, und zum anderen als Teil der Schleife.

Kontrollfluss $\Longrightarrow$ While: Seien $g_1,\ g_2,\ x_1,\ y_1,\ x_2,\ y_2 \in O$, wobei $g_1 = (id_1,\ \text{XOR},\ \varnothing)$ und $g_2 = (id_2,\ \text{XOR},\ \varnothing)$ Gateways sind. Seien $(g_1,\ x_1,\ \varepsilon)$, $(y_1,\ g_2,\ \varepsilon)$, $(g_2,\ x_2,\ c)$, $(y_2,\ g_1,\ \varepsilon) \in SF$, und seien z_1 und z_2 Skript-Elemente, sodass $(x_1,\ y_1) \Longrightarrow z_1$ und $(x_2,\ y_2) \Longrightarrow z_2$. Dann gilt:

$$(g_1,\ g_2) \implies seq(z_1,\ while(c,\ seq(z_2,\ z_1)))$$

Ein einfacher Subprozess (Abbildung 7.5 g) wird so abgebildet, dass der Workflow im Inneren des Subprozesses direkt an der entsprechenden Stelle in das Agenten-Skript eingefügt wird. Das Verhalten entspricht dem des Prozesses; je nach Implementierung kann der Subprozess auch auf eine eigene Funktion oder eine eingebettete Klasse abgebildet werden (siehe Abschnitt 8.3).

Subprocess $\Longrightarrow$ Eingebettetes Skript: Sei $sp = (id_s,\ swf,\ \varnothing,\ \varnothing,\ \varnothing)$ ein Subprozess ohne Event-Handler, mit $swf = (O,\ SF,\ Prop)$, wobei $e_s,\ e_e \in O$ eindeutige Start- bzw. End-Events sind, und $\nexists w \in O$, sodass $(w,\ e_s,\ c) \in SF$ oder $(e_e,\ w,\ c) \in SF$. Sei Z ein Skript-Element, sodass $(e_s,\ e_e) \Longrightarrow Z$. Dann gilt:

$$sp \implies Z$$

Ein *ad-hoc* Subprozess (Abbildung 7.5 h) wird auf das Aufstellen eines Zieles abgebildet. Hierbei entspricht das Attribut *ad-hoc completion condition* direkt der Zielbe-

dingung. Die Pläne, die für die Erfüllung des Ziels zur Verfügung stehen, werden über *service* Tasks im Innern des Subprozesses bestimmt (die Abbildung der Service-Tasks folgt im nächsten Abschnitt).

Ad-hoc Subp. $\Longrightarrow$ Achieve Goal: Sei $sp = (id_s,\ swf,\ \varnothing,\ \varnothing,\ At)$ ein Subprozess mit ('comp-cond', cc) $\in At$ und Workflow $swf = (\{t_1,\ \ldots,\ t_n\},\ \varnothing,\ \varnothing)$, mit *Service* Tasks $t_i = (id_i,\ \text{SERVICE},\ Ass_t,\ At_t)$, und ('impl', $service_i$) $\in At_t$ (mit $service_i = (n_i,\ pt_i,\ i_i,\ o_i,\ p_i,\ e_i)$), sodass $t_i \Longrightarrow P_i$ $(1 \leq i \leq n)$. Dann gilt:

$$\begin{aligned} sp \ &\Longrightarrow\ achieve(goal) \\ &\qquad\ goal = (cc,\ \{P_1,\ \ldots,\ P_n\}) \end{aligned}$$

Die Abbildung eines Subprozesses mit angefügten Event-Handlern (Abbildung 7.5 i) ist ähnlich der für ein EVENT-BASED XOR-Gateway. Wieder werden die verschiedenen Ereignisse in separaten Threads überprüft, die ggf. den Haupt-Thread unterbrechen und bestimmen, welcher Teil des Prozesses als nächstes ausgeführt wird. Anders als beim Gateway wird im Haupt-Thread nicht nur gewartet, sondern der eigentliche Subprozess abgearbeitet. Ist dieser abgeschlossen, bevor das erste Ereignis eintritt, werden die Event-Threads unterbrochen und der an den Subprozess anschließende Teil des Prozesses ausgeführt.

Subp. mit Event-H. $\Longrightarrow$ Event-Handler-Threads: Seien $sp,\ g,\ x_0,\ ...,\ x_n,\ y_0,\ ...,\ y_n \in O$, wobei $sp = (id_s,\ swf,\ EH,\ succ_{EH},\ \varnothing)$ ein Subprozess und g ein Gateway ist, und seien $eh_1,\ ...,\ eh_n \in EH$ Event-Handler des Subprozesses (Events). Seien $(eh_i,\ x_i) \in succ_{EH}$ $(1 \leq i \leq n)$ und $(sp,\ x_0,\ \varepsilon)$, $(y_i,\ g,\ \varepsilon) \in SF$ $(0 \leq i \leq n)$ Sequence-Flows, und seien $Z,\ z_0,\ ...,\ z_n,\ X_1,\ ...,\ X_n$ Skript-Elemente, sodass $sp \Longrightarrow Z$, $(x_i,\ y_i) \Longrightarrow z_i$ und $eh_i \Longrightarrow X_i$. Dann gilt:

$$\begin{aligned} (sp,\ g) \ &\Longrightarrow\ seq(sub,\ ...,\ eh_i,\ ...,\ join(id_{sp}),\ ...,\ stop(\text{`h}_\text{i}\text{'}),\ ...,\ continue) \\ &\qquad\ sub = fork(id_{sp},\ Z) \\ &\qquad\ eh_i = fork(\text{`h}_\text{i}\text{'},\ seq(assign(\text{`t}_\text{i}\text{'}, \bot),\ X_i,\ assign(\text{`t}_\text{i}\text{'}, \top),\ stop(id_s))) \\ &\qquad\ continue = seq(assign(\text{`n'}, \top),\ ...,\ branch_i,\ ...,\ cond(\text{`n'},\ z_0,\ nop)) \\ &\qquad\ branch_i = cond(\text{`t}_\text{i}\text{'},\ seq(assign(\text{`n'}, \bot),\ z_i),\ nop) \end{aligned}$$

Mit diesen Regeln lassen sich die wichtigsten Strukturen des Prozess-Diagramms auf entsprechende Elemente der Agenten-Skripte abbilden. Dennoch gibt es auch Prozess-

graphen, die sich nicht direkt in eine strukturierte Form überführen lassen (siehe Abschnitt 3.3, insb. [95]). Dies stellt jedoch keine wesentliche Einschränkung der Ausdrucksfähigkeit dar, da ein Großteil solcher Strukturen ohnehin zu Deadlocks und ähnlichem unerwünschten Verhalten führen würden, etwa wenn die Typen von verzweigenden und zusammenführenden Gateways nicht zueinander passen.

Prozesselemente

Nachdem im letzten Abschnitt die verschiedenen Strukturen im BPMN-Prozess auf entsprechende Agenten-Skript-Elemente abgebildet wurden, befasst sich dieser Teil mit den „atomaren“ Elementen der Prozesse, aus denen diese Strukturen zusammengesetzt sind: Den Tasks und Events. Hierbei werden an dieser Stelle weder Subprozessen und Gateways (die zu den Prozessstrukturen zählen) noch Start-Events (die auf die Agenten-Architektur abgebildet werden) betrachtet; diese wurden in den letzten Abschnitten bereits beschrieben.

Tasks und Events können auf verschiedene Skript-Elemente abgebildet werden, je nachdem um was für einen Typ es sich handelt und welche weiteren Attribute gesetzt sind. In jedem Fall können sowohl Tasks als auch Events Zuweisungen enthalten (Assignments), etwa um die Parameter von Services zu setzen und deren Ergebnisse auszuwerten. Die Tasks und Events werden daher nicht auf ein einzelnes Skript-Element, sondern auf eine Sequenz abgebildet, sodass vor und nach der eigentlichen Aktion die entsprechenden Assignments ausgeführt werden können – solche mit *assign time* BEFORE davor, solche mit *assign time* AFTER danach.

Die Assignments, die Teil jedes Flow Objects sein können, werden auf entsprechende *assign*-Elemente abgebildet. Die Assign-Time spielt an dieser Stelle keine Rolle, da diese bereits in der Abbildung der jeweiligen Tasks und Events ausgewertet wurde, um das Assignment entsprechend zu platzieren.

Assignment $\Longrightarrow$ Assign: Sei $ass = (prop,\ expr,\ \text{BEFORE} \mid \text{AFTER})$ ein Assignment als Bestandteil eines Tasks oder Events. Dann gilt:

$$ass \implies assign(prop,\ expr)$$

Der eigentliche Task kann, seinem Task-Typen entsprechend, auf verschiedene Skript-Elemente abgebildet werden, insbesondere auf das Senden und Empfangen von Nachrichten, oder Aufrufe anderer Pläne oder Services.

Task $\Longrightarrow$ verschiedene Skript-Elemente: Sei $task = (id,\ type,\ Ass,\ Attr) \in O$ ein Task mit Typ *type*, Assignments Ass und weiteren Attributen $Attr$. Dann gilt:

$$
\begin{aligned}
task \implies\ & seq(ass^b_1,\ \ldots,\ ass^b_n,\ X,\ ass^a_1,\ \ldots,\ ass^a_n), \text{ wobei} \\
& ass^b_i \in \{assign(prop,\ expr) \mid (prop,\ expr,\ \textsc{before}) \in Ass\} \\
& ass^a_i \in \{assign(prop,\ expr) \mid (prop,\ expr,\ \textsc{after}) \in Ass\} \\
& X = \begin{cases}
send(m) & \text{wenn } type = \textsc{send},\ (\text{'msg'},\ m) \in Attr \\
receive(m) & \text{wenn } type = \textsc{rec},\ (\text{'msg'},\ m) \in Attr \\
invoke(n, i, o) & \text{wenn } type = \textsc{service},\ (\text{'impl'},\ service) \in Attr \\
 & \text{wobei } service = (n,\ pt,\ i,\ o,\ p,\ e) \\
script & \text{wenn } type = \textsc{script},\ (\text{'script'},\ script) \in Attr \\
nop & \text{sonst}
\end{cases}
\end{aligned}
$$

Die Abbildung der Service-Tasks in einem Ad-Hoc-Subprozess unterscheidet sich von der normaler Tasks, da hier keine Elemente innerhalb des aktuellen Plans erzeugt werden; stattdessen werden aus den jeweiligen Services sowie den Assignments innerhalb des Tasks weitere Pläne abgeleitet. Die Input- und Output-Parameter sowie Precondition und Effect des Plans werden hierbei vom Service übernommen, wobei einzelne Variablen gegebenenfalls entsprechend der Assignments des Tasks instanziiert werden.

Ad-hoc Service Task $\Longrightarrow$ Plan: Sei $task = (id,\ \textsc{service},\ Ass,\ Attr) \in O$ ein Service Task innerhalb eines Ad-hoc Subprozesses mit ('impl', *service*) $\in Attr$ und $service = (name,\ pt,\ i,\ o,\ p,\ e)$, sodass $task \Longrightarrow Z$ Dann gilt:

$$
\begin{aligned}
task \implies\ & plan, \text{ wobei} \\
& plan = (id_name,\ In,\ Out,\ pre,\ \mathit{eff},\ Z) \\
& In = \{x \mid x \in i \wedge \nexists\ (x,\ y,\ z) \in Ass\} \\
& Out = \{y \mid y \in o \wedge \nexists\ (x,\ y,\ z) \in Ass\} \\
& pre = p, \text{ wobei Klauseln auf } In \text{ beschränkt werden} \\
& \mathit{eff} = e, \text{ wobei Klauseln auf } Out \text{ beschränkt werden}
\end{aligned}
$$

Intermediate Events im normalen Prozessverlauf unterscheiden sich nicht wesentlich von Tasks, und so fällt auch ihre Abbildung sehr ähnlich aus. Der Schwerpunkt liegt hier-

bei auf „passiven“ oder „abwartenden“ Tätigkeiten, etwa bis eine bestimmte Nachricht eingetroffen oder eine Wartezeit überschritten ist.

Event $\Longrightarrow$ verschiedene Skript-Elemente: Sei $event = (id,\ type,\ Ass,\ Attr) \in O$ ein Intermediate Event mit Typ $type$, Assignments Ass und weiteren Attributen $Attr$. Dann gilt:

$$
\begin{aligned}
event \quad \Longrightarrow \quad & seq(ass_1^b,\ \ldots,\ ass_n^b,\ X,\ ass_1^a,\ \ldots,\ ass_n^a), \text{ wobei} \\
& ass_i^b \in \{assign(prop,\ expr) \mid (prop,\ expr,\ \textsc{before}) \in Ass\} \\
& ass_i^a \in \{assign(prop,\ expr) \mid (prop,\ expr,\ \textsc{after}) \in Ass\} \\
& X = \begin{cases}
wait(t) & \text{wenn } type = \textsc{timer},\ (\text{‘time’},\ t) \in Attr \\
receive(m) & \text{wenn } type = \textsc{message},\ (\text{‘msg’},\ m) \in Attr \\
while(\neg c,\ nop) & \text{wenn } type = \textsc{rule},\ (\text{‘rule’},\ c) \in Attr \\
nop & \text{sonst}
\end{cases}
\end{aligned}
$$

Diese Regel gilt hierbei ausschließlich für Intermediate Events, sowohl für solche im normalen Prozessablauf als auch für an einen Subprozess angeheftete Event-Handler (siehe $eh_i \Longrightarrow X_i$ in Abbildungsregel *Subp. mit Event-H.*). Start- und End-Events wurden in Abschnitt 7.3.1 betrachtet und auf entsprechende Reaktionsregeln sowie Ein- und Ausgabeparameter der Pläne abgebildet.

7.3.3. Zusammenfassung und Fazit

Mit den hier diskutierten Strukturen und Elementen lassen sich die wichtigsten Aspekte der Geschäftsprozesse auf entsprechende Multiagenten-Systeme und deren Verhalten abbilden. Weitere Elemente, bzw. Typen, die an dieser Stelle nicht betrachtet wurden, da dies unverhältnismäßige Erweiterungen des Formalismus erfordert hätte, sich in der Praxis jedoch ebenfalls einfach umsetzen lassen, sind:

- *Call Activity:* Eine Call-Activity wird auf eine Subroutine abgebildet und kann auf diese Weise an anderer Stelle im Prozess wiederverwendet werden.
- *Link Event:* Zusammengehörige Link-Events werden über eine Preprocessing-Regel durch einen Sequence Flow verbunden.

- *Error Event:* Unter Verwendung des in vielen modernen Programmiersprachen verfügbaren Exception-Handlings wird eine Exception geworfen und ggf. durch ein anderes Error-Boundary-Event aufgefangen.

- *Terminate Event:* Der entsprechende Agent wird beendet und alle aktuell ausgeführten Pläne werden abgebrochen.

Durch die hier vorgestellten Abbildungsregeln lassen sich insbesondere auch die Organisations-Elemente des Prozesses auf entsprechende Agentenrollen und Pläne abbilden. Der Kontrollfluss innerhalb der Pools wird auf entsprechende Kontrollstrukturen der Agenten-Pläne abgebildet, wobei von der Verfügbarkeit der üblichen Strukturen, wie etwa Sequenzen, Schleifen und Verzweigungen, ausgegangen wird. Aspekte wie Planung und Ziele können über den Ad-Hoc-Subprozess modelliert werden, wobei die Tasks im Inneren des Subprozesses die für die Erfüllung des Ziels verfügbaren Pläne darstellen. In der hier beschriebenen Abbildung müssen die Tasks hierfür den *Service*-Typ haben, sodass sie über diese semantisch beschrieben sind.

Im folgenden Kapitel wird beschrieben, wie die Abbildung von Prozessdiagrammen auf Multiagenten-Systeme umgesetzt wurde, und wie sie in der praktischen agentenorientierten Softwareentwicklung eingesetzt werden kann.

Kapitel 8.

Methodik und Implementierung

Über die im vorigen Kapitel beschriebene Abbildung ist es möglich, komplexe Multiagenten-Systeme mithilfe von Prozessmodellen zu beschreiben. Jedoch können auch mit den vorgestellten Erweiterungen nicht alle Aspekte eines Multiagenten-Systems (oder eines anderen Softwaresystems) sinnvoll mit Prozessen beschrieben werden – manche Aspekte liegen schlichtweg nicht im Anwendungsbereich der Prozessmodellierung.

Besonders gut mit Prozessen zu modellieren sind naturgemäß Aspekte wie Verhaltensweisen und insbesondere Interaktionen zwischen Agenten: Abläufe und Verzweigungen innerhalb von Plänen und Verhaltensweisen, deren Zusammenspiel durch Orchestrierung einzelner Dienste, sowie nicht zuletzt die Kommunikation zwischen Teilnehmern eines Interaktionsprotokolls. Hingegen ist die Prozessmodellierung nicht dazu geeignet, Datentypen oder Ontologien zu modellieren, oder die Verteilung der einzelnen Agenten auf verschiedene Computersysteme zu beschreiben. Und auch die Prozesse selbst lassen sich nur auf relativ hohem Abstraktionsniveau sinnvoll graphisch modellieren; auf niedriger Abstraktionsebene, etwa zur Beschreibung eines komplexen Algorithmus, eignet sich die Prozessmodellierung nur bedingt. Solche Arbeiten gehen einem erfahrenen Entwickler mit textuellen Programmiersprachen leichter von der Hand.

Während die Prozessmodellierung auf hohem Abstraktionsniveau also gut eingesetzt werden kann, sind klassische Methoden der (agentenorientierten) Softwareentwicklung für diese anderen Gesichtspunkte besser geeignet.

In diesem Kapitel wird beschrieben, wie die Prozessmodellierung und die Abbildung der Prozesse auf Multiagenten-Systeme in den weiteren Entwicklungsprozess integriert werden und diesen sinnvoll ergänzen kann. Danach wird die prototypische Implementierung eines Prozessmodellierungswerkzeugs beschrieben, in dem die Abbildung von Prozessmodellen auf Agenten und die Methodik umgesetzt wurden.

8.1. Methodik zur prozessorientierten Entwicklung von Multiagenten-Systemen

Wie bereits dargelegt eignet sich die Prozessmodellierung insbesondere für die Beschreibung des Multiagenten-Systems auf hohem Abstraktionsniveau: Für die Abfolge der einzelnen Aktionen, die die Verhaltensweisen der Agenten ausmachen, für Kommunikation und Interaktionen zwischen den Agenten, und für die Beziehungen der Agenten untereinander, wobei für diesen letzten Aspekt die Erweiterung um die Usecase-artigen *Business Process System* Diagramme (Abschnitt 6.2) herangezogen wird.

Hingegen lassen sich die folgenden Aspekte nur unzureichend mit BPMN oder mit Prozessmodellen im Allgemeinen darstellen:

- Die Beschreibung von Datentypen oder Ontologien,
- die Organisation der Agenten und deren Verteilung auf Computersysteme,
- sowie die detaillierte Beschreibung von Berechnungen und Algorithmen.

Für diese Arbeiten gibt es geeignetere Notationen und Werkzeuge. In diesem Abschnitt soll daher eine Methodik beschrieben werden, wie die Prozessmodellierung effektiv und nicht zuletzt effizient in den weiteren Agenten-Entwicklungsprozess integriert werden kann (siehe [93]). Sie gliedert sich in die folgenden Schritte (Abbildung 8.1):

1. *Use Cases und Ontologien*: Wie in der Softwareentwicklung üblich, beginnt der Prozess mit der Identifikation von Anwendungsfällen (Use Cases) und der ontologischen Beschreibung der Domäne.
2. *Geschäftsprozessmodellierung*: Im nächsten Schritt werden die einzelnen Anwendungsfälle mit jeweils einem Geschäftsprozessdiagramm spezifiziert, wobei jeder Akteur durch einen Pool repräsentiert wird.
3. *Von Prozessen zu Agenten*: Aus diesen Prozessmodellen, den Use Cases und den Akteuren wird ein Rollenmodell sowie Verhaltensweisen und Regeln für die einzelnen Agenten des Multiagenten-Systems abgeleitet oder generiert.
4. *Ergänzung und Verfeinerung*: Die abgeleiteten Artefakte werden um solche Aspekte, die sich in BPMN nicht geeignet modellieren lassen, ergänzt und verfeinert.

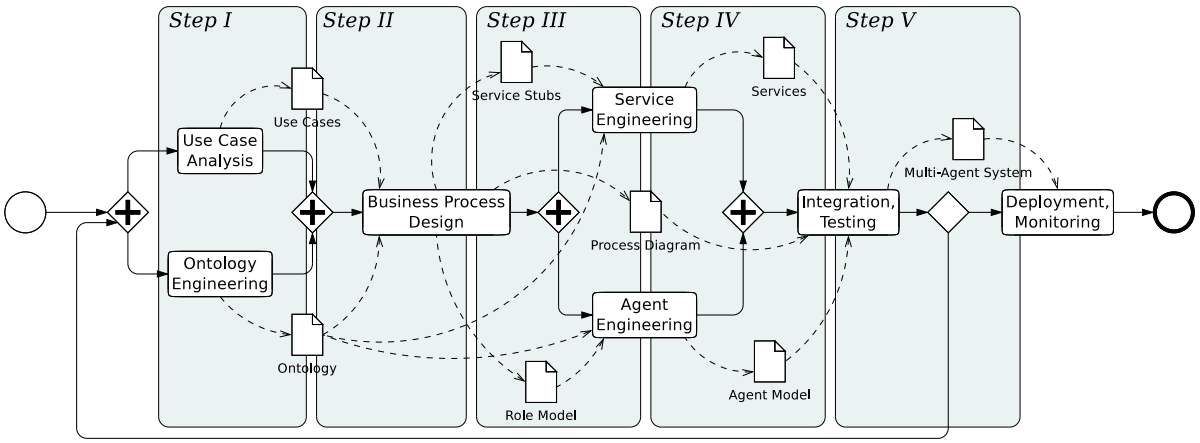


Abbildung 8.1.: Methodik zur prozessorientierten Entwicklung von Multiagenten-Systemen, dargestellt als BPMN-Prozessdiagramm. [93]

5. *Integration, Testen, Deployment*: Schließlich werden die einzelnen Komponenten zum vollständigen Multiagenten-System integriert und getestet. Je nach Testergebnis werden Teile des Entwicklungsprozessen wiederholt, bevor das fertige System schließlich deployed wird.

Im Folgenden sollen die Schritte der Methodik ausführlich beschrieben werden.

8.1.1. Use Cases und Ontologien

Der Entwicklungsprozess beginnt mit der Identifikation der verschiedenen Anwendungsfälle (Use Cases), die das Multiagenten-System behandeln soll. Als graphische Repräsentation werden Usecase-Diagramme nach Art der UML [119] verwendet. Hierbei entspricht jeder *Actor* im Usecase-Diagramm einer Rolle im Multiagenten-System (wobei diese Rolle wiederum die Schnittstelle für einen menschlichen Benutzer darstellen kann), und jeder *Use Case* einem Prozess, in den die jeweiligen Akteure involviert sind.

Neben dem üblichen Nutzen, einen sehr abstrakten Überblick über das System und somit einen Einstiegspunkt in den Entwicklungsprozess zu bieten, dienen die Usecase-Diagramme hier auch dazu, eine Verbindung zwischen den einzelnen Prozessen und den jeweiligen Prozessmodellen herzustellen. Mit Version 2 wurde die BPMN um *Conversation-Diagrams* erweitert, die eine ähnliche Funktion haben [2, 118]. In dieser Arbeit (bzw. dem darin verwendeten Modellierungswerkzeug) werden hierfür dennoch weiterhin Usecase-Diagramme der UML verwendet, da diese besonders intuitiv verständlich und – als Bestandteil der UML – auch weit verbreitet sind.

Parallel zu den Anwendungsfällen sollten auch die Domänen-Ontologien identifiziert und spezifiziert werden. Im Rahmen dieser Methodik spielt es jedoch keine Rolle, ob hierfür einfache Klassen erstellt werden, wie in der objektorientierten Programmierung üblich, beispielsweise mit UML Klassendiagrammen [119], oder ob spezielle Ontologie-Beschreibungssprachen wie OWL [99] verwendet werden.

Während die Modellierung der Ontologie einen wichtigen Aspekt des Entwicklungsprozesses darstellt, stellt die prozessorientierte Entwicklung diesbezüglich keine besonderen weiteren Anforderungen, weswegen an dieser Stelle nicht weiter auf diesen Aspekt eingegangen wird.

8.1.2. Geschäftsprozessmodellierung

Im nächsten Schritt wird, basierend auf den zuvor identifizierten Anwendungsfällen, eine Reihe von Prozessdiagrammen erstellt. Hierbei entspricht jeder Use Case einem Prozessdiagramm, in dem jeder an diesem Use Case beteiligte Akteur durch einen Pool repräsentiert wird. In diesen Pools wird hierbei nicht der komplette Lifecycle der Agenten modelliert, sondern nur ihr Verhalten im Kontext dieses einen Anwendungsfalles, insbesondere ihre Interaktionen mit den anderen beteiligten Akteuren.

Die Prozessdiagramme bzw. die einzelnen Pools sollten hierbei mindestens jene Aktivitäten und Verzweigungen enthalten, die für die Kommunikation mit den anderen Akteuren relevant sind. In der BPMN-Spezifikation wird dies als der *public process* bezeichnet [118, Kapitel 10]. Pools, die keinen Teil des zu entwickelnden Systems darstellen – beispielsweise solche, die bereits bestehende Agenten repräsentieren – können auch als *Black Boxes* dargestellt werden. Hierbei werden nur ein- und ausgehende Nachrichten, jedoch kein interner Kontrollfluss dargestellt. Die Inhalte der Nachrichten sowie Variablen und Bedingungen an Verzweigungen und Schleifen werden über Klassen und Ausdrücke der im vorigen Schritt spezifizierte Domänen-Ontologie beschrieben.

Natürlich können die Prozessdiagramme weiter detailliert werden und auch solche Aktivitäten und Verzweigungen enthalten, die nicht für die Kommunikation und Interaktion mit anderen Akteuren relevant sind (der *private process*). Hier muss im Einzelfall abgewogen werden, ob der zusätzliche Modellierungsaufwand gerechtfertigt ist, also bis wohin die Modellierung als Prozess einfacher ist, als die Implementierung in Code, und ob die zusätzlichen Details zum Verständnis des Anwendungsfalles beitragen, oder diesem möglicherweise sogar schaden. Eine komplexe, in sich abgeschlossene Berechnung sollte beispielsweise eher mit einer einzelnen Aktivität dargestellt und später im generierten Programmcode implementiert werden.

8.1.3. Von Prozessen zu Agenten

Die Prozessdiagramme werden nun dazu genutzt, entsprechend der im vorigen Kapitel beschriebenen Abbildung Teile des Multiagenten-Systems zu erzeugen. Hierbei können insbesondere die folgenden Aspekte aus den Modellen abgeleitet werden:

- Aus den in den Prozessdiagrammen modellierten Pools werden die Verhaltensweisen der einzelnen Agenten im Rahmen der jeweiligen Anwendungsfälle abgeleitet, was insbesondere ihre Interaktionen mit anderen Agenten mit einschließt.
- Basierend auf den Start-Events, mit denen die Workflows in den einzelnen Pools eingeleitet werden, werden Reaktionsregeln abgeleitet, über die die verschiedenen Verhaltensweisen ausgelöst werden.
- Aus den Akteuren ergeben sich Agentenrollen, die die entsprechenden Verhaltensweisen und Reaktionsregeln implementieren.

Die Abbildung von Prozessen auf Agenten kann hierbei sowohl manuell als auch automatisch umgesetzt werden. Ein solches Werkzeug zur automatisierten Abbildung auf das JIAC-Agentenframework wird in Abschnitt 8.2 beschrieben.

Das resultierende Multiagenten-System ist jedoch noch nicht vollständig: Je nachdem, wie detailliert die Prozesse modelliert wurden, müssen die Verhaltensweisen der Agenten noch ergänzt bzw. vervollständigt werden. Auch können aus den Akteuren in den Prozessdiagrammen lediglich Rollen, aber keine konkreten Agenten, geschweige denn deren konkrete plattformabhängige Konfiguration abgeleitet werden.

8.1.4. Ergänzung und Verfeinerung

In den ersten drei Schritten wurden Prozessmodelle erstellt und dazu verwendet, Agentenrollen und deren Verhaltensweisen abzuleiten, wobei insbesondere die Interaktionen zwischen den Agenten sehr anschaulich dargestellt und in eine entsprechende Implementierung überführt werden kann. In diesem Schritt werden die Ergebnisse der Abbildung überarbeitet und um Aspekte erweitert, die sich in den Prozessmodellen nicht angemessen beschreiben lassen.

- Wie bereits beschrieben ist es nicht sinnvoll, *low-level*-Algorithmen oder implementierungsnahe Aspekte wie etwa GUI-Aufrufe in den Prozessdiagrammen zu beschreiben. Solche Aspekte sollten in den Prozessen abstrakt als einzelne Aktivitäten dargestellt und nun im generierten Code implementiert werden.

- Ebenfalls nicht im Anwendungsbereich der Prozessmodellierung liegt die Konfiguration der einzelnen Agenten. Die generierten Rollen müssen also noch konkreten Agenten zugewiesen und im Netzwerk verteilt werden.

Die einzelnen Komponenten des Multiagenten-Systems sind nun vollständig und können miteinander integriert, getestet und deployed werden.

8.1.5. Integration, Testen, Deployment

Abschließend müssen die einzelnen Teile zu einem Gesamtsystem integriert und getestet werden – in dieser Hinsicht unterscheidet sich die Methodik nicht von anderen Entwicklungsprozessen. Die Zuordnung der verschiedenen Verhaltensweisen zu den einzelnen Agenten (bzw. deren Rollen) wurde bereits aus den Modellen abgeleitet, sodass diese nur noch an solche Komponenten angebunden werden müssen, die in den Prozessmodellen nicht enthalten sind. Beim Testen wiederum können die Prozessdiagramme als Dokumentation des gewünschten Systemverhaltens von Nutzen sein.

Schlagen die Tests fehl oder wird ein Fehler in den ursprünglichen Prozessmodellen entdeckt, müssen diese ggf. überarbeitet und die Komponenten des Multiagenten-Systems neu von den Modellen abgeleitet werden. Da die erzeugten Komponenten – insbesondere die Verhaltensweisen – aber in der Regel um einzelne Aspekte erweitert werden müssen, ist es wichtig, dass diese Teile bei einer automatischen Code-Erzeugung erhalten bleiben. Noch weiter geht das sogenannte *round-trip engineering*, bei dem die Änderungen und Erweiterungen am generierten Code automatisch in das Modell übernommen werden. Im Kontext dieser Methodik ist dies jedoch nicht unbedingt wünschenswert, da manche Aspekte bewusst nicht in das Prozessdiagramm aufgenommen wurden, um dieses übersichtlich und verständlich zu halten.

Schließlich wird das Multiagenten-System deployed. Hier könnte das Prozessmodell – entsprechende Werkzeugunterstützung vorausgesetzt – dazu verwendet werden, den Zustand der einzelnen Agenten zu überwachen, indem die aktuell ausgeführten Aktivitäten hervorgehoben und die Belegung der im Prozess definierten Variablen angezeigt werden. Das Prozessdiagramm könnte also zur Laufzeitüberwachung des Systems eingesetzt werden, oder sogar um korrigierend in das System einzugreifen.

8.1.6. Fazit

In diesem Abschnitt wurde eine Methodik beschrieben, mit der die Prozessmodellierung und die Abbildung von Prozessen auf Multiagenten-Systeme in den gesamten Agenten-

Entwicklungsprozess integriert werden kann. Während sich für viele Konzepte Entsprechungen in der BPMN finden, lässt sich die Prozessmodellierung doch am effektivsten auf einem relativ hohen Abstraktionsniveau einsetzen: Die Prozessmodelle sind sehr gut dafür geeignet, das Verhalten und die Interaktionen der Agenten zu beschreiben, doch für viele Details der Implementierung sind klassische bzw. agentenspezifische Modelle und Beschreibungssprachen nach wie vor besser geeignet.

Während die Methodik selbst, ebenso wie die zugrunde liegende Abbildung, auf verschiedene Agenten-Frameworks angewendet werden kann, werden doch gewisse Anforderungen an Modellierungswerkzeuge gestellt, damit die Methodik effektiv und effizient eingesetzt werden kann – beispielsweise um die Abbildung von Prozessen auf Agentensysteme zu automatisieren. Im folgenden Abschnitt soll daher ein Werkzeug beschrieben werden, mit dem ein verteiltes System nach der hier dargestellten Methodik modelliert und in ein ausführbares Multiagenten-System übertragen werden kann.

8.2. Implementierung des Modellierungswerkzeugs

Die in dieser Arbeit beschriebenen Konzepte zur prozessorientierten Entwicklung von Multiagenten-Systemen wurden im BPMN-Editor *VSDT (Visual Service Design Tool)* umgesetzt. Das VSDT wurde ursprünglich als Diplomarbeit implementiert [86] und in den folgenden Jahren in verschiedenen Projekten am DAI-Labor der TU-Berlin weiterentwickelt und ergänzt [87, 88, 93, 89, 153, 164]. Als Zielplattform für die Abbildung auf Multiagenten-Systeme hat sich hierbei das Framework *JIAC V* angeboten, das bereits im Abschnitt 2.4 beschrieben wurde.

Für diese Arbeit wurde ein eigener BPMN-Editor entwickelt, anstatt auf ein bestehendes Werkzeug aufzusetzen. Zum einen wird ein geringfügig erweitertes Prozessmodell genutzt, sodass ein generischer BPMN-Editor nicht über alle notwendigen Attribute verfügt hätte. Zum anderen sollte auch der Editor selbst einige Funktionen bieten, die in keinem anderen BPMN-Werkzeug gefunden werden konnten, insbesondere Schnittstellen zu diversen anderen Werkzeugen des JIAC-Frameworks.

In den folgenden Abschnitten soll das VSDT – insbesondere diejenigen Aspekte, die für diese Arbeit besonders relevant sind – sowie das darin enthaltene Transformationsframework, mit dessen Hilfe die Abbildungen umgesetzt wurden, beschrieben werden.

8.2.1. Umsetzung des Metamodells

Zur Zeit der Entstehung des VSDT bot die BPMN-Spezifikation weder eine formale Beschreibung des Metamodells, noch eines Austauschformats, etwa als XML Schema Definition (XSD). Somit musste die BPMN-Spezifikation zunächst in einem BPMN-Metamodell umgesetzt werden. Da das VSDT als GMF-basierter Eclipse-Editor implementiert werden sollte (siehe Abschnitt 8.2.2), wurde das Metamodell mit dem *Eclipse Modelling Framework (EMF)* als *Ecore*-Modell umgesetzt.

Das BPMN-Metamodell wurde, mit einigen wenigen Auslassungen und Vereinfachungen, sehr nahe an Version 1 der BPMN-Spezifikation [117] umgesetzt und enthält insbesondere alle zur Erzeugung von ausführbarem Code notwendigen Attribute. *Containment*-Beziehungen wurden so gewählt, dass sie den Verschachtelungen der graphischen Elemente im Editor entsprechen. Die verschiedenen in der BPMN-Spezifikation genannten Einschränkungen wurden als Java-Constraints im Modell und im Editor umgesetzt. Später wurde das Metamodell schrittweise an neuere Versionen der BPMN-Spezifikation angepasst und um neu hinzugekommene Elemente ergänzt.

Darüber hinaus wurde das BPMN-Metamodell auch an anderer Stelle erweitert:

- Noch bevor mit Version 2 der BPMN-Spezifikation [118] Conversation-Diagramme eingeführt wurden, wurde das Metamodell um ein neues Wurzel-Element erweitert: Das *Business Process System*, das verschiedene Business Process Diagrams und Participants, die zu einem Gesamtsystem gehören, enthält und in Usecase-artigen Diagrammen in Beziehung zueinander setzt.

- *Messages* in der BPMN beziehen sich fast ausschließlich auf das Aufrufen und die Rückgabewerte von Webservices und vergleichbaren Technologien. Um auch Interaktionsprotokolle für Agenten sinnvoll darstellen zu können, wurde das Modell um einen zusätzlichen Implementierungs-Typen komplementär zum Service-Aufruf erweitert: Den *Nachrichtenkanal* (Abbildung 8.2). Nachrichtenkanäle haben einen Payload, einen Absender und einen Empfänger, die allesamt als BPMN-Properties modelliert sind und als ebensolche in Assignments und Expressions verwendet werden können. Auf diese Weise kann nicht nur der Payload der Nachricht an eine lokale Variable gebunden werden, sondern es kann auch der Absender gespeichert und als Empfänger einer späteren Nachricht verwendet werden. Als Empfänger können sowohl Agenten-IDs als auch beliebige Nachrichtenkanäle verwendet werden.

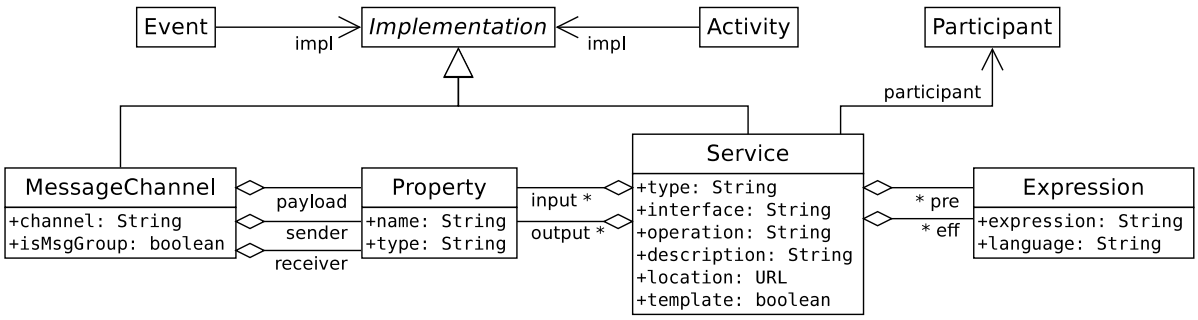


Abbildung 8.2.: Klassendiagramm: Messages und Services im BPMN-Metamodell. [48]

- *Service-Beschreibungen* in der BPMN orientieren sich stark an den in BPEL verwendeten Webservices, und verfügen über entsprechende Attribute. Einige dieser Eigenschaften, etwa *interface*, *operation* oder *participant*, sind allgemeingültig genug, dass sie auch für andere Arten von Services verwendet werden können, etwa für Agenten. Zugleich fehlt den Diensten jedoch eine semantische Beschreibung der Vorbedingungen und Effekte, wie sie etwa für die automatische Planung benötigt wird. Die Dienstbeschreibungen wurden daher um entsprechende Attribute erweitert (Abbildung 8.2). Die für die semantischen Beschreibungen verwendete Sprache, beispielsweise SWRL (Semantic Web Rule Language), ist dabei unerheblich.

- Um neben den eigentlichen Prozessen auch die in diesen Prozessen verwendeten Ausdrücke, etwa in Zuweisungen und Bedingungen, in verschiedene Zielsprachen übersetzen zu können, wurde ein einfaches Modell (abstrakte und konkrete Syntax) für Ausdrücke (Expressions) entwickelt: Die *VSDT Expression Language (VXL)* [87]. Ausdrücke werden weiterhin als einfache Zeichenketten gespeichert, können aber, sofern sie in der VXL angegeben wurden, interpretiert und in andere Sprachen übersetzt werden.

- In Hinblick auf die Abbildung auf ausführbare Programmiersprachen, die in der Regel einer Blockstruktur folgen, wurde das Metamodell um eine Reihe von *Struktur-Elementen* erweitert (Abbildung 8.3). Die Prozess-Modelle werden weiterhin als gerichtete Graphen modelliert, können aber – für die Transformation in ausführbaren Programmcode (siehe Abschnitt 8.2.3) oder zur Strukturanalyse – in eine Baumstruktur überführt werden.

Das Modell enthält jedoch nicht von vollen Umfang der BPMN 2.0. Insbesondere liegt

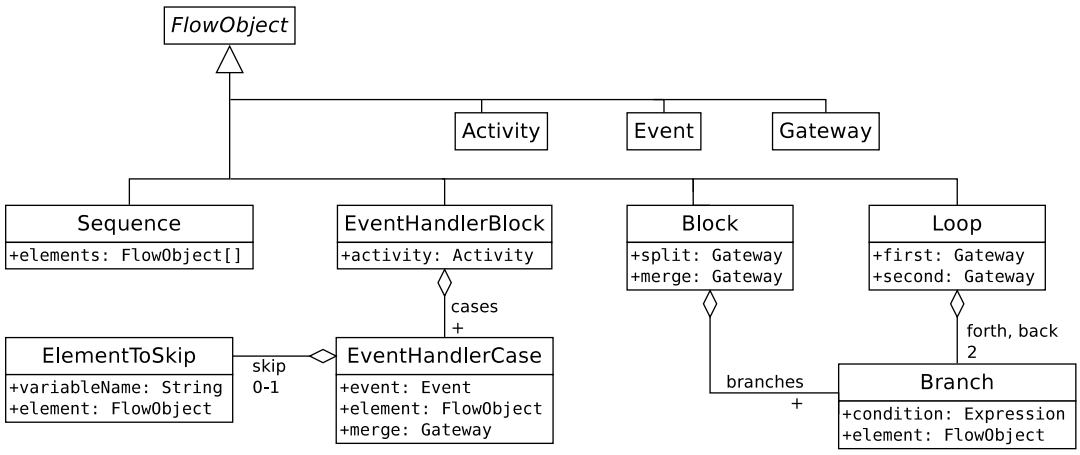


Abbildung 8.3.: Klassendiagramm: Struktur-Elemente im BPMN-Metamodell.

der Fokus weiterhin auf den aus Version 1 bekannten „Collaboration“-Diagrammen; „Choreography“-Diagramme werden nicht unterstützt. Alles in allem ist das im VSDT verwendete BPMN-Metamodell umfangreicher und komplexer als das in Abschnitt 6.2 beschriebene Prozess-Metamodell, ist aber mit diesem konsistent.

8.2.2. Umsetzung des BPMN-Editors

Der BPMN-Editor wurde mithilfe des *Graphical Modeling Framework (GMF)*[1] als Plugin für die Java-basierte Entwicklungsumgebung *Eclipse* umgesetzt. Auf diese Weise konnte das Grundgerüst für den graphischen Editor schnell erstellt und bei Änderungen am Metamodell angepasst werden. Abbildung 8.4 zeigt einen Screenshot der aktuellen Version des VSDT.

Während ein mit GMF generierter Diagrammeditor bereits die wichtigsten Funktionen bereit stellt, musste das Modellierungswerkzeug noch um einige Funktionen erweitert werden, sodass es für die effiziente Erstellung von Prozessdiagrammen eingesetzt werden kann. Mehr noch als bei Editoren für textuelle Programmiersprachen spielt bei graphischen Editoren die *Usability* eine entscheidende Rolle: Während die graphische Darstellung übersichtlicher ist als reiner Quelltext, stellt die Erstellung der Prozessdiagramme und der dahinter liegenden Semantik – beispielsweise der Properties und Assignments – einen deutlichen Mehraufwand gegenüber textueller Programmiersprachen dar.

Die größten Herausforderungen hierbei sind:

[1] Eclipse GMF: `https://www.eclipse.org/modeling/gmp/`

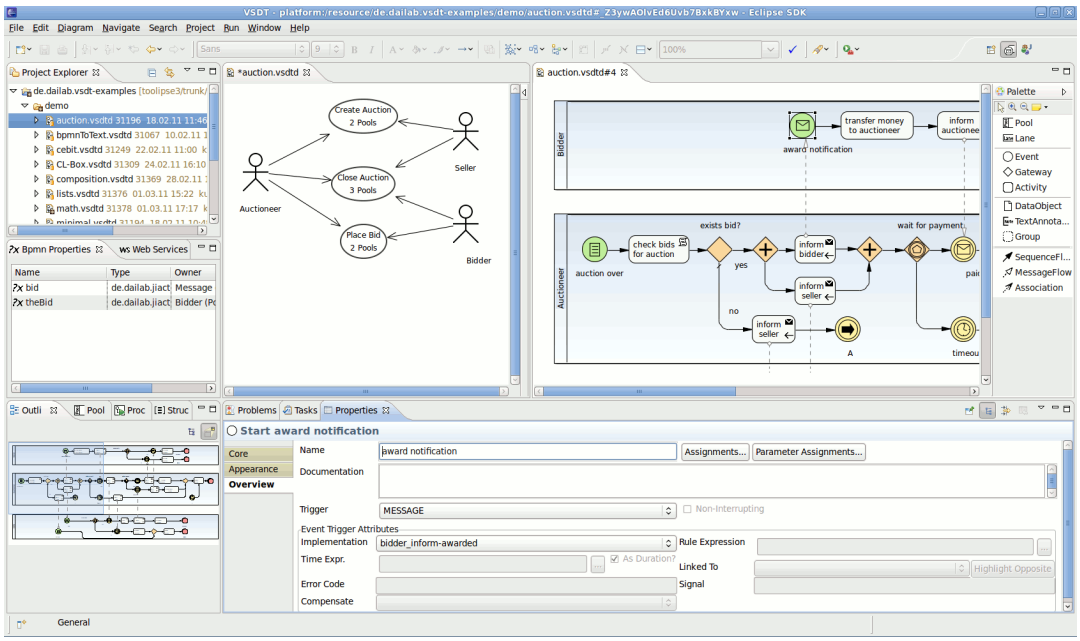


Abbildung 8.4.: Screenshot des Visual Service Design Tools (VSDT), Version 1.4.0.

1. der Mehraufwand beim Modellieren durch häufigen Wechsel von Maus zu Tastatur, verbunden mit langen Mauswegen,

2. die Erstellung und Verwaltung von Elementen ohne Repräsentation im Diagramm, die doch entscheidend sind für die Semantik des Prozesses, etwa Properties, Assignments und Services, und

3. die Modellierung des Prozesses in solcher Weise, dass er sich in ein Programm einer typischen strukturierten Programmiersprache übertragen lässt.

Um diesen Problemen zu begegnen, wurden an dem BPMN-Editor verschiedene Erweiterungen vorgenommen [87], unter anderem

- Aktionen zum Einfügen und automatischen Ausrichten einzelner Elemente und größerer Prozess-Bausteine, wie Schleifen und Verzweigungen, die sich auch über die Tastatur bedienen lassen,

- übersichtlichere und einfacher navigierbare Property-Sheets und einheitliche Dialoge zum Erstellen nicht sichtbarer Elemente, sowie

- zusätzliche Ansichten (Views), über die das Prozessdiagramm probeweise strukturiert und/oder interpretiert werden kann.

Wie im vorigen Abschnitt 8.1 beschrieben, ist es dennoch nicht sinnvoll, das gesamte Agentensystem über die Prozessmodelle zu beschreiben. Nicht zuletzt für Aspekte wie die Beschreibung der Datentypen/Ontologien, die Verteilung und Konfiguration der Agenten auf verschiedenen Computern, oder für *low-level*-Programmierung sind andere Werkzeuge und Methoden besser geeignet.

Einen wichtigen Aspekt stellt somit auch die Integration des BPMN-Editors mit anderen Werkzeugen, die bei der Entwicklung von Agenten eingesetzt werden, dar [93]. Neben der im folgenden Abschnitt beschriebenen Transformation des Prozessmodells zu JIAC-Agenten, die in Form von Export-Wizards in die Eclipse-Umgebung integriert wurde, wurden weitere Funktionen zur Integration der Prozessmodellierung in die übrige JIAC-Werkzeugkette implementiert. Hier ist insbesondere der Import bestehender Dienste bzw. Dienstbeschreibungen aus aktuell laufenden JIAC-Anwendungen und aus dem *JIAC-AgentStore* [65] zu nennen, sowie die Möglichkeit, mit dem VSDT modellierte Dienste direkt auf dieselben aufzuspielen, oder zu deployen. Ferner können auch semantisch mit OWL-S und SWRL beschriebene Dienste aus dem *Semantic Service Manager (SSM)* [101] gesucht und in den aktuell bearbeiteten VSDT-Prozess importiert werden [48]. Um die generierten Agenten-Rollen zu konkreten Agenten zu aggregieren, integriert sich das VSDT mit dem *Agent World Editor (AWE)* [97].

8.2.3. Umsetzung des Transformations-Frameworks

Für die Umsetzung der in dieser Arbeit beschriebenen Abbildung wurde ein erweiterbares Transformations-Framework entwickelt, mit dem sich diese – ebenso wie Abbildungen auf andere Zielsprachen – effizient umsetzen lässt. Die einzelnen Transformationen werden hierbei in mehrere Stufen unterteilt. Typischerweise sind dies:

1. *Validierung*: Überprüfung des Prozessmodells auf mögliche Fehler
2. *Normalisierung*: Vereinfachung / Vereinheitlichung des Prozessmodells
3. *Strukturierung*: Überführung des Prozess-Graphen in eine Baumstruktur
4. *Abbildung*: Abbildung der Prozess-Elemente auf Elemente der Zielsprache
5. *Abschluss*: Abschließende Arbeiten, Speicherung der Ergebnisse

Die Abarbeitung der Stufen wird in Algorithmus 1 dargestellt. Hierbei ist *sourceObj* das Stammelement des zu transformierenden Prozessdiagramms und $targetDir$ das Ziel-

verzeichnis. Die Methoden *initializeMappingStages*, *createMappingWrapper* sowie *getResultSaver* werden von den entsprechenden Subklassen implementiert.

Algorithm 1 performTransformation(sourceObj, targetDir): boolean

```
wrapper ← createMappingWrapper(sourceObj)
noErrors ← true
stages[] ← initializeMappingStages()
for all stage ∈ stages do
  stage.initialize()
  noErrors ← noErrors ∧ stage.apply(wrapper)
end for
resultSaver ← getResultSaver()
noError ← noErrors ∧ resultSaver.apply(wrapper, targetDir)
return noErrors
```

Die einzelnen Stufen können hierbei als Pattern-Matching-Regeln oder als einfacher Visitor implementiert werden, wobei sich insbesondere für die Normalisierung und Strukturierung Pattern-Matching-Regeln anbieten, die in den folgenden Abschnitten näher beschrieben werden. Nach erfolgter Strukturierung kann die eigentliche Abbildung als einfacher Top-Down Visitor implementiert werden.

Die Anwendung der Regeln ist in Algorithmus 2 beschrieben. Die Regeln können hierbei in verschiedene „Layer“ gruppiert werden, wobei erst dann mit dem nächsten Layer begonnen wird, wenn keine weitere Regel des aktuellen Layers anwendbar ist.

Algorithm 2 transform(eObject, rules[][]): boolean

```
instanceMap ← createInstanceMap(eObject)
executedAnyRules ← false
for all layer[] ∈ rules do
  applicableRules ← queue(layer)
  while applicableRules ≠ ∅ do
    rule ← applicableRules.pop()
    if rule.apply(instanceMap) then
      executedAnyRules = true
      applicableRules ← queue(layer)
      instanceMap ← createInstanceMap(eObject)
    end if
  end while
end for
return executedAnyRules;
```

Die Methode *createInstanceMap* erstellt hierbei eine Zuordnung von Klassen zu In-

stanzen dieser Klassen, auf welcher in der Methode *rule.apply* das eigentliche Pattern-Matching angewandt wird (Algorithmus 3).

Algorithm 3 apply(instanceMap): boolean

$lhsVariables[\,] \leftarrow initLhsVars(instanceMap)$
$nacVariables[\,][\,] \leftarrow initNacVars(instanceMap)$
$match[\,] \leftarrow Matcher.findMatches(lhsVariables,\ nacVariables)$
if $match \neq \varnothing$ **then**
  $execute(match)$
end if
return $match \neq \varnothing$

Die Graphtransformationsregeln wurden zunächst mit dem *Tiger EMF Model Transformation* Framework (EMT, [9]) umgesetzt. Später wurde, den Kernprinzipien von EMT folgend, eine eigene Pattern-Matching-Engine implementiert. In dieser wurde das Pattern-Matching per *Left Hand Side (LHS)* und optionaler *Negative Application Conditions (NACs)* beibehalten, der Effekt der Regel wird jedoch nicht länger in einer *Right Hand Side (RHS)* beschrieben, sondern als eine Methode, die gewöhnlichen Java-Code enthält, und somit flexibler ist und beispielsweise weitere Verzweigungen und Schleifen definieren kann. Diese *execute*-Methode muss neben *initLhsVars* und *initNacVars* von den jeweiligen Normalisierungs- und Strukturierungsregeln implementiert werden.

In der ersten Version des VSDT wurde die Abbildung noch vor der Strukturierung ausgeführt [86], was später zu der jetzigen Form vereinfacht wurde [88]. Die ersten drei Stufen – Validierung, Normalisierung und Strukturierung – sind hierbei weitestgehend unabhängig von der Zielsprache und können für die meisten Transformationen direkt wiederverwendet oder ggf. geringfügig erweitert werden.

In den folgenden Abschnitten sollen die einzelnen Stufen der Transformation und die typischerweise in diesen Stufen durchgeführten Aktivitäten näher erläutert werden. Anschließend wird ein Überblick über die Umsetzung der eigentlichen Abbildung der Prozessmodelle auf JIAC-Agenten geworfen.

Validierung

In der ersten Phase der Transformation werden die in der BPMN-Spezifikation gegebenen Modellierungsregeln überprüft und sämtliche Elemente des Prozessmodells dahingehend untersucht, ob alle notwendigen Attribute und Referenzen gesetzt sind; so muss etwa für einen Service-Task eine entsprechende Service-Implementierung angegeben werden.

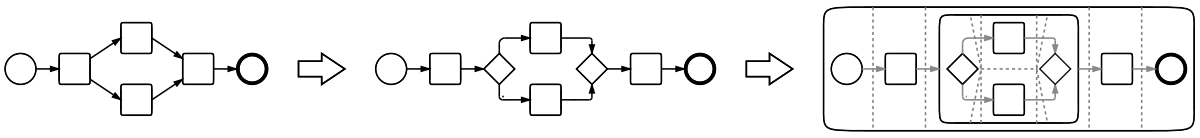


Abbildung 8.5.: Normalisierung und Strukturierung eines einfachen Prozesses. [88]

Indem diese Überprüfungen in eine eigene Phase ausgegliedert werden, wird die Implementierung der eigentlichen Abbildung einfacher und übersichtlicher. Zudem sind diese Überprüfungen für alle Abbildungen notwendig und können auf diese Weise einfach wiederverwendet werden.

Normalisierung

Nach der Validierung wird das Diagramm in eine einheitliche Form überführt, oder „normalisiert“ [45, 88]. Die Semantik des Diagramms bleibt bei der Normalisierung erhalten, jedoch folgt das Diagramm nach erfolgter Normalisierung strengeren Regeln als es nach der BPMN-Spezifikation notwendig wäre (Abbildung 8.5) .

Der Zweck der Normalisierung besteht darin, das Diagramm in eine einheitliche Form zu bringen, wodurch in der nächsten Phase, der Strukturierung, weit weniger unterschiedliche Fälle betrachtet werden müssen, um Blöcke und Schleifen zu erkennen. Beispielsweise darf eine Activity laut BPMN-Spezifikation über mehrere ein- und ausgehende Sequence Flows verfügen; die Verwendung eines entsprechenden Gateways für Zusammenführungen und Verzweigungen ist somit in vielen Fällen optional. Durch die Normalisierung würde an dieser Stelle automatisch ein entsprechendes Gateway eingefügt werden. Im Einzelnen werden die folgenden Normalisierungsregeln verwendet:

- *Connect Link Rule*: Zusammengehörende Link Intermediate Events werden mit einem Sequence Flow verbunden.
- *Insert Empty Rule*: Zwischen zwei direkt aufeinander folgende Gateways wird ein *no-op*-Task eingefügt (analog auch für Aktivitäten mit Event Handler und Kombinationen von beidem).
- *Insert Gateway Rule*: Hat eine Aktivität oder ein Event mehrere ein- oder ausgehende Sequence Flows, wird davor bzw. dahinter ein Gateway eingefügt und die Sequence Flows an das Gateway umgelegt.

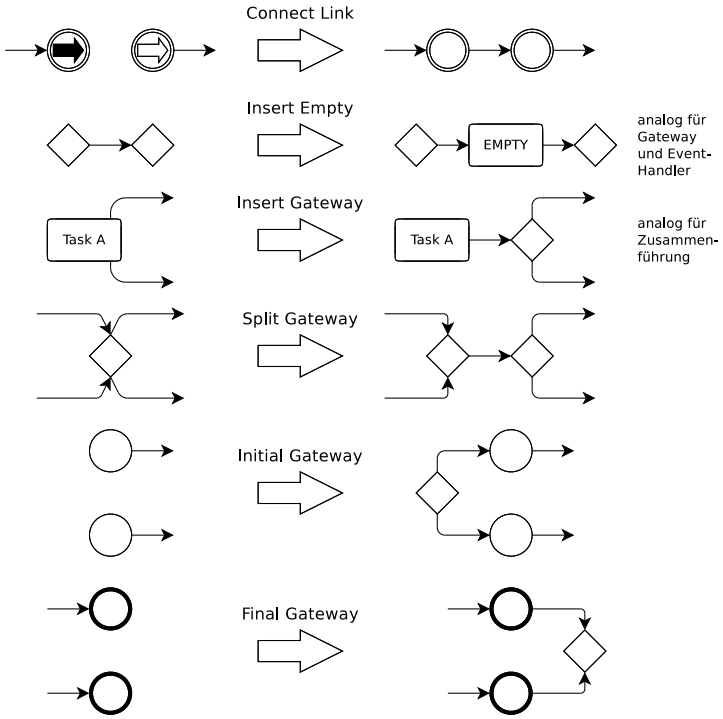


Abbildung 8.6.: Normalisierungsregeln. Von Oben: Connect Link, Insert Empty, Insert Gateway, Split Gateway, Initial Gateway, und Final Gateway Rule.

- *Split Gateway Rule*: Gateways, die sowohl mehrere ausgehende als auch mehrere eingehende Sequence Flows haben, werden aufgespalten.
- *Initial Gateway Rule*: Hat der Prozess mehrere Start Events, wird ein spezielles Gateway eingefügt und mit den Start Events verbunden.[2]
- *Final Gateway Rule* (analog zur Initial Gateway Rule)

Die Normalisierung wurde mit Pattern-Matching-Regeln umgesetzt (Abbildung 8.6), die das gegebene Prozessmodell entsprechend der Normalisierungsvorschriften umbauen. Nach der Normalisierung sind in der folgenden Phase, der Strukturierung, weit weniger unterschiedliche Fälle zu betrachten, wodurch diese einfacher und weniger fehleranfällig wird.

[2]Diese Regel dient dazu, dass Prozesse, die über mehrere Start- aber nur ein End-Event verfügen, korrekt strukturiert werden können. Nach der Strukturierung wird das Gateway wieder entfernt.

Strukturierung

Wie in Abschnitt 3.3 beschrieben, muss der Prozessgraph zunächst strukturiert werden. Durch die Kapselung der Strukturierung in eine separate Phase der Transformation lässt sich dieser sehr komplexe Schritt in allen Transformationen für ähnlich strukturierte Zielsprachen wiederverwenden.

Diese Stufe folgt hierbei einer *Structure Identification* Strategie [103], wobei Teile des Prozessgraphen, die strukturierten Workflows entsprechen, identifiziert und durch die jeweiligen Elemente aus dem erweiterten BPMN-Metamodell (Abbildung 8.3) ersetzt werden [88]. Die folgenden Strukturierungsregeln werden verwendet:

- *Sequence Rule*: Zwei Flow Objects, die über einen Sequence Flow verbunden sind, werden in eine Sequence-Struktur gekapselt.
- *Loop Rule*: Zwei Gateways, die über genau zwei (komplexe) Flow Objects in entgegengesetzter Richtung verbunden sind, werden in einer Loop-Struktur gekapselt.
- *Block Rule*: Zwei Gateways, die über mindestens zwei (komplexe) FlowObjects in der gleichen Richtung verbunden sind, werden in einer Block-Struktur gekapselt. Weitere Pfade zwischen den Gateways werden identifiziert und ggf. dem Block hinzugefügt, oder die Gateways werden aufgespalten.
- *Overlapped Block Rule*: Einfach ineinander verschränkte Blöcke werden aufgelöst, wobei ein Teil des Workflows (bzw. eine entsprechende Subroutine im generierten Code) dupliziert werden muss.[3]
- *Boundary Event Block Rule*: Aktivitäten mit Event Handlern werden in einer EventHandlerBlock-Struktur gekapselt.
- *Boundary Event Comp Rule*: Compensation Event Handler und die dazugehörigen Compensation Activities werden in eine EventHandlerCase-Struktur gekapselt und dem EventHandlerBlock hinzugefügt.
- *Boundary Event Skip Rule*: Andere Arten von Event Handlern, zusammen mit den dazugehörigen Compensation-Workflows, werden in eine EventHandlerCase-Struktur gekapselt und dem EventHandlerBlock hinzugefügt. Der Abschnitt des

[3] Tatsächlich ist diese Regel eher eine Normalisierungs- als eine Strukturierungsregel; sie wird jedoch erst in dieser Phase ausgeführt, da die einzelnen Zweige des Blocks bereits strukturiert sein müssen.

Workflows hinter dem EventHandlerBlock, der bei diesem Event Handler übersprungen wird, wird in ein ElementToSkip-Struktur gekapselt und mit dem EventHandlerCase verknüpft.

- *Boundary Event Loop Rule*: Behandelt den Fall, dass der Kontrollfluss durch einen EventHandler vor den EventHandlerBlock zurück gelenkt wird. Ähnlich der Boundary Event Skip Rule, wobei das Ergebnis in eine Loop-Struktur gekapselt wird.

Wie die Normalisierung wurde auch die Strukturierung über Pattern-Matching-Regeln umgesetzt (Abbildung 8.7). Da die Struktur-Elemente des erweiterten BPMN-Metamodells wiederum Flow Objects sind, können die Regeln iterativ angewendet werden, bis der gesamte Prozessgraph strukturiert ist oder aufgrund struktureller Mängel keine weiteren Regeln angewendet werden können. Das resultierende strukturierte BPMN-Modell hat eine Baumstruktur, die im folgenden Abschnitt, der eigentlichen Abbildung der einzelnen Elemente, einfach traversiert werden kann.

Abbildung

Die Abbildung ist der zentrale Bestandteil der Transformation von Prozessen auf Agentensysteme (oder jede andere Zielsprache) und zugleich der einzige Teil, der für jede Zielsprache individuell implementiert werden muss. Während hier – ebenso wie für die Normalisierung und Strukturierung – Pattern-Matching-Regeln definiert werden können, hat sich ein einfacher Visitor an dieser Stelle als praktikabler erwiesen.

Mit diesem kann das strukturierte Prozessmodell *top-down* von Wurzelelement ausgehend durchgegangen und jedes Element auf das entsprechende Element der Zielsprache abgebildet werden. Das Zielmodell wird hierbei parallel zu dem durchwanderten Prozessmodell aufgebaut. Die Abbildung der konkreten Elemente des Quellmodells auf die des Zielmodells kann während der Transformation in einem assoziativen Array (einer Tabelle oder Map) festgehalten werden.

Die Abbildung der Prozessmodelle auf Multiagenten-Systeme wurde auf mehrere verschiedene Arten umgesetzt: Zum einen als Abbildung auf JADL-Dienste und auf JIAC AgentBeans, und zum anderen als JIAC-basierter Prozess-Interpreter. Diese Implementierungen und ihre jeweiligen Vor- und Nachteile werden im Abschnitt 8.3 beschrieben.

Abschluss

In der abschließenden Phase der Transformation kann der aus dem Prozessmodell generierte Programmcode bereinigt und optimiert werden. So können etwa während der Nor-

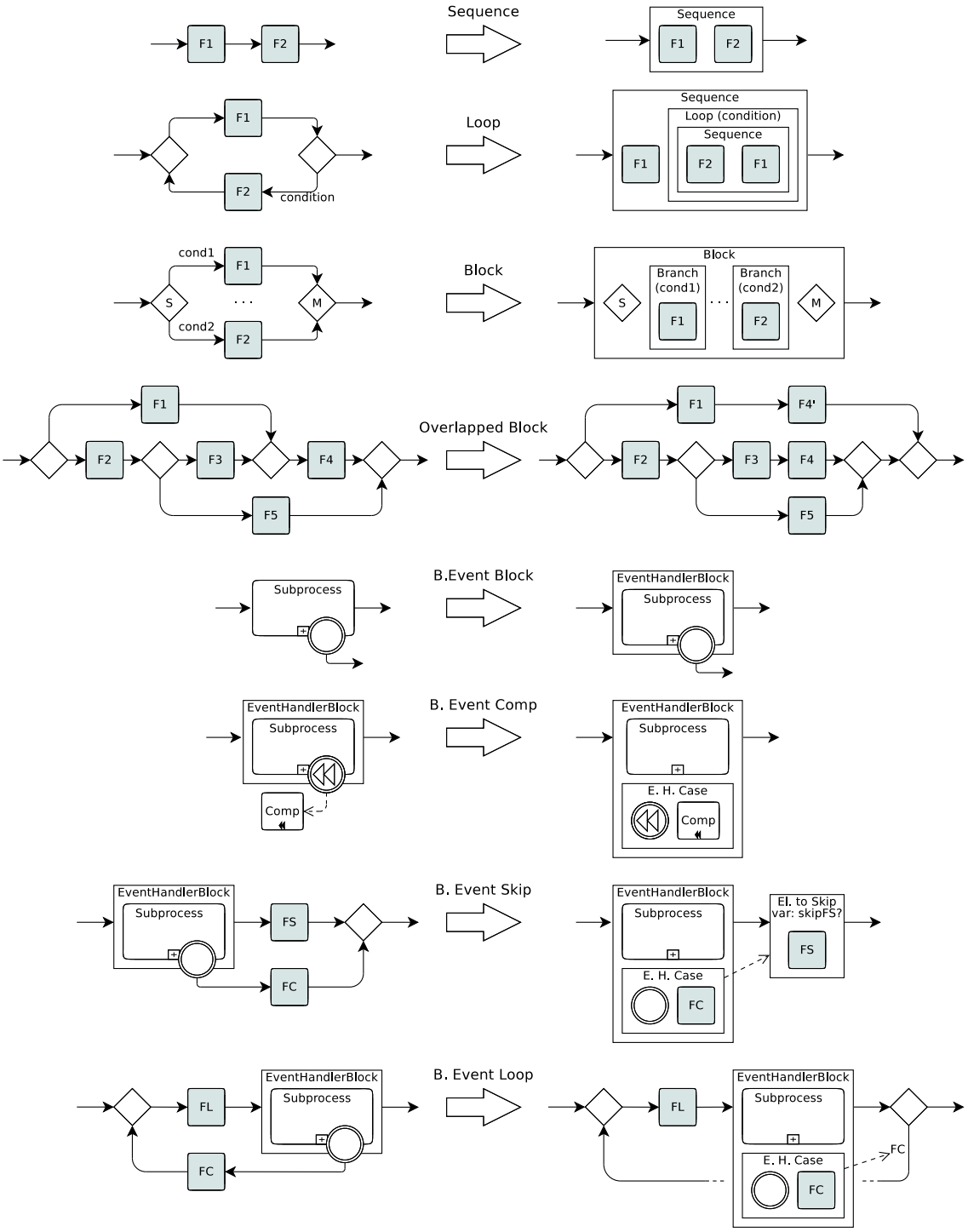


Abbildung 8.7.: Strukturierungsregeln. Von Oben: Sequence, Loop, Block, Overlapped Block, Boundary Event Block, -Comp, -Skip, und -Loop Rule. Grau hinterlegte Knoten sind beliebige Flow Objects, einschließlich bereits identifizierter Struktur-Elemente.

malisierung eingefügte Platzhalter-Elemente entfernt, geschachtelte Sequenzen zu einer Sequenz vereinigt oder Sequenzen mit nur einem einzelnen Element aufgelöst werden.

Schließlich werden die einzelnen Ergebnisse der Transformation in Dateien geschrieben und gespeichert.

8.3. Implementierung der Abbildung

Basierend auf dem im letzten Abschnitt beschriebenen Transformationsframeworks wurde die Abbildung von BPMN auf Agenten in unterschiedlicher Ausführung für verschiedene Zielsysteme umgesetzt. Im Folgenden sollen einige dieser Transformationen beschrieben werden.

Die unterschiedlichen Implementierungen setzen hierbei verschiedene Schwerpunkte, und haben unterschiedliche Stärken und Schwächen, auf die im nächsten Kapitel näher eingegangen werden soll. Hieraus ergeben sich auch einige Besonderheiten hinsichtlich der Entwicklungsmethodik. So ist es teils erlaubt, die aus den Prozessen abgeleiteten Verhaltensweisen direkt auf bereits laufende Agenten zu deployen; in diesem Fall wäre das Agentensystem und die Ontologie bereits vorhanden und der Prozess beschränkt sich auf die Entwicklung der Verhaltensweisen und deren Deployment und Monitoring. Demgegenüber entspricht das Vorgehen im Falle der Transformation auf JIAC AgentBeans am genauesten der vorgestellten Methodik.

Vor den hier beschriebenen Umsetzungen wurden außerdem Transformationen nach BPEL [112] und einer früheren Version von JADL für JIAC IV [149] erstellt [86, 88]. Die Transformation nach BPEL sollte hierbei in erster Linie als *proof-of-concept* für das Transformationsframework dienen, und die Transformation nach JADL wurde bald aufgegeben, da JIAC IV nicht länger unterstützt wurde und durch JIAC V abgelöst wurde. Diese beiden Transformationen werden an dieser Stelle daher nicht weiter betrachtet.

8.3.1. Erzeugung von JADL-Diensten für JIAC V

Die Abbildung wurde zunächst als Transformation zu *JADL++* -Services umgesetzt.[4] JADL++ ist eine auf Prinzipien der serviceorientierten Architektur aufbauende Skriptsprache für JIAC-Agenten [66]. JADL-Skripte können einem mit der entsprechenden *JADL Interpreter AgentBean* ausgestatteten Agenten bei der Initialisierung oder zur

[4] Ist im Folgenden von „JADL“ die Rede, so ist, sofern nicht explizit von JIAC IV gesprochen wird, immer JADL++ für JIAC V gemeint.

Laufzeit übergeben werden. Die darin enthaltenen Services werden vom Agenten veröffentlicht und sind nach außen hin von in AgentBeans implementierten Actions nicht zu unterscheiden.

Die BPMN-Prozesse können hierbei sehr direkt auf die verschiedenen Sprachelemente von JADL abgebildet werden, beispielsweise zum Versenden und Empfangen von Nachrichten. Die Abbildung ist somit in vielerlei Hinsicht ähnlich der Abbildung nach BPEL [118, Kapitel 14]. Ein Überblick der verschiedenen Aspekte der Abbildung wird in Tabelle 8.1 dargestellt.

Tabelle 8.1.: Überblick der Abbildung von BPMN auf JADL++-Services

BPMN-Element	***JADL++*-Konzept**
Participant	Agentenrolle in generierter AWE-Datei
Pool	JADL-Services (einer pro Start-Event des Pools)
Workflow	Kontrollfluss des Services
Start-Events	Drools-Starter-Regeln
Tasks	verschiedene JADL-Sprachelemente
Subprozesse	Kontrollfluss, *inline* im Service
Boundary-Events	—
Properties	Variablen im Service

Umsetzung der einzelnen Elemente

Jeder Prozess wird auf einen in sich abgeschlossenen JADL-Service abgebildet.

Sequenzen, Verzweigungen, parallele Ausführung und Schleifen können direkt auf die entsprechenden JADL-Strukturen abgebildet werden. Das Event-basierte XOR-Gateway wird hierbei mit einem speziellen `switch/case`-Konstrukt abgedeckt, das BPELs `pick`-Element nachempfunden ist.

Sämtliche Tasks und Subprozesse werden direkt „in-line“ in den jeweiligen Service eingebettet. Auf diese Weise stellen sie einen eigenen Variablen-Scope dar, in dem etwa temporäre Variablen für Dienst-Parameter deklariert werden können, und können zugleich auf alle auf Prozess-Ebene definierten Variablen zugreifen. Event Handler (an eine Aktivität angeheftete Intermediate Events) können mit JADLs derzeitigem Sprachumfang nicht abgebildet werden.

Aktivitäten und Ereignisse zum Versenden und Empfangen von Nachrichten werden auf die `send`- und `receive`-Elemente von JADL abgebildet; im JADL-Interpreter werden diese durch das Versenden und Empfangen entsprechenden JIAC-Messages umgesetzt,

wobei JIACs Kommunikationsinfrastruktur genutzt wird. Nachrichten werden hierbei ausschließlich an Gruppenadressen versendet bzw. von diesen empfangen; Nachrichten an individuelle Agenten werden von JADL noch nicht unterstützt. Ein Service-Task wird auf das `invoke`-Element abgebildet, das intern als Aufruf einer JIAC-Action (oder eines anderen JADL-Services) umgesetzt wird. Die Ein- und Ausgabewerte des Dienstes werden entsprechend der jeweiligen Start- und End-Events gewählt. Analog wird bei einem Message-Start-Event der Payload der empfangenen Nachricht als Dienst-Parameter verwandt. Weiterhin lassen sich über den Script-Task beliebige Stücke JADL-Skript einfügen.

Start Events und Konfiguration

Die von den Start-Events abgeleiteten Reaktionsregeln können nicht direkt in JADL umgesetzt werden; stattdessen werden Regeln für die Drools-Regelengine[5] erzeugt. Eine solche Regelengine kann in einen JIAC-Agenten integriert und mit dessen Faktenbasis synchronisiert werden [68] und sorgt mit den entsprechenden Regeln dafür, dass etwa beim Eintreffen einer Nachricht ein Dienst aufgerufen wird, indem die Regel eine entsprechende *DoAction* (Intention) in den Agenten-Speicher einfügt.

Um die Participants auf Agentenrollen abzubilden wird eine JIAC-Konfigurationsdatei erzeugt. In dieser wird für jeden der Participants ein Agent konfiguriert, welcher mit einer JADL-Interpreter-Bean und einer RuleEngine-Bean ausgestattet ist. Die erzeugten JADL-Services und Drools-Regeln werden dann in der Konfigurationsdatei von den Interpreter- bzw. RuleEngine-Beans der entsprechenden Agenten referenziert.

Fazit

Als vom Agenten interpretierte Skriptsprache bietet JADL nicht alle Möglichkeiten, die JIAC bzw. Java selbst bietet; so ist etwa der Zugriff auf Java-Bibliotheken (beispielsweise für GUI- oder Datenbank-Operationen) nur eingeschränkt möglich. Die Möglichkeiten der Abbildung sind somit durch den relativ eingeschränkten Umfang der JADL-Skriptsprache eingeschränkt. Zugleich bietet JADL zahlreiche high-level-Konstrukte an, etwa für Kommunikation und Event-Handling. Die generierten Dienste sind somit vergleichsweise übersichtlich und eignen sich am besten zur Beschreibung von Verhalten auf hoher Abstraktionsebene.

[5] Drools: `http://www.drools.org/`

JADL bietet zudem den Vorteil, dass die erzeugten Dienste, zusammen mit den jeweiligen Starter-Regeln, zur Laufzeit auf einen Agenten deployed werden können. Auf diese Weise kann das Verhalten eines bestehenden Agenten dynamisch angepasst werden. Die Abbildung der Participants auf Agentenrollen kommt in diesem Fall nicht zum Tragen; stattdessen nimmt der Interpreter-Agent, auf den die Dienste und Regeln installiert wurden, das entsprechende Verhalten an.

8.3.2. Erzeugung von JIAC V-AgentBeans

Die Abbildung auf JIAC V-AgentBeans wurde begonnen, als die Abbildung und Transformation nach JADL bereits in einem weit fortgeschrittenen Stadium war. Während JADL gut geeignet ist, Verhalten und Dienste auf einem hohen Abstraktionsniveau zu beschreiben, sind JIAC AgentBeans vielseitiger einsetzbar. Manche Aspekte des Agenten lassen sich auch ausschließlich über AgentBeans beschreiben, und nicht über JADL, sodass komplementär zur Abbildung auf JADL auch eine Abbildung auf JIAC AgentBeans entwickelt wurde [89, 90].

Die erste Version der Abbildung wurde von P. S. Tan implementiert [153]. Hierbei wird das strukturierte Prozessdiagramm zunächst in ein Zwischenmodell übertragen, das die Struktur der AgentBeans und der darin enthaltenen Methoden widerspiegelt. Anschließend wird aus diesem Zwischenmodell Java-Code generiert. Ein Beispiel für die Verwendung der Abbildung auf JIAC V AgentBeans und den dabei generierten Quellcode wird in Anhang A gezeigt.

Konzeptionell orientiert sich die Abbildung an *WADE* [27] (Abschnitt 4.3.1): Für die einzelnen Pools des Prozessdiagramms wird jeweils eine JIAC AgentBean (d. h. eine Java-Klasse) erzeugt, wobei der Workflow des Prozesses auf eine *Workflow-Methode*, und die einzelnen Aktivitäten auf weitere *Activity-Methoden* abgebildet werden. Die Workflow-Methode dient hierbei als Eintrittspunkt zur Ausführung des Prozesses und ruft die einzelnen Activity-Methoden entsprechend des ursprünglichen Workflows auf. Eine Übersicht über die Abbildung kann Tabelle 8.2 entnommen werden.

Die Workflow-Methode

Für die Workflow-Methode können die strukturierten Prozessdiagramme direkt in entsprechende Verzweigungen und Schleifen übertragen werden. Der generierte Code fällt dementsprechend natürlicher und verständlicher aus, vergleichbar mit handgeschriebenem Code.

Tabelle 8.2.: Überblick der Abbildung von BPMN auf JIAC AgentBeans

BPMN-Element	**AgentBeans-Konzept**
Participant	Agentenrolle (implizit, nicht generiert)
Pool	Workflow AgentBean, bestehend aus folgenden Teilen
Workflow	strukturierte Workflow-Methode
Start-Events	Mechanismen zum Auslösen der Workflow-Methode
Tasks	Activity-Methoden für die eigentliche Arbeit
Subprozesse	innere Klasse, Struktur analog zur Workflow Bean
Boundary-Events	Event-Handler-Threads, zum Unterbrechen der Akt.
Properties	Variablen, in entsprechendem Scope

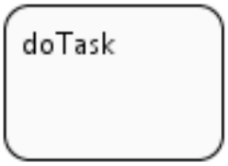
Zugleich bietet BPMN eine große Ausdruckskraft. Insbesondere können die folgenden Aspekte umgesetzt werden:

- Durch die Verwendung von Threads können Teile des Workflows parallel ausgeführt werden, etwa bei der Verwendung von AND- oder OR-Gateways.
- Subprozesse werden auf interne Klassen abgebildet, die wiederum eine Methode für den Workflow und eine für jede einzelne Aktivität aufweisen.
- Event Handler werden auf Threads abgebildet, die nebenläufig zum Haupt-Thread des Workflows laufen und diesen bei Eintreffen des jeweiligen Ereignisses (bspw. einer Nachricht) abbrechen können.
- Ähnlich wird mit Event-basierten XOR-Gateways verfahren; der Haupt-Thread führt in diesem Fall nur eine Warteschleife aus.

Properties und Assignments

Properties (Variablen) können in BPMN für einzelne Aktivitäten, für Subprozesse oder für den gesamten Prozess definiert werden. Dementsprechend werden die Properties von Prozessen auf Klassenvariablen der AgentBean, Properties von Subprozessen auf Variablen der eingebetteten Subprozess-Klassen und Properties einzelner Aktivitäten auf lokale Variablen innerhalb der jeweiligen Activity-Methode abgebildet. Die Sichtbarkeit, die sich hieraus ergibt, entspricht der im BPMN-Diagramm.

Assignments (Zuweisungen) sind immer an eine Aktivität oder ein Event gebunden und werden in den jeweiligen Activity-Methoden ausgeführt; entsprechend ihrer *assign time* vor oder nach dem die eigentliche Aktivität beschreibenden Code.




```
1 public void doTask(){
2   [declaration of properties]
3   [start assignments]
4   [special mapping depending on task-type]
5   [end assignments]
6 }
```

Abbildung 8.8.: Abbildungsregel für die Activity-Methode. [153]

Activity-Methoden

Die einzelnen Activity-Methoden haben weder Parameter noch Rückgabewert und folgen immer dem gleichen Schema (Abbildung 8.8):

1. *Properties*: Als erstes werden Variablen für alle Properties der Activity deklariert.
2. *Start Assignments*: Als nächstes folgen Assignments, bei denen als *assign time* „before“ eingestellt ist, z. B. um Werte an die Parameter eines Services zu binden.
3. *Activity Body*: Es folgt der Code für die eigentliche Aktivität, beispielsweise der Aufruf eines Services, Versand oder Empfang einer Nachricht.
4. *End Assignments*: Schließlich werden Assignments mit *assign time* „after“ ausgeführt, z. B. um den Rückgabewert eines Services an eine Property zu binden.
5. *Loop*: Ist das Loop-Attribut der Aktivität gesetzt, wird der gesamte Inhalt der Activity-Methode in eine entsprechende While-Schleife verpackt.

Genau wie bei der Abbildung auf JADL++ kann hierbei zum Senden und Empfangen von Nachrichten (Send/Receive-Task) und für Service-Aufrufe (Service-Task) auf die Kommunikationsinfrastruktur von JIAC zurückgegriffen werden. Im Falle eines Script-Tasks wird an dieser Stelle das im Prozessdiagamm hinterlegte Code-Fragment integriert. Mit Timer-Intermediate-Events kann die Ausführung eine Zeit lang verzögert werden. User-Tasks können dazu genutzt werden, eine spezielle Action zum Abfragen von Nutzer-Eingaben zur Laufzeit aufzurufen. Eine Übersicht über die verschiedenen Tasks und Events und die Abbildung auf entsprechenden auf JIAC-Konzepte kann Tabelle 8.3 entnommen werden.

Event Handler

Wie oben beschrieben werden Event Handler (Intermediate-Events die an einen Task oder Subprozess angeheftet sind) auf Threads abgebildet, die parallel zur eigentlichen

Tabelle 8.3.: Abbildung von Aktivitäten und Ereignissen auf JIAC-Konzepte

Task/Event	**Abgebildet auf JIAC-Konzept**
Service Task	JIAC Action-Aufruf (synchron)
Send Task	Nachricht an Nachrichtengruppe verschicken (asynchron)
Receive Task	Nachricht von Nachrichtengruppe empfangen (blockierend)
Script Task	Code-Fragment in Activity-Methode einbinden
User Task	Action für Nutzer-Interaktion aufrufen
Message Event	Sende (throwing) oder empfange (catching) Nachricht
Timer Event	Ausführung für bestimmte Dauer aussetzen
Rule Event	Ausführung aussetzen bis Regel-Bedingung wahr ist
Link Events	Per Sequence Flow verbunden (Normalisierung)

Aktivität ausgeführt werden. Die Aktivität selbst wird ebenfalls in einem Thread ausgeführt. Der Event-Handler-Thread kann nun periodisch überprüfen, ob das jeweilige Ereignis eingetroffen ist, und den Activity-Thread gegebenenfalls unterbrechen.

In der Workflow Methode wird der Activity-Thread zusammen mit den verschiedenen Event-Handler-Threads gestartet. Sobald der Activity-Thread abgeschlossen ist (normal beendet oder abgebrochen) wird überprüft, ob einer der Event-Handler ausgelöst wurde, und der Workflow wird dementsprechend fortgesetzt.

Start Events und Konfiguration

Verschiedene Mechanismen zum Starten der modellierten Prozesse entsprechend der jeweiligen Start-Events können direkt in den AgentBeans umgesetzt werden:

- Bei einem unspezifizierten Start-Event (Typ *none*) wird die Workflow-Methode über die `doStart()`-Methode aufgerufen und somit ausgeführt, wenn der Agent gestartet wird.
- Bei einem Message-Start-Event mit *Service*-Implementation wird die Workflow-Methode über den `@Expose`-Mechanismus direkt als JIAC-Action mit entsprechenden Parametern und Rückgabewerten veröffentlicht.
- Bei einem Message-Start-Event mit *Message Channel*-Implementierung wird ein `SpaceObserver` eingerichtet, der den Speicher des Agenten auf entsprechende JIAC-Messages überwacht und ggf. die Workflow-Methode startet.

- Bei einem Timer-Start-Event wird in der regelmäßig ausgeführten `execute()`-Methode der AgentBean eine Abfrage eingerichtet, die die aktuelle Systemzeit mit der letzten Ausführungszeit des Workflows abgleicht und diesen ggf. startet.

Neben diesen Mechanismen werden außerdem die Parameter der Workflow-Methode so eingestellt, dass – je nach Start-Event – die empfangene Nachricht, der aktuelle Zeitpunkt, oder die Service-Parameter an die Methode übergeben werden können.

Schließlich wird, analog zur Abbildung auf JADL-Services, eine Agenten-Konfiguration erstellt, der jedoch statt der JADL-Interpreter-Bean und der RuleEngine-Bean nur die hier erzeugten AgentBeans hinzugefügt werden.

Fazit

Dadurch, dass in den AgentBeans zusätzlich zu den für Agenten typischen Konzepten, wie dem Versenden von Nachrichten, auch der volle Sprachumfang von Java zur Verfügung steht, kann prinzipiell alles ausgedrückt werden, was in BPMN modelliert werden kann. Während die Workflow-Methoden – abhängig von der Komplexität des Workflows – mitunter schwierig zu durchschauen sein können, sind die Prozesse durch die Auslagerung der Aktivitäten in separate Methoden und die Verwendung einer abstrakten Oberklasse, in der u. a. die Event-Handler-Klassen definiert sind, dennoch gut verständlich:

Die generierten AgentBeans erweitern die abstrakte Klasse `AbstractWorkflowBean` aus dem Paket `de.dailab.jiactng.agentcore.workflow` im JIAC-AgentCore-Modul. Diese Klasse enthält einige Hilfsmethoden für das Versenden und Empfangen von Nachrichten und zum Aufrufen von Actions, sowie die Definitionen der verschiedenen Event-Handler-Klassen. Durch diese speziell für diese Abbildung entwickelte Klasse bleibt der generierte Code vergleichsweise kompakt und übersichtlich.

Durch die Verwendung des JET-Frameworks[6] ist es zudem möglich, einzelne Activity-Methoden zu verändern oder zu überschreiben und den Code nach einer Änderung im Prozessdiagramm neu zu generieren, ohne dass die manuellen Änderungen verloren gehen, indem die `@generated`-Annotation an der entsprechenden Methode entfernt wird. Ein echtes *round-trip engineering*, bei dem Änderungen am generierten Code zurück in das Prozessmodell übertragen werden, ist jedoch nicht möglich.

Ein Nachteil gegenüber der Abbildung auf JADL-Services ist, dass die AgentBeans nicht zur Laufzeit deployed oder verändert werden können. Durch die hohe Ausführungs-

[6] JET (Java Emitter Templates): `http://www.eclipse.org/modeling/m2t/`

geschwindigkeit des Java-Codes und die gute Erweiterbarkeit der erzeugten Module eignet sich die Abbildung auf JIAC-AgentBeans somit insbesondere für die Modellierung und Generierung der Kernkomponenten des Agentensystems, während die Abbildung auf JADL-Services am besten für Dienste auf höherem Abstraktionsniveau und für zur Laufzeit veränderliche Verhaltensweisen geeignet sind.

Ein mögliches Problem kann sich zudem aus der intensiven Nutzung von Threads ergeben. Alternativ hierzu könnte das Event-Handling gegebenenfalls auch in einem einzelnen, zentralen Thread gebündelt werden. Außerdem eignet sich dieser Ansatz weniger für die Ausführung sehr lange laufender Prozesse, da der Agent solange die Workflow-Methode dauert nicht den derzeitige Ausführungszyklus abschließen kann. Als Alternative hierzu kann die Abbildung auch in Form eines Interpreters implementiert werden, wobei in jedem Ausführungszyklus des Agenten nur eine einzelne Aktivität ausgeführt wird.

8.3.3. Direkte Interpretation der Prozesse

Die Abbildung in Form eines JIAC-basierten Prozess-Interpreter-Agenten wurde in erster Version von M. Voß implementiert [164]. Diese Umsetzung unterscheidet sich insofern fundamental von den beiden zuvor genannten, als dass in diesem Falle kein Quellcode einer anderen Sprache erzeugt wird, sondern die Prozessdiagramme direkt interpretiert werden [90]. Hierdurch fällt insbesondere der Schritt zur Strukturierung des Workflows weg.

Stattdessen werden die Prozessdiagramme an den Interpreter-Agenten übergeben, zusammen mit der Rolle (oder den Rollen), die der Agent in diesem Prozess übernehmen soll, woraufhin der Agent den aktuellen Zustand des Prozesses überwacht und nacheinander die verschiedenen Aktivitäten des Prozesses ausführt. Der Interpreter-Agent baut hierbei auf einen im VSDT integrierten BPMN-Interpreter auf (siehe [87, Abschnitt 5.3.3]), und erweitert diesen um die Schnittstellen zum Agenten, insbesondere zum Event-Handling und zur Kommunikation. Tabelle 8.4 zeigt eine Übersicht, wie die verschiedenen Teile des Prozesses im Interpreter umgesetzt werden.

Aufbau des Interpreters

Der Interpreter-Agent gliedert sich hierbei in drei Ebenen:

- *Interpreter Bean:* Stellt die Schnittstelle zum Agenten her, nimmt Prozesse zur Interpretation an und erzeugt für diese je eine Interpreter-Runtime.

Tabelle 8.4.: Überblick der Abbildung per BPMN-Interpreter-Agent

BPMN-Element	**Interpreter-Konzept**
Participant	Interpreter-Runtime für jede Rolle
Pool	Interpreter-Instanz für jeden laufenden Prozess
Workflow	Ausführungsreihenfolge im Interpreter
Start-Events	Interpreter Runtime erzeugt neue Instanz
Tasks	`doStep`-Methode der Interpreter-Instanz
Subprozesse	`doStep`-Methode der Interpreter-Instanz
Boundary-Events	Event-Handling-Layer der Interpreter-Runtime
Properties	Variablen-Context der Interpreter-Instanz

- *Interpreter Runtime:* Entspricht einem Participant im Prozessdiagramm, überwacht Start-Events der entsprechenden Pools und startet Interpreter-Instanzen.
- *Interpreter Instanz:* Kurzlebige Instanz zum einmaligen Interpretieren eines bestimmten Pools, kapselt dessen internen Zustand, führt Aktivitäten aus.

Die *Interpreter-Bean* stellt eine Reihe von JIAC-Actions zur Verfügung, über die neue Interpreter-Runtimes für einen bestimmten Participant in einem gegebenen Prozess-Modell gestartet und überwacht werden können. Vor allem aber stellt sie die Schnittstelle dar, über die diese Interpreter Runtimes mit dem JIAC-Agenten interagieren: Einerseits werden verschiedene Ereignisse an die Interpreter-Runtimes weitergegeben, wie etwa eintreffende Nachrichten und Action-Aufrufe; andererseits stellt sie Methoden bereit, über die die Interpreter-Runtimes auf Funktionen des Agenten zugreifen können, etwa um selbst Nachrichten zu verschicken. Außerdem sorgt sie dafür, dass für die Pools der Prozesse – also die Pläne des Agenten – entsprechende JIAC-Action-Beschreibungen erstellt und bekannt gemacht werden.

Jede *Interpreter-Runtime* ist für einen Participant eines Prozess-Diagrams zuständig, entspricht also einer Rolle, die der Interpreter-Agent spielt (ein Interpreter-Agent kann hierbei durchaus mehrere Rollen annehmen, auch im gleichen Prozess). Die Aufgabe der Interpreter-Runtime besteht darin, die Ereignisse, die von der Interpreter-Bean kommen, auszuwerten, und entsprechend der Start-Ereignisse der verschiedenen Pools des Participants neue Interpreter-Instanzen für eben diese Pools zu starten, sowie bereits laufende Instanzen zu überwachen und ggf. Ereignisse an diese weiter zu geben.

Jede *Interpreter-Instanz* ist für die Ausführung eines Pools (bzw. eines Plans des Agenten) zuständig. Hierzu wird in jedem Schritt der aktuelle Zustand des Prozesses berechnet – insbesondere die aktuelle Variablenbelegung und die als nächstes auszufüh-

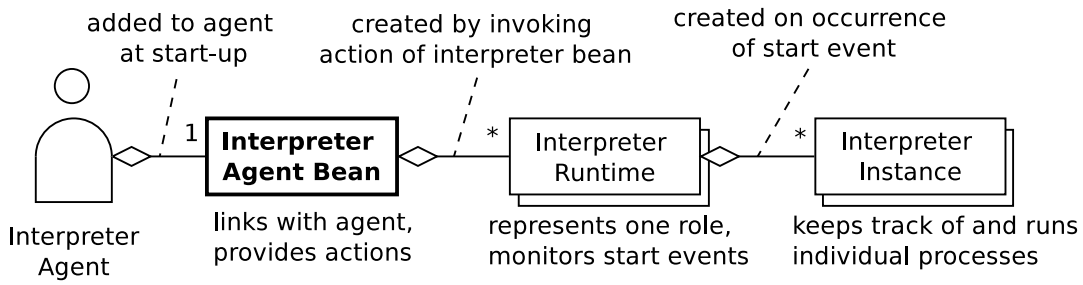


Abbildung 8.9.: Schematischer Aufbau des Prozess-Interpreter-Agenten. [90]

renden Aktivitäten. Diese Funktionen konnten aus dem bestehenden Prozess-Interpreter übernommen werden, der als Debugging-Werkzeug Teil des VSDT ist. Zuweisungen werden mittels einer Skriptsprache ausgewertet, und die Ergebnisse in einem *Context* gespeichert, der die aktuelle Variablenbelegung widerspiegelt. Für die eigentlichen Aktivitäten, etwa eine Nachricht zu verschicken oder einen Dienst aufzurufen, werden die entsprechenden Funktionen der Interpreter-Runtime und Interpreter-Bean aufgerufen. Ist der Workflow im Pool abgearbeitet, wird die Interpreter-Instanz aufgelöst, bis durch ein entsprechendes Ereignis eine weitere Instanz gestartet wird.

Abbildung 8.9 zeigt einen Überblick über die Komponenten des Interpreters.

Ablauf der Interpretation

Einem Agenten mit *Interpreter-Bean* muss zunächst mitgeteilt werden, welche Prozesse in welcher Rolle interpretiert werden sollen. Dies kann entweder über die Agenten-Konfiguration geschehen, oder über eine Action. Das Prozessmodell wird daraufhin geladen und für jede zu interpretierende Rolle wird eine *Interpreter-Runtime* erstellt.

Die Prozesse werden nicht sofort interpretiert. Zunächst werden lediglich die Start-Events ausgewertet, und die Interpreter-Bean beginnt, auf entsprechende Ereignisse zu horchen; außerdem werden für Service-Start-Events entsprechende JIAC-Actions angelegt und veröffentlicht. Registriert die Interpreter-Bean ein Ereignis, etwa eine bestimmte Uhrzeit, eine eintreffende Nachricht, oder ein Action-Aufruf, so wird dies an die zuständige Interpreter-Runtime weitergeleitet, und es wird für jeden Pool eine *Interpreter-Instanz* erstellt, dessen Start-Event dem eingetroffenen Ereignis entspricht. Abhängig vom Ereignis werden der Interpreter-Instanz Eingabeparameter übergeben und im internen Zustand des Interpreters wird das Start-Event als „aktiv“ markiert.

Nun wird in jedem Schritt des Ausführungszyklus des Agenten ein Schritt in diesem (und jedem anderen aktiven) Prozess ausgeführt: In der `execute`-Methode der

Interpreter-Bean wird jede Interpreter-Runtime angewiesen, (a) die Start-Events zu überprüfen um ggf. weitere Prozess-Instanzen zu starten, und (b) jede Prozess-Instanz einen Schritt ausführen zu lassen. Die Logik zum Auswählen der jeweils nächsten aktiven Elemente und für die Auswertung von Zuweisungen ist hierbei dem bestehenden, VSDT-eigenen Interpreter entliehen (siehe [87, Abschnitt 5.3.3]); die Interpreter-Runtime erweitert dies nur um die konkreten Aktionen, wie das Verschicken von Nachrichten oder das Aufrufen von Services, die wiederum an die Interpreter-Bean zurück delegiert werden. Ist jede Aktivität abgeschlossen und lassen sich keine weiteren aktivierten Aktivitäten ableiten, so gilt der Prozess als beendet und die Interpreter-Instanz wird aufgelöst. Die Interpreter-Runtime bleibt bestehen um ggf. weitere Instanzen des Prozesses zu starten.

Nutzerschnittstelle

Der Interpreter-Agent verfügt über eine spezielle Nutzerschnittstelle (UI), die gemeinsam mit dem Agenten gestartet und über die der Agent kontrolliert werden kann. Über die UI können andererseits die Aktionen zum Registrieren und Deregistrieren von Interpreter-Runtimes aufgerufen werden, zum anderen aber auch der aktuelle Zustand der einzelnen Interpreter-Instanzen abgerufen und dargestellt werden, insbesondere die zur Zeit ausgeführten Aktivitäten sowie die derzeitige Variablenbelegung.

Ferner ermöglicht es der Interpreter, während der Ausführung mit dem Nutzer in Interaktion zu treten.[7] Diese Funktion kann zum einen für *User Tasks* genutzt werden, und zum anderen um nach der (semantischen) Dienstsuche ggf. fehlende Parameter abzufragen. In beiden Fällen wird eine einfache UI angezeigt, in der der Nutzer seine Eingaben tätigen kann, wobei auch andere Abfragemechanismen genutzt werden können, etwa zur Integration mit einer bestehenden UI.

Fazit

Durch den Verzicht auf generierten Code und die damit einher gehende zusätzliche Abstraktionsschicht ist die Interpreter-AgentBean die direkteste Form der Abbildung des BPMN-Prozesses auf Agenten-Verhalten. Insbesondere ist es die einzige der hier vorgestellten Implementierungen, die auch die Graph-Struktur des Prozesses direkt abbilden, ohne die sonst notwendige Strukturierung.

Ein weiterer Vorteil ist, dass Prozesse dynamisch zur Laufzeit deployed werden können, wobei der Interpreter-Agent jeweils alle Prozesse einer Rolle übernimmt. Darüber

[7] Prinzipiell ist das gleiche auch mit den anderen Implementierungen möglich; für den Interpreter wurde diese Funktion jedoch im Rahmen des EMD-Projekts als erstes umgesetzt.

hinaus ist eine Integration mit dem im VSDT enthaltenen Interpreter-View denkbar, über die die vom Agenten gerade ausgeführten Prozesse und deren innerer Zustand im Modellierungswerkzeug überwacht werden könnten.

Ein möglicher Nachteil besteht darin, dass es ohne generierten Code keine Möglichkeit gibt, die zu interpretierenden Prozesse vor der Ausführung weiter zu bearbeiten. Das gesamte Verhalten muss vollständig im Prozessdiagramm beschrieben werden. Komplexere Algorithmen, deren graphische Modellierung zu umständlich wäre, und andere Aspekte, die sich im Diagramm nur schlecht ausdrücken lassen, müssen somit in andere Dienste (JIAC-Actions) ausgelagert werden.

Einen weiteren möglichen Nachteil stellt die Ausführungsgeschwindigkeit dar. Da in jedem Ausführungsschritt des Agenten nur eine einzelne Aktivität ausgeführt wird, wird der interpretative Ansatz immer erheblich langsamer sein als der generierte Java-Code der AgentBeans.

Aus diesem Grund eignet sich der Interpreter am besten dafür, das Verhalten auf hoher Abstraktionsebene zu modellieren und komplexere Berechnungen in Dienste auszulagern, die durch den Prozess orchestriert werden. Kernaspekte des Agenten sollten weiterhin per Hand programmiert werden – wofür die generierten AgentBeans eine Grundlage bieten könnten. Auf diese Weise kommen die Nachteile nicht zum Tragen und die Vorteile der verschiedenen Ansätze werden optimal genutzt.

Teil III.

Auswertung und Zusammenfassung

Kapitel 9.

Anwendung und Evaluation

In diesem Kapitel wird beschrieben, wie die Abbildung von Prozessmodellen auf Multiagenten-Systeme und die darauf basierende Implementierung und Methodik angewendet und evaluiert wurden. Hierbei soll der Anfangs aufgestellten Frage nachgegangen werden, inwiefern eine Prozessnotation wie BPMN für die umfassende Modellierung von Agenten und Multiagenten-Systemen geeignet ist, und inwiefern die in dieser Arbeit beschriebene Abbildung, deren verschiedene Umsetzungen und die entsprechende Entwicklungsmethodik in der Praxis anwenden lassen.

Zu diesem Zweck wird zunächst ein komplexeres Beispielszenario formuliert, in dem die verschiedenen Aspekte der Abbildung von Prozessen auf Agenten zum Tragen kommen. Anschließend wird ein Überblick über verschiedene vergangene und aktuelle Projekte, in denen die Abbildung und die Implementierung Anwendung fanden, gegeben. Zuletzt werden die verschiedenen Umsetzungen der Abbildung untereinander und mit dem Stand der Technik verglichen und kritisch betrachtet, für welche Anwendungsgebiete diese jeweils am besten geeignet sind.

9.1. Anwendungsbeispiel

In diesem Abschnitt soll die Abbildung und die Implementierung anhand eines einfachen Beispiel-Diagramms veranschaulicht werden. Weitere Beispiele können in den Vorarbeiten gefunden werden, insbesondere zur Integration mit anderen Werkzeugen zu einer Methodik zur prozessorientierten Entwicklung von Agentensystemen [93] sowie zur Abbildung der Prozessdiagramme auf JIAC-AgentBeans [89].

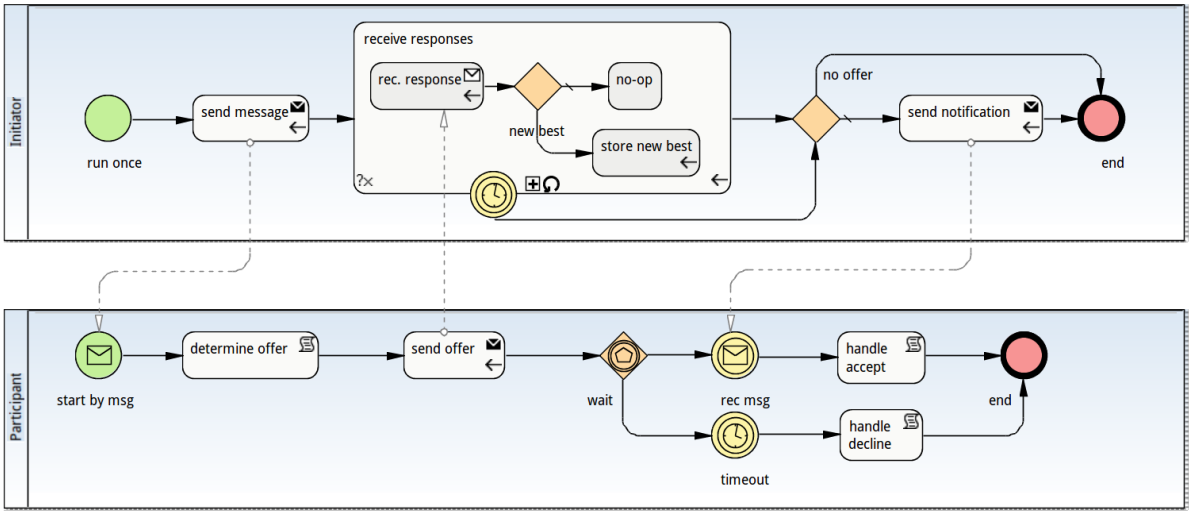


Abbildung 9.1.: BPMN-Diagramm des Auktionsprotokolls, modelliert im VSDT.

9.1.1. Beispiel: Auktionsprotokoll

In diesem Beispiel wird ein abstraktes Auktionsprotokoll betrachtet, in dem ein Initiator mehrere Teilnehmer nach Geboten fragt, sich für das beste Gebot entscheidet und dem entsprechenden Teilnehmer eine Bestätigungsnachricht schickt. Abbildung 9.1 zeigt das entsprechende BPMN-Diagramm wie es im VSDT-Editor modelliert wurde.

Das Beispiel ist vergleichsweise kompakt und aus sich selbst heraus verständlich, und stellt doch einen üblichen und nutzbringenden Anwendungsfall für die Modellierung mit BPMN dar. Zugleich deckt es die wichtigsten Aspekte der BPMN und der Abbildung auf Agenten ab, insbesondere nachrichtenbasierte Kommunikation (Unicast und Multicast), sowie komplexeren Kontrollfluss, einschließlich Event-Handling.

An dem Protokoll sind zwei Rollen beteiligt: Der Initiator und der Teilnehmer, wobei es vom Teilnehmer mehrere Instanzen geben kann. Das Beispiel kann mit jeder der drei Implementierungen umgesetzt werden. In diesem Fall wird der obere Pool, d. h. der Initiator, vom Interpreter ausgeführt, während für den unteren Pool, die Teilnehmer, die Abbildung auf JIAC AgentBeans verwendet wird. Die beiden Implementierungen sind kompatibel und können im Beispiel gemeinsam eingesetzt werden.

- Der Initiator-Pool hat ein None-Start-Event, d. h. der Prozess wird einmalig gestartet sobald der Prozess auf den Interpreter-Agenten deployed wird. Andere Start-Modi wären natürlich ebenso möglich, etwa über ein Message-Start-Event als Service. Der Initiator beginnt die Auktion indem er eine Nachricht an die Nach-

richtengruppe aller Teilnehmer sendet. Im darauf folgenden Subprozess werden die Antworten der Teilnehmer empfangen. Diese werden mit dem bisher besten Angebot verglichen und das bessere wird zusammen mit der dazugehörigen Absenderadresse gespeichert. Der Subprozess wird solange wiederholt, bis er nach einigen Sekunden durch das Timer-Event unterbrochen wird. Schließlich wird dem Agenten mit dem besten Angebot – sofern es eines gibt – geantwortet und der Prozess wird beendet. Die anderen Teilnehmer gehen leer aus.

- Der Teilnehmer-Pool wird mit dem Empfang einer Nachricht an die entsprechende Nachrichtengruppe gestartet. Der Teilnehmer erstellt ein Angebot (im Beispiel: eine Zufallszahl) und schickt diese zurück an den Absender der ersten Nachricht, d. h. zum Initiator. Während der Initiator die verschiedenen Angebote empfängt und vergleicht, wartet der Teilnehmer an dem Event-based Gateway auf die Benachrichtigung über den Zuschlag. Wird diese Nachricht empfangen löst ein Nachrichten-Event-Handler aus und über einen Script-Task wird eine entsprechende Meldung ausgegeben. Wurde nach einigen Sekunden noch keine Nachricht erhalten, so löst der Timer-Event-Handler aus und eine andere Meldung wird ausgegeben. Schließlich endet auch der Teilnehmer-Prozess.

Die vollständigen formalen Modelle für die Prozesse und das daraus abgeleitete Agentensystem sowie der vollständige generierte JIAC-AgentBeans-Code kann im Anhang A dieser Arbeit nachgeschlagen werden.

Während das hier gezeigte Auktionsprotokoll keinen direkten Nutzen erfüllt und nur für den Zweck des Beispiels erstellt wurde, kann es einfach in ein Protokoll für eine echte Auktion erweitert werden. Hierfür muss lediglich die AgentBean des Auktions-Teilnehmers erweitert und die Methoden für das Bestimmen des Angebots (im Beispiel: *determine offer*) und für die Aktivität im Falle, dass die Auktion gewonnen wird (im Beispiel: *handle accept*), überschrieben werden. Dadurch, dass jede Aktivität im generierten Code durch eine eigene Methode dargestellt wird, ist dies ohne Probleme möglich und es bleibt kompatibel mit der Initiator-Seite des Prozesses. Auf diese Weise können die Prozessdiagramme dazu genutzt werden, Templates (Vorlagen) für verschiedene Standard-Protokolle zu modellieren, welche dann mittels Vererbung für verschiedene Szenarien implementiert werden können.

9.2. Anwendung in Projekten

Die Abbildung von Prozessen auf Agenten und die dazugehörigen Modelle, das Prozessmodellierungswerkzeug und die verschiedenen Implementierungen der Abbildung wurden in zahlreichen Projekten angewandt und sukzessive weiterentwickelt: Zunächst in *SerCHo*, später in *Smart Senior* und *ILIas*, und zuletzt im Projekt *EMD*. Hierbei wurde in der Regel nur ein Teil der Abbildung verwendet – insbesondere die Modellierung der einzelnen Agenten-Pläne oder Dienste als Prozesse – während etwa die Struktur der Agenten vorgegeben war.

9.2.1. Anwendung im Projekt „SerCHo“

Die erste Version des Werkzeugs zur Prozessmodellierung und der Abbildung wurde im Rahmen des Projekts *SerCHo* (*Service Centric Home*) entwickelt, um auf diesem Wege Regeln zur Heimautomatisierung zu beschreiben. Der Schwerpunkt lag dementsprechend auf der Abbildung der Prozesse auf Agenten-Pläne oder Services, während die Agentenarchitektur weniger relevant, bzw. vom System vorgegeben war.

Die erste Version der Abbildung [86] erzeugte hierbei JADL-Skripte für das ältere Agenten-Framework JIAC IV [149], das heute nicht mehr unterstützt wird. Aufgrund externer Faktoren kam JIAC IV im Projekt jedoch nicht zum Einsatz, und somit konnte auch die eigentliche Abbildung auf Agenten-Services nicht verwendet werden; stattdessen wurde die (ursprünglich nur als *proof of concept* für das Transformationssystem entwickelte) Abbildung auf BPEL-Prozesse verwendet [88]. Diese wurden dann als Webservices angeboten und aus der *Smart Home Energy Assistant* (SHEA) Web-UI aufgerufen [1].

Während im Projekt *SerCHo* am Ende keine Agenten eingesetzt wurden, konnte in dessen Rahmen somit dennoch eine frühe Version der Abbildung auf Agenten entwickelt und ein Großteil des *VSDT* BPMN-Editors umgesetzt werden.

9.2.2. Anwendung im Projekt „Smart Senior“

Die Abbildung auf *JADL++* (JIAC V-Agenten-Skripte) wurde erstmals im Projekt *Smart Senior* angewandt [134]. Das Ziel dieses Projekts bestand darin, Regeln und Dienste für ein altersgerechtes Wohnen zu entwickeln. Ähnlich wie in *SerCHo* war die Agenten-Architektur vorgegeben: Es gab verschiedene Agenten-Knoten, die jeweils einem

Haushalt entsprachen, und auf jedem lief ein JADL++-Interpreter-Agent; zudem gab es einen zentralen Backend-Knoten, der mit den verschiedenen Haushalten kommunizierte.

Die einzelnen Agenten-Skripte wurden in erster Linie dazu genutzt, verschiedene „Basisdienste“ zu verknüpfen; Agenten-Kommunikation und Interaktion innerhalb der Skripte wurden zwar unterstützt, im Projekt jedoch nicht benötigt. Neben den eigentlichen Diensten wurde außerdem die Abbildung auf Agenten-Reaktionsregeln verwendet, die im Falle von JADL++ über Drools-Regeln realisiert wurden. Auf diese Weise konnte ein Dienst etwa immer dann aufgerufen werden, wenn der Agent von einem anderen Teil des Systems eine bestimmte Nachricht erhielt, oder zu bestimmten Zeiten.

Für die Installation der Dienste in den Haushalten wurde eine Erweiterung für das Prozess-Modellierungswerkzeug entwickelt: Über den sogenannten *Service Deployment View* konnten die Prozessmodelle direkt aus dem VSDT heraus nach JADL++ übersetzt und die resultierenden Agenten-Dienste über einen *Backend Proxy Node* auf die verschiedenen Haushalte deployed werden.

Neben der Entwicklung der Transformation auf JADL++ trugen die Arbeiten im Projekt *Smart Senior* insbesondere zur Abbildung der Start-Events auf verschiedene Regeln zum Starten der Prozesse bei.

9.2.3. Anwendung im Projekt „ILIas“

Das Anwendungsszenario im Projekt *ILIas* bestand darin, Dienste zur Gewährleistung einer reibungslosen Versorgung bei teilweisen Ausfällen der Netz-Infrastruktur bereitzustellen [84].

Die Umsetzung in Hinblick auf das VSDT und die Verwendung der Abbildung gestaltete sich hierbei weitestgehend identisch zu derjenigen in *Smart Senior*: Die Dienste und deren jeweilige Start-Bedingungen wurden im VSDT in Form von Prozessdiagrammen modelliert und über einen speziellen *View* auf die verschiedenen Netzversorgungsknoten deployed. Auch hierbei wurde die Abbildung auf JADL-Services in erster Linie für die Verknüpfung von bestehenden Basisdiensten verwendet; die Agenten-Architektur war durch das laufende System bereits vorgegeben.

Bezogen auf die Abbildung der Prozesse auf Agenten gab es in diesem Projekt keine grundlegenden Neuentwicklungen, jedoch konnten die bestehenden Ansätze und Implementierungen weiter getestet und verfeinert werden.

9.2.4. Anwendung im Projekt „MSG EUREF“

Das Projekt *Micro Smart Grid EUREF* befasste sich mit der Optimierung der Ladesteuerung für elektrische Fahrzeuge in einem Micro Grid (MSG) am EUREF-Campus in Berlin. Hierbei kam eine aus verschiedenen Komponenten bestehende und auf mehrere Agenten verteilte Optimierung zum Einsatz, bei der einzelne Agenten verschiedene Aufgaben übernahmen, etwa die Anreicherung der Daten mittels Lernverfahren, die Zuordnung der Fahrzeuge zu Ladestationen, die eigentliche Ladeplan-Optimierung sowie die letztendliche Umsetzung der Pläne [69].

Das Agentensystem und die darin implementierten Optimierungsalgorithmen wurden zunächst direkt in JIAC umgesetzt. Die Optimierung wurde dabei in mehrere „Phasen“ unterteilt (Lernen, Prognose, Zuordnung, Planung, u.a., teils in mehreren alternativen Ausführungen) die jeweils in Form von Actions repräsentiert wurden und frei miteinander kombiniert werden konnten.

Mit dem Ziel, einen besseren Überblick über die Abläufe zwischen den Agenten zu erhalten, wurden Teile der Interaktion später in Form von BPMN-Prozessen nachmodelliert und mithilfe des JIAC-basierten Prozess-Interpreter-Agenten ausgeführt [70]. Die erfolgreiche Durchführung dieser Umstellung wurde im Rahmen der Abschlussarbeit von M. Voß dokumentiert [164, Kapitel 6], wobei insbesondere der Nutzen für die Prozessdokumentation und für die Kommunikation zwischen den Projektpartnern positiv eingeschätzt wurde.

9.2.5. Anwendung im Projekt „EMD“

Im Projekt *EMD* (*Erweiterte und Adaptive Elektromobilitätsdienste*) wurden komplexe Prozesse zur Steuerung von Fahrzeug-Buchungs und -Ladeprozessen modelliert und in einem *micro grid* eingesetzt. Die Prozesse setzten sich hierbei aus verschiedenen, semantisch beschriebenen Basisdiensten zusammen, etwa für die Steuerung von Ladestationen, für die Fahrzeugbuchung oder die Abrechnung [49, 48].

Die semantischen Dienste wurden im *Semantic Service Manager (SSM)* [101] beschrieben (oder bestehende Dienste durchsucht), aus demselben in das VSDT importiert, dort zu komplexen Prozesse kombiniert und schließlich in Verhaltensweisen in Form von JIAC-Actions übersetzt. Für die Ausführung kamen hierbei zunächst die Transformation nach JIAC AgentBeans und später der JIAC-basierte Prozessinterpreter zum Einsatz.

Wie in den anderen Projekten so war auch hier die Agentenarchitektur vorgegeben, und auch nachrichtenbasierte Kommunikation spielte nur eine untergeordnete Rolle, wo-

bei komplexere Dienste prinzipiell auch aus mehreren kommunizierenden Agenten bestehen konnten. Neben der Orchestrierung der (semantischen) Basisdienste fand auch das Event-Handling Anwendung. Ferner wurden die erzeugten Dienste mit dem *Semantic Service Matcher (SeMa²)* verbunden [50], um unvollständig bzw. nur als Template angegebene und aus dem SSM importierte Dienste zur Laufzeit auf verfügbare Basisdienste zu matchen.

Eine weitere Neuerung des Projekts EMD sind die in Abschnitt 8.3.3 beschriebenen Nutzerschnittstellen, über die der Interpreter überwacht werden und dieser Nachrichten und Nutzeranfragen (in Form von *User Tasks*) in die verwendete *SPA*-UI [17] schicken kann. Auf diese Weise können insbesondere auch fehlende Dienst-Parameter abgefragt werden, im Falle dass ein durch den SeMa² gefundener Dienst andere oder zusätzliche Parameter erwartet, als im Template vorgesehen waren.

9.2.6. Anwendung im Projekt „AcRoSS“

Im aktuell laufenden Projekt *AcRoSS* werden die Ansätze aus EMD weitergeführt und auf die Augmented-Reality-Domäne (AR) angewandt [102]. Wie schon in EMD liegt der Fokus auf der Beschreibung komplexer Dienstkompositionen, bestehend aus semantisch beschriebenen Basisdiensten. Für die Übertragung auf Agenten und die Ausführung der Prozesse wird der in Abschnitt 8.3.3 beschriebene BPMN-Interpreter-Agent verwendet.

Ein besonderer Schwerpunkt liegt hierbei zum einen in der Nutzerinteraktion, und zum anderen in der Zuordnung der Prozesse zu verschiedenen Rollen. Für Nutzerinteraktionen, beispielsweise in Form von Einblendungen im Sichtfeld der AR-Geräte, werden komplementär zu den in EMD verwendeten *User Tasks* über spezielle *UI Services* realisiert, die wie gewöhnliche Dienste im Prozess orchestriert werden können und mehr Flexibilität in der Interaktion mit dem Nutzer erlauben. Die Zuordnung der Prozesse zu unterschiedlichen Rollen, etwa für Kunden oder Wartungstechniker, geschieht über die Participants im BPMN-Prozess.

Schließlich wird in AcRoSS auch die automatische Planung anhand semantischer Dienstbeschreibungen und deren Anwendung sowohl bei der Entwicklung als auch zur Laufzeit erforscht, und wie diese in die prozessorientierte Entwicklung der Agentensysteme integriert werden kann.

9.2.7. Fazit

In allen Projekten lag der Schwerpunkt auf der Modellierung von „Mehrwertdiensten“, basierend auf einer Auswahl von „Basisdiensten“, die in ein bestehendes Multiagenten-System installiert und von diesem ausgeführt werden sollen. Dementsprechend wurde in den Projekten in erster Linie die Abbildung auf Agenten-Pläne und ggf. Reaktionsregeln zum Starten dieser Pläne genutzt. Die Abbildung der Participants auf Agentenrollen und damit verwandte Aspekte der Agenten-Architektur wurde nur sehr begrenzt eingesetzt, etwa für die Zuordnung von Prozessen zu bestimmten Rollen in AcRoSS.

Tabelle 9.1 zeigt einen Überblick darüber, welche Aspekte der Abbildung in den verschiedenen Projekten verwendet wurden.

Tabelle 9.1.: Vergleich der Anwendung in den verschiedenen Projekten

	Aspekt	**SerCHo**	**Smart Senior**	**ILIas**	**MSG EUREF**	**EMD**	**AcRoSS**
Abbildung	BPEL	x	-	-	-	-	-
	JIAC IV JADL	x	-	-	-	-	-
	JIAC V JADL++	-	x	x	-	-	-
	JIAC V AgentBeans	-	-	-	-	x	-
	JIAC V Interpreter	-	-	-	x	x	x
im Proj. verwendet	Agenten-Pläne/Dienste	(x)	x	x	x	x	x
	Dienstverknüpfungen	x	x	x	x	x	x
	Kommunikation	-	(x)	(x)	x	(x)	x
	Regelbasiertes Verhalten	-	x	x	x	x	(x)
	Agenten-Architektur	-	-	-	x	(x)	(x)
	Dienst-Semantik	-	-	-	-	x	x
	Dynamik, Planung	-	-	-	-	(x)	x
	Nutzerinteraktion	-	-	-	-	x	x

Für das Service-Engineering hat sich der Ansatz hierbei gut bewährt. Aufgrund der jeweiligen Rahmenbedingungen der Projekte konnten die eigentlichen Stärken der Prozessmodellierung – etwa die Übersichtliche Darstellung der Kommunikation in komplexen Multiagenten-Systemen – jedoch nicht oder nur sehr begrenzt ausgespielt werden. Hierfür wird stattdessen auf das komplexere Beispiel in Abschnitt 9.1 verwiesen.

9.3. Diskussion

In diesem Abschnitt sollen die verschiedenen im vorigen Kapitel vorgestellten Umsetzungen der Abbildung sowohl untereinander als auch mit dem entsprechenden Stand der Technik verglichen, und die individuellen Stärken und Schwächen der Implementierungen und die Einsatzgebiete, für die diese jeweils am geeignetsten erscheinen, identifiziert werden. Abschließend wird versucht, die Frage zu beantworten, in wie weit die in dieser Arbeit vorgestellte Abbildung von Geschäftsprozessmodellen auf Multiagenten-Systemen dazu geeignet ist, die komplexen Verhaltensweisen und Interaktionen von Agenten zu beschreiben.

9.3.1. Vergleich der verschiedenen Implementierungen

Jede der drei Implementierungen entspricht der beschriebenen Abbildung, doch jede hat ihre eigenen Stärken und Schwächen und eignet sich für unterschiedliche Anwendungsgebiete. (Siehe Abschnitte 8.3.1, 8.3.2 und 8.3.3 für ein ausführliches Fazit der jeweiligen Implementierung).

- Die Transformation in JADL-Skripte erzeugt sehr kompakten, gut lesbaren Code, jedoch ist die Ausdruckskraft durch den geringen Sprachumfang eingeschränkt. Die Skripte ebenso wie die dazugehörigen Starter-Regeln können zur Laufzeit deployed und undeployed werden, wodurch sich diese Implementierung insbesondere für austauschbare, abstrakte Dienste und Verhaltensweisen geeignet ist.

- Die Transformation in JIAC-AgentBeans ist am mächtigsten, da nicht nur jedes BPMN-Element durch ein entsprechendes Java-Konstrukt nachgebildet, sondern der generierte Code auch beliebig erweitert werden kann. Indem verschiedene Hilfsfunktionen über eine Super-Klasse bereitgestellt werden, ist der Code immer noch lesbar, wenn auch komplexer als der entsprechende JADL-Code. AgentBeans können nicht zur Laufzeit ausgetauscht werden, wodurch diese Implementierung am besten für die Umsetzung von Basiskomponenten geeignet ist.

- Der Interpreter unterscheidet sich grundsätzlich von den anderen Ansätzen, da die Prozesse nicht in eine entsprechende Blockstruktur übertragen werden müssen, sondern beliebig strukturiert sein können, wodurch jedoch auch strukturelle Probleme wie Deadlocks möglich sind. Die Ausdrucksfähigkeit ist ähnlich hoch wie bei den AgentBeans. Zugleich müssen die Prozessdiagramme selbst alles enthalten, was für die Ausführung notwendig ist, da es keinen generierten Code gibt, den

man entsprechend erweitern könnte. Ebenso wie JADL-Skripte können die Prozesse zur Laufzeit deployed werden und eignen sich somit ebenfalls in erster Linie für „high-level“ Dienste und Verhaltensweisen, insbesondere für die Orchestrierung bestehender Basisdienste.

Ein wichtiger Unterschied ergibt sich zudem aus der Integration in den Ausführungszyklus des Agenten: Während bei der Transformation in AgentBeans der gesamte Workflow in einer Methode gekapselt wird, die der Agent am Stück ausführt, werden in beiden interpretativen Ansätzen – sowohl bei der Transformation nach JADL als auch beim BPMN-Interpreter – in jedem Intervall des Ausführungszyklus nur ein einzelnes Statement[1] bzw. ein einzelner Task des Prozesses ausgeführt. Diese beiden Ansätze haben daher eine erheblich niedrigere und vom Ausführungsintervall des Agenten bestimmte Ausführungsgeschwindigkeit, eignen sich zugleich aber auch besser für länger laufende Prozesse, ohne den Agenten zu blockieren.

Eine Übersicht der individuellen Unterschiede, Stärken und Schwächen der verschiedenen Umsetzungen kann Tabelle 9.2 entnommen werden.

Tabelle 9.2.: Vergleich der Eigenschaften der verschiedenen Umsetzungen

Eigenschaft	**JADL**	**AgentBeans**	**Interpreter**
Sprache des Pläne	JADL++	Java	(BPMN)
Sprache der Regeln	Drools	Java	(BPMN)
Prozessstruktur	Block	Block	beliebig
Ausdruckskraft / Umfang	gering	sehr hoch	hoch
Erweiterbarkeit des Codes	gut	sehr gut	schlecht
Änderbar zur Laufzeit	ja	nein	ja
Ausführungsgeschwindigkeit	gering	hoch	gering

Auch gibt es geringfügige Unterschiede bezüglich der Abdeckung der Abbildung, bezogen auf die Elemente des verwendeten BPMN-Metamodells, bedingt durch Einschränkungen der jeweiligen Zielsprachen. Tabelle 9.3 zeigt einen Vergleich der von den verschiedenen Implementierungen unterstützten Elemente, einschließlich der offiziellen Abbildung von BPMN nach BPEL [118, Kapitel 14] als Referenz.

Dadurch, dass jede der drei Umsetzungen dieselbe Kommunikations-Infrastruktur verwendet und auf der gleichen Abbildung von Prozessen auf Multiagenten-Systeme basiert

[1] In JADL besteht auch die Möglichkeit, einzelne Skript-Abschnitte als `protected` zu kennzeichnen; diese werden dann am Stück ausgeführt.

Tabelle 9.3.: Vergleich der verschiedenen Abbildungen von BPMN nach *X* [92]

	Element	**BPEL** (Referenz)	**JADL**	**Agent Beans**	**Inter-preter**
Kontrollfluss	XOR, AND, OR Gateway	x	x	x	x
	Event-bsd. XOR Gateway	x	x	x	x
	Complex Gateway	-	-	-	-
	Event Handler, Error	x	x	x	x
	Event Handler, Other	(x)	-	x	x
Aktivitäten	Send, Receive Task	(x)	x	x	x
	Service Task	x	x	x	x
	User Task	(x)	-	(x)	x
	Manual Task	-	-	-	-
	Script Task	-	x	x	(x)
	Subprocess	(x)	(x)	x	x
	Transaction	-	-	-	-
	Call Activity	(x)	(x)	(x)	(x)
	Ad-Hoc-Subprocess	-	-	(x)	(x)
Ereignisse	Message	(x)	x	x	x
	Timer	x	x	x	x
	Rule	-	(x)	x	(x)
	Signal	-	-	-	(x)
	Escalate	-	-	-	-
	Error	x	-	x	x
	Compensate	x	-	-	-
	Cancel	-	-	-	-
	Terminate	x	-	x	x
Anderes	Properties, Assignments	x	x	x	x
	Multiple Lanes	-	-	-	-
	Data Objects	-	-	-	-
	Roles	-	x	x	x
	Service Starter	(x)	x	x	x

– auch wenn diese nicht jedes mal vollständig abgedeckt wird – sind Verhaltensweisen, die mit den verschiedenen Implementierungen realisiert wurden, untereinander kompatibel und können miteinander interagieren. So ist es etwa möglich, die Grund-Komponenten eines Agenten mit der Transformation nach JIAC AgentBeans zu erstellen, und diese zur Laufzeit um generierte JADL-Skripte zu erweitern, die mit einem zweiten Agenten interagieren, dessen Verhalten vom Prozess-Interpreter bestimmt wird.

Auf diese Weise kann ein „heterogenes“ System erstellt werden, in dem die verschiedenen Implementierungsarten optimal eingesetzt werden können.

9.3.2. Ausblick und weiterführende Fragestellungen

Die in dieser Arbeit beschriebene Abbildung von Prozessmodellen auf Agenten kann bereits sinnvoll für die Beschreibung direkt ausführbarer Multiagenten-Systeme angewendet werden, und wurde zu diesem Zweck in verschiedenen Forschungsprojekten eingesetzt. Dennoch gibt es Aspekte, in denen die Abbildung, sowie die sie umgebende Methodik und deren Implementierungen noch verbessert und ergänzt werden können:

- Die Abbildung deckt noch nicht sämtliche Elemente der BPMN ab; insbesondere die Aufteilung der Prozesse in Lanes sowie die Unterstützung von Transaktionen und Data Objects könnten wertvolle Erweiterungen darstellen.
- Die Abbildung von Ad-hoc Subprozessen auf Ziele und Planung konnte in den Implementierungen noch nicht vollständig umgesetzt werden und ist Thema der aktuellen Forschungsprojekte des Autors.
- Das Prozessmodell und die Abbildung könnten um weitere, agentenspezifische Artefakte erweitert werden. Diese Erweiterungen sollten jedoch optional sein und die Semantik der übrigen Elemente nicht verfremden.
- Während die Abbildung auf drei verschiedene Arten umgesetzt wurde, wird bislang in jedem Fall das JIAC-Framework verwendet. Eine Implementierung für ein weiteres Agenten-Framework könnte die Übertragbarkeit des Ansatzes demonstrieren.
- Die graphische Modellierung der Verhaltensweisen als Prozesse kann mitunter mühsam sein. Damit neben der übersichtlicheren Darstellung auch eine echte Zeitersparnis gegenüber der Beschreibung im Programmcode entsteht, müsste die Bedienung der Modellierungswerkzeuge vereinfacht und weiter automatisiert werden.

Diese Erweiterungen würden helfen, den in dieser Arbeit vorgestellten Ansatz weiter abzurunden und in der Praxis effektiver einsetzen zu können.

9.3.3. Vergleich mit dem Stand der Forschung

An dieser Stelle soll die Abbildung und deren Umsetzungen mit verwandten Ansätzen verglichen werden. Neben der Abbildung von BPMN auf BPEL, die als „Referenzimplementierung“ für jede automatische Abbildung von BPMN-Diagrammen auf ausführbare Programme angesehen werden kann, werden insbesondere drei verwandte Arbeiten betrachtet: WADE, GO-BPMN und AUML.

Da diese im Kapitel 4 bereits beschrieben wurden, sollen an dieser Stelle nur die jeweiligen Unterschiede untereinander sowie Vor- und Nachteile verglichen mit der in dieser Arbeit beschriebenen Abbildung von BPMN auf Agenten betrachtet werden.

Abbildung auf BPEL

Eine Motivation hinter der Entwicklung von BPMN war es, über eine graphische Notation für BPEL-Prozesse zu verfügen. Die daraus entstandene offizielle Abbildung von BPMN auf BPEL [118, Kapitel 14] stellt zugleich eine Ausführungssemantik für viele Elemente der BPMN und einen Maßstab für jede andere Abbildung von BPMN auf ausführbaren Programmcode dar.

Um so überraschender ist es, dass diese Abbildung auf BPEL nur einen kleinen Teil der BPMN abdeckt. Neben den üblichen Einschränkungen bei der Abbildung der Prozessstruktur auf blockstrukturierte BPEL-Prozesse, gibt es auch zahlreiche BPMN-Elemente, die nicht oder nur teilweise abgebildet werden können. Am auffälligsten ist dies bei Nachrichten, die in BPEL ausschließlich auf Webservice-Interaktionen abgebildet werden. Nachrichten an Gruppen von Empfängern werden nicht unterstützt, wodurch zahlreiche Interaktionsprotokolle nicht umsetzbar sind. Die Unterstützung für Start-Events beschränkt sich ebenfalls auf Message-Start-Events zum Starten der beschriebenen Webservices.

Verglichen mit der Abbildung auf BPEL erscheint die hier beschriebene Abbildung auf Agenten vielfältiger und durch die Verwendung von Rollen und Start-Regeln für die unterschiedlichen Start-Events auch näher an der eigentlichen Intention von BPMN: Der Modellierung von Prozessen zwischen verschiedenen Rollen, nicht zwischen Services.

WADE: Workflows for JADE

Die JADE-Erweiterung *WADE (Workflows and Agents Development Environment)* [27] ist konzeptionell ähnlich zur hier beschriebenen Abbildung und insbesondere der Implementierung zur Erzeugung von JIAC-AgentBeans. Insbesondere wurde die in dieser Abbildung verwendete Aufteilung in Workflow- und Activity-Methoden von WADE inspiriert. Zugleich gibt es jedoch auch einige wichtige Unterschiede.

Am auffälligsten ist, dass die Prozesse bei WADE nicht zu äquivalenten Block-Strukturen umgeformt, sondern direkt über eine spezielle Datenstruktur abgebildet werden, in der die verschiedenen Nachfolger-Beziehungen des Prozessgraphen kodiert sind. Der Vorteil dieser Technik ist, dass die Prozesse keiner Blockstruktur folgen müssen; der Nachteil ist jedoch, dass der auf diese Weise generierte Quellcode weniger intuitiv verständlich ist und nicht so einfach erweitert oder verändert werden kann.

Ein weiterer Unterschied betrifft die verwendete Prozessnotation. In WADE wird nicht BPMN, sondern eine eigene Notation verwendet. Gerade zu Beginn war diese Notation sehr einfach und erlaubte es lediglich, Aktivitäten mit (bedingten) Flussrelationen zu verknüpfen. Insbesondere gab es weder Event-Handling noch Interaktionen. In späteren Versionen wurde WADE in diese Richtung erweitert und bietet inzwischen auch Event-Handling, Kommunikation und Web-Service-Integration und wird in zahlreichen Projekten eingesetzt [6, 7].

GO-BPMN und Go4Flex

GO-BPMN (Goal-oriented BPMN) verwendet ebenfalls BPMN um das Verhalten von Agenten zu beschreiben [21]. Ein ähnlicher Ansatz findet sich auch bei *Go4Flex* wieder, das BPMN mit Ziel-Hierarchien für Jadex-Agenten verbindet [15].

Der Unterschied zu dem vorliegenden Ansatz besteht darin, dass die BPMN-Diagramme in GO-BPMN dazu verwendet werden, isolierte Pläne zu beschreiben. Hierbei wird nur ein kleiner Teil der BPMN verwendet; insbesondere macht der Ansatz keinen Gebrauch von Nachrichten, um etwa Interaktionen zwischen verschiedenen Agenten darzustellen. Und auch die verschiedenen Start-Events werden nicht genutzt; stattdessen werden die Prozesse unterschiedlichen Zielen zugeordnet und über (sprachfremde) Zielhierarchien miteinander in Beziehung gesetzt und ausgeführt.

Während die Kombination von BPMN-Prozessen mit Zielen vielversprechend ist, wird so ein Großteil der Ausdruckskraft und der eigentlichen Stärken von BPMN nicht genutzt.

In dieser Arbeit wird BPMN auf höherer Abstraktionsebene eingesetzt: Die Prozessdiagramme beschreiben über ihre Start-Events selbst, in welchen Situationen sie ausgeführt werden sollen. Ziele und deren Zuordnung zu konkreten Plänen wiederum findet auf niedrigerer Ebene statt, innerhalb von Ad-hoc Subprozessen. Auf diese Art ist das Verhalten der Agenten näher an der ursprünglichen Bedeutung der BPMN-Prozesse.

AUML-Interaktionsdiagramme

Verglichen mit *AUML* Interaction Diagrams [4] haben die BPMN-Prozesse deutlich mehr Ausdruckskraft. Während in AUML, genau wie in UML, jeder Aspekt des Gesamtsystems in einem eigenen Diagramm-Typus dargestellt wird, vereinen die BPMN-Prozessdiagramme die Aspekte „Interaktionen“ und „Aktivitäten“ in einem Diagramm, und erlauben somit, sowohl das interne Verhalten der Agenten als auch deren Kommunikation mit anderen Agenten gemeinsam zu modellieren.

Dies hat Vor- und Nachteile. Einerseits vermittelt es besser die Zusammenhänge: Nicht nur *welche* Nachricht ein Agent schickt, sondern auch *wieso*, beispielsweise welche Prüfung zuvor vorgenommen wurde, oder welche Aktionen folgen. Andererseits kann genau diese Trennung aber auch erwünscht sein, etwa wenn das Interaktionsprotokoll als „Interface“ dienen soll, das auf verschiedene Arten implementiert werden kann. Letzteres ist im Falle der Transformation von BPMN auf JIAC-V-AgentBeans und in analogen Ansätzen, wie etwa in WADE, allerdings auch möglich, mittels Vererbung der generierten Klassen.

9.3.4. Fazit

Geschäftsprozessmodellierung, insbesondere mit BPMN, wird hauptsächlich auf zwei Arten eingesetzt: Einerseits lässt sich mit der BPMN ein umfassender Überblick über ein komplexes System darstellen, einschließlich aller beteiligter Akteure und deren Interaktionen untereinander ebenso wie die jeweiligen internen Prozesse. Andererseits eignet sich die Notation auch dazu, einfache Dienstverknüpfungen, oder „Orchestrierungen“, auf hoher Abstraktionsebene zu beschreiben.

Die in dieser Arbeit vorgestellte Abbildung von BPMN auf Agenten eignet sich in ihren verschiedenen Implementierungen für beide Einsatzgebiete: Mit der Abbildung auf JIAC-AgentBeans lässt sich das Grundgerüst für ein komplexes Multiagenten-System generieren, das sich effizient und robust ausführen lässt und beliebig erweitert werden kann. Die Abbildung auf JADL-Dienste sowie die direkte Interpretation der Prozessdiagram-

me durch einen Prozess-Interpreter-Agenten wiederum eignen sich durch die Möglichkeit des Deployments zur Laufzeit besser für *high-level* Service-Orchestrierungen.

Dadurch, dass alle drei Implementierungen dieselbe Abbildung von Prozessen auf Agenten umsetzen, und auf derselben Kommunikationsinfrastruktur aufsetzen, ist es zudem möglich, die Abbildungen auf JADL und auf JIAC-AgentBeans und den Interpreter gemeinsam einzusetzen, und einen Teil des Systems auf die eine, und einen anderen auf die andere Weise auszuführen. So kann man etwa den Systemkern mit AgentBeans und verschiedene Erweiterungen als JADL-Services oder durch direkte Prozess-Interpretation umsetzen.

Verglichen mit den betrachteten verwandten Arbeiten ergeben sich sowohl Vor- als auch Nachteile. Gegen die Verwendung reiner BPMN, bzw. für die Verwendung einer eigenen Notation, spricht, dass die Überdeckung von BPMN- und Agenten-Konzepten nicht perfekt ist: In beiden Domänen gibt es Aspekte, die sich nicht oder nicht optimal auf die andere abbilden lassen. Dies trifft insbesondere auf Ziele und Planung zu, die sich in BPMN zwar in Form von Ad-Hoc Subprozessen darstellen lassen, in GO-BPMN hingegen über spezielle Ziel-Hierarchien modelliert werden können. Für die Verwendung von BPMN, bzw. allgemein für eine standardisierte Prozessnotation, spricht hingegen, dass die Prozesse, und damit die entsprechenden Verhaltensweisen der Agenten, von einer breiten Gruppe von Prozessentwicklern sowie mit marktüblichen Werkzeugen und Methoden erstellt und verstanden werden können.

In diesem Kapitel wurde gezeigt, dass sich die in dieser Arbeit beschrieben Abbildung von BPMN-Prozessen auf Agenten für die Entwicklung komplexer Multiagenten-Systeme eignet und wie diese in verschiedenen Projekten erfolgreich eingesetzt werden konnte. Ferner wurde gezeigt, wie je nach Anwendungsbereich unterschiedliche Implementierungen, etwa als Interpreter oder mittels Code-Erzeugung, den größten Nutzen bringen.

Kapitel 10.

Zusammenfassung

In diesem abschließenden Kapitel soll die Arbeit zusammengefasst und ein Fazit gezogen werden. Es wird betrachtet, inwiefern die in dieser Arbeit beschriebene Abbildungsvorschrift und die dazugehörigen Methoden und Werkzeuge dazu beitragen, Multiagenten-Systeme auf einfache Art und Weise in Form von Geschäftsprozessdiagrammen zu beschreiben, und wie sich die so beschriebenen Agenten für die Ausführung der Prozesse einsetzen lassen.

10.1. Zusammenfassung der Arbeit

Im Bereich verteilter Systeme haben sich in den letzten Jahren unterschiedliche Techniken etabliert: Für die Verknüpfung verteilter Systeme werden häufig serviceorientierte Architekturen verwendet, für deren Modellierung zunehmend Prozessnotationen wie die BPMN genutzt werden. Gerade im akademischen Bereich wird darüber hinaus das Paradigma der agentenorientierten Softwareentwicklung erforscht, welches neben Verteiltheit und Kommunikation auch Konzepte wie Ziele und Planung einbringt. Eine Schwäche der Multiagenten-Systeme liegt jedoch in der fehlenden Standardisierung und dem Mangel an verbreiteten und einheitlichen Notationen. In dieser Arbeit sollte daher untersucht werden, inwiefern Geschäftsprozessdiagramme, insbesondere solche der BPMN, für die Modellierung von Multiagenten-Systemen eingesetzt werden können.

Hierfür wurde in den Kapiteln 2 und 3 zunächst ein Überblick über die Hintergründe der agentenorientierten Softwareentwicklung und der Prozess-Modellierung gegeben. Dabei wurden verschiedene Definitionen und Eigenschaften von Agenten betrachtet und unterschiedliche Agenten-Architekturen, -Sprachen und -Frameworks beschrieben. Ebenso wurden verschiedene Prozessnotationen verglichen und untersucht, wie Prozessmodelle in der Softwareentwicklung eingesetzt werden können, wobei insbesondere der Aspekt der Prozessstruktur betrachtet wurde. Es gibt hierbei zahlreiche konzeptionelle Ähnlich-

keiten, sodass Geschäftsprozesse gut dazu geeignet erschienen, verteilte Multiagenten-Systeme zu modellieren.

Dementsprechend gibt es bereits einige Ansätze, die auf verschiedene Weisen Agenten und Prozesse verknüpfen. Einige dieser Arbeiten wurden in Kapitel 4 betrachtet. Diese Ansätze lassen sich hierbei grob in verschiedene Kategorien unterteilen: Schnittstellen zwischen Agenten und „klassischen“ Webservices, agentenbasierte Prozess-Management-Systeme, Abbildungen zwischen Agenten und Prozessen, sowie verschiedene Methodologien mit Anleihen an Prozessen und entsprechenden Notationen. Hier sind insbesondere *Go4Flex* und *WADE* zu erwähnen, die aufgrund der verwendeten Notationen und Techniken die größte Ähnlichkeit mit dieser Arbeit aufweisen. Oft werden jedoch vereinfachte Prozessnotationen eingesetzt, oder Prozesse werden nur für die Beschreibung des Verhaltens einzelner Agenten genutzt, und können ihr volles Potential damit nicht entfalten.

Basierend auf diesen Vorarbeiten wurde in Kapitel 5 das Ziel aufgestellt, ein Konzept zu entwickeln, wie Prozessnotationen wie die BPMN am effektivsten und am direktesten für die Modellierung von Multiagenten-Systemen eingesetzt werden können, um auf diese Weise die Entwicklung komplexer und verteilter Systeme zu vereinfachen. Hierfür sollte eine Abbildung von Prozess-Modellen auf Agentensysteme erarbeitet werden.

Zu diesem Zweck wurde im Kapitel 6 jeweils ein abstraktes Metamodell für Multiagenten-Systeme und Prozessmodelle erstellt. Der Schwerpunkt lag dabei auf all jenen Elementen, die für die Abbildung zwischen Prozessen und Agenten besonders relevant sind; andere Konzepte, die sich in manchen Agenten-Frameworks finden, jedoch nicht sinnvoll mit Prozessen darstellen lassen, wurden nicht betrachtet. Dementsprechend lag ein besonderer Fokus auf Aspekten, die sowohl in Prozessen als auch in Agenten eine wichtige Rolle spielen, wie nachrichtenbasierte Kommunikation und Ereignisse, aber auch Regeln und Ziele.

Im Kapitel 7 wurde die Abbildung zunächst konzipiert und danach, basierend auf den zuvor definierten Metamodellen, formal beschrieben. Neben der Abbildung auf die grobe Architektur, wie Agenten, Rollen, Regeln und Pläne, wurden hierbei auch verschiedene Abbildungsregeln für die einzelnen Prozess-Elemente sowie für zahlreiche Prozess-Strukturen, und wie diese auf entsprechende Agenten-Skripte übertragen werden können, aufgestellt.

Anschließend, in Kapitel 8, wurde beschrieben, wie die Abbildung von Prozessen auf Agenten am besten in der agentenorientierten Softwareentwicklung und in Kombination mit anderen Techniken und Werkzeugen eingesetzt werden kann. Außerdem wurde eine prototypische Implementierung eines entsprechenden Entwicklungswerkzeugs vorgestellt,

das *Visual Service Design Tool (VSDT)*, in dem die Abbildung auf verschiedene Arten umgesetzt wird: Als Transformation in *JADL++*-Dienste und *JIAC V AgentBeans* sowie als Prozess-Interpreter-Agent.

Zuletzt, in Kapitel 9, wurde die Abbildung und die damit verbundene Methodik in Hinblick auf die Frage evaluiert, inwiefern diese für die einfache prozessorientierte Modellierung von Multiagenten-Systemen geeignet sind, und wie diese in der Praxis eingesetzt werden können. Hierfür wurden zum einen Erfahrungen aus Projekten herangezogen, und zum anderen – da in diesen oft nur ein Teil der Abbildung verwendet wurde – eine komplexere Beispielanwendung beschrieben. Ferner wurden die verschiedenen, im vorigen Abschnitt vorgestellten Umsetzungen der Abbildung untereinander sowie mit verwandten Arbeiten verglichen und die jeweiligen Stärken und Schwächen herausgestellt.

10.2. Ergebnisse und Diskussion

Die verschiedenen Vor- und Nachteile, Stärken und Schwächen der Abbildung von Prozessmodellen auf Multiagenten-Systeme und der verschiedenen Umsetzungen wurde bereits am Ende des vorigen Kapitels besprochen und sollen an dieser Stelle nicht in aller Ausführlichkeit wiederholt werden. Stattdessen soll ein Blick zurück auf die konkreten Fragestellungen aus Kapitel 1 geworfen und diese beantwortet werden:

- Sind Prozessmodelle dazu geeignet, Multiagenten-Systeme zu beschreiben?
- Sind Multiagenten-Systeme dazu geeignet, Prozessmodelle auszuführen?
- Ist die in dieser Arbeit vorgestellte Abbildung von Prozessen auf Agenten eine geeignete Umsetzung für diese Modellierung von Agentensystemen und Ausführung von Prozessmodellen?

Beschreibung von Multiagenten-Systemen mit Prozessen Um zu gewährleisten, dass sich Agenten mit Prozessnotationen tatsächlich einfacher modellieren lassen, müssen diese Notationen genau so verwendet werden, wie es vorgesehen wurde, und wie Domänenexperten es gewohnt sind. Es sollten somit weder Elemente zweckentfremdet noch übermäßig viele neue Elemente hinzugefügt, und stattdessen nur diejenigen Aspekte modelliert werden, die in beiden Welten – Agenten und Prozesse – vorkommen.

Aufgrund der vielen gemeinsamen Konzepte und Abstraktionen ist diese Schnittmenge jedoch ausreichend groß, sodass die Beschreibung des Verhaltens von Agenten mit Geschäftsprozessdiagrammen, insbesondere solchen der BPMN, sehr gut gelingt.

Hierbei ist natürlich zu beachten, für welche Zwecke die Diagramme ausgelegt sind: Während sich die Aktionen und Interaktionen der Agenten gut darstellen lassen, müssen für andere Aspekte, wie etwa das Domänenmodell, andere Notationen gewählt werden. Auch ist es durch die Heterogenität der verschiedenen Agenten-Frameworks nicht praktikabel, sämtliche Aspekte aller Agentenmodelle zu berücksichtigen.

Ausführung von Prozessen mit Multiagenten-Systemen Die Ausführung der Prozessdiagramme durch Agenten stellt keine Probleme dar. Viele Agenten-Frameworks haben ein Konzept von Diensten – ob dies nun so explizit ist, wie in *JIAC V*, oder in Form von Plänen und Aktionen – und beherrschen i. d. R. alles, was beispielsweise BPEL auch kann. Darüber hinaus verfügen Agenten über Konzepte, die eine „herkömmliche“ Prozess-Ausführungssprache wie BPEL nicht bietet, etwa Unicast- und Multicast-Nachrichten oder komplexes reaktives Verhalten, und selbst deliberatives Verhalten kann in den Prozessen genutzt werden, etwa in Form der Ad-hoc-Subprozesse. Somit können in der Abbildung von BPMN auf Agenten mehr Konzepte berücksichtigt werden, als in der Abbildung von BPMN auf BPEL.

Im Falle, dass die Prozesse existierende (Web-)Services orchestrieren oder mit bestehenden Prozessen integriert werden sollen, müssen die Agenten über geeignete Schnittstellen verfügen. Entsprechende Bibliotheken und Erweiterungen sind für viele gängigen Agenten-Frameworks verfügbar.

Umsetzung in Form einer Abbildung von Prozessen auf Agenten Die in dieser Arbeit vorgestellte Abbildung erfüllt die Anforderung, Agenten mit etablierten Prozessdiagrammen modellierbar zu machen und sie für deren Ausführung einsetzen zu können. Durch die verwendete Notation lassen sich insbesondere Kommunikation und komplexes Event-Handling und Reaktionsregeln deutlich übersichtlicher in einem einzelnen Diagramm darstellen, als dies im Quellcode möglich wäre.

Ein möglicher Kritikpunkt ist, dass nicht die gesamte BPMN-Spezifikation unterstützt wird, und dass auch seitens der Agenten einige Konzepte nicht abgedeckt werden, was sich daraus begründet, dass nur diejenigen Elemente für die Abbildung verwendet werden, die sich in ähnlicher Form sowohl bei den Prozessen als auch bei den Agenten finden lassen. Die Alternative hierzu hätte darin bestanden, entweder den Zweck

einiger Elemente der BPMN zu entfremden, oder eine gänzlich andere, auf die Modellierung von Agenten maßgeschneiderte Notation zu entwickeln. Beides wäre jedoch nicht zweckdienlich gewesen – es hätte einem Domänenexperten, der sich mit den etablierten Prozessnotationen auskennt, nicht geholfen, Agentensysteme zu modellieren, und hätte stattdessen nur eine weitere, mit den Standards konkurrierende Notation und eine weitere AOSE-Methodik geschaffen.

Stattdessen wurde die BPMN-Notation nur geringfügig erweitert, etwa um semantische Annotationen für Services und zusätzliche Nachrichtentypen. Weitere Ergänzungen dieser Art wären denkbar, etwa in Form von BPMN-*Artefacts*.

Die Umsetzung des Modellierungswerkzeugs ist für die Modellierung der Prozessdiagramme, deren Strukturierung und die Anwendung der Abbildung ausreichend, und durch die drei verschiedenen Implementierungen kann die Abbildung in verschiedenen Szenarien angewendet werden. Für den Produktivbetrieb wären jedoch mehr Komfort-Funktionen wünschenswert. Tatsächlich kann die graphische Modellierung der Prozesse mit allen für die Ausführung notwendigen Details für einen geübten Programmierer, der sich mit den entsprechenden Agenten-Frameworks auskennt, aufwändiger sein, als die Agenten-Verhalten direkt in Quellcode zu schreiben. Erst bei komplexeren Prozessen überwiegen die Vorteile, welche die übersichtliche Darstellung und die Erzeugung von nicht-trivialem Event-Handler-Code mit sich bringen, den Mehraufwand durch die graphische Modellierung. Mit geeigneten Modellierungshilfen könnte der Aufwand erheblich reduziert werden. Dies ginge jedoch deutlich über den im Rahmen dieser Arbeit entwickelten Prototypen hinaus.

10.3. Fazit

Zusammenfassend kann gesagt werden, dass die in dieser Arbeit gestellten Forschungsfragen positiv beantwortet werden konnten. Durch den Einsatz von BPMN zur Modellierung des gesamten Multiagenten-Systems lassen sich insbesondere die Kommunikation zwischen den Agenten, aber auch deren internes Verhalten sowie ihre Reaktionen auf verschiedene Ereignisse sehr gut modellieren, ohne dass die Notation selbst hierfür erweitert oder abgewandelt werden muss. Die in dieser Arbeit beschriebene Abbildung wurde in drei verschiedenen Ausprägungen umgesetzt und erfolgreich in mehreren praxisnahen Forschungsprojekten verwendet.

Teil IV.

Anhänge

Anhang A.

Auktions-Beispiel: Modelle und Ressourcen

An dieser Stelle soll das Beispiel aus Abschnitt 9.1 in allen Einzelheiten beschrieben werden. Insbesondere soll das Prozess- und das daraus abgeleitete Agenten-Modell anhand des formellen Metamodells beschreiben (siehe Kapitel 6) und der vollständigen generierten JIAC V AgentBeans-Quellcode gezeigt werden. Abbildung A.1 zeigt noch einmal das Prozessdiagramm, wobei jeder Knoten mit der Bezeichnung des entsprechenden Elements im formellen Modell annotiert ist.

A.1. Formale Beschreibung des Beispiel-Modells

Im Folgenden sollen sowohl das Prozess-Modell als auch das entsprechend der in dieser Arbeit beschriebenen Abbildung abgeleitete Agenten-Modell formal beschrieben werden.

A.1.1. Prozess-Modell

Das Business Process System *bps* besteht aus einem einzelnen Prozessdiagramm *bpd* mit zwei Pools, p_1 und p_2, die von zwei Participants pt_1 (der Initiator) und pt_2 (der Teilnehmer) ausgeführt werden. Ferner gibt es drei Message Flows, mf_1 bis mf_3, die jedoch rein dokumentarisch sind und keine weitere Funktion haben.

- $bps = (\text{'bps'},\ \{bpd\},\ \{pt_1,\ pt_2\})$
- $bpd = (\text{'bpd'},\ \{p_1,\ p_2\},\ \{mf_1,\ mf_2,\ mf_3\},\ \varnothing)$
- $p_1 = (\text{'Initiator'},\ wf_1,\ pt_1),\ \ p_2 = (\text{'Participant'},\ wf_2,\ pt_2)$

Im Workflow des Initiators wird zunächst eine Nachricht verschickt und danach in einem Subprozess auf Antworten gewartet, bis ein Timer-Event zuschlägt. Anschließend wird eine letzte Nachricht an den besten Teilnehmer verschickt. Der Looping Subprocess wird hierbei durch eine Kontrollfluss-Schleife approximiert.

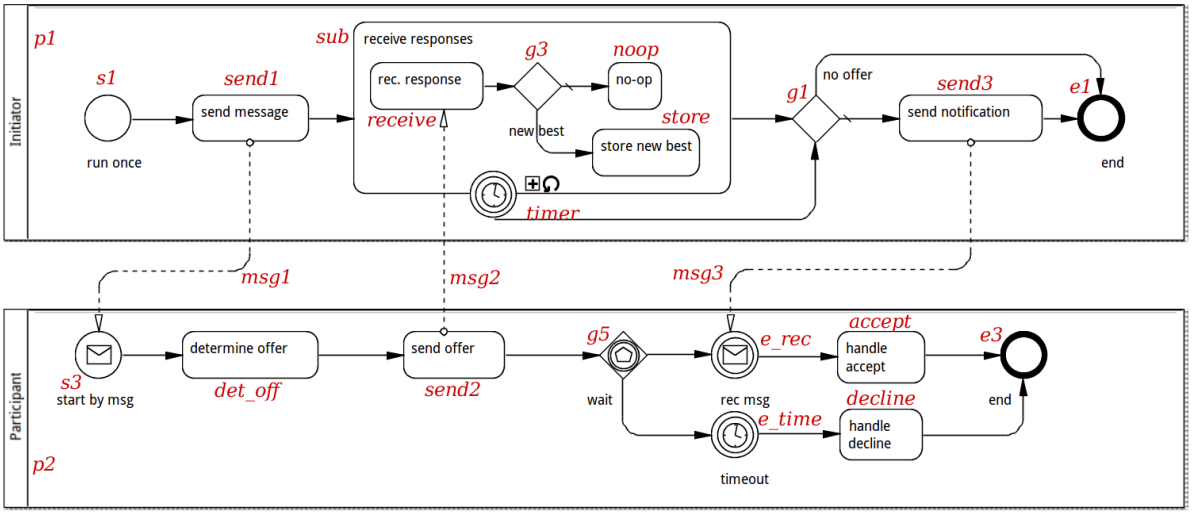


Abbildung A.1.: BPMN-Diagramm des Auktionsprotokolls, annotiert mit den Bezeichnungen der Symbole im formellen Prozessmodell.

- $wf_1 = (\{s_1,\ send_1,\ sub,\ g_1,\ send_3,\ e_1\},\ \{(s_1, send_1, \varepsilon),\ \ldots,\ (send_3, e_1, \varepsilon)\},\ props_1)$, wobei $props_1 = \{best,\ best_id\}$ und $sub = (\text{‘sub’},\ wf_{sub},\ \{timer\},\ \{(timer,\ g_1)\},\ \varnothing)$
- $wf_{sub} = (\{s_2,\ g_2,\ receive,\ g_3,\ store,\ noop,\ g_4,\ e_2\},\ seq,\ \varnothing)$, wobei $seq = \{\ldots,\ (g_4,\ g_2,\ \top),\ (g_3,\ store,\ b > best),\ \ldots\}$

Die Flow Objects in wf_1 enthalten jeweils typspezifische Attribute sowie Assignments. Jedes dieser Elemente ist ein Tupel in der Form $(id,\ type,\ [Ass,]\ Attr)$.

- $s_1 = (\text{‘s-1’},\ \textsc{none},\ \varnothing,\ \varnothing)$, $e_1 = (\text{‘e-1’},\ \textsc{none},\ \varnothing,\ \varnothing)$, $g_1 = (\text{‘g-1’},\ \textsc{xor},\ \varnothing)$
- $send_1 = (\text{‘send1’},\ \textsc{send},\ ass_{send1},\ \{(\text{‘impl’},\ msg_1)\})$, wobei $ass_{send_1} = \{(msg1_{\,cnt},\ \text{‘offer_msg’},\ \textsc{before}),\ (msg1_{\,rec},\ \text{‘channel:first-msg’},\ \textsc{before})\}$
- $send_3 = (\text{‘send3’},\ \textsc{send},\ ass_{send3},\ \{(\text{‘impl’},\ msg_3)\})$, wobei $ass_{send_3} = \{(msg1_{\,cnt},\ \text{‘winner!’},\ \textsc{before}),\ (msg1_{\,rec},\ best_id,\ \textsc{before})\}$
- $timer = (\text{‘timer’},\ \textsc{timer},\ \varnothing,\ \{(\text{‘timeout’},\ 10s)\})$

Ebenso verhält es sich mit den Elementen des Sub-Workflows wf_{sub}:

- $s_2 = (\text{‘s-2’},\ \textsc{none},\ \varnothing,\ \varnothing)$, $e_2 = (\text{‘e-2’},\ \textsc{none},\ \varnothing,\ \varnothing)$
- $g_2 = (\text{‘g-2’},\ \textsc{xor},\ \varnothing)$, $g_3 = (\text{‘g-3’},\ \textsc{xor},\ \varnothing)$, $g_4 = (\text{‘g-4’},\ \textsc{xor},\ \varnothing)$

- $receive$ = (‘receive’, RECEIVE, ass_{rec}, {(‘impl’, msg_2)}),
  wobei ass_{rec} = {(b, $msg2_{cnt}$, AFTER), (id, $msg2_{snd}$, AFTER)}
- $store$ = (‘store’, NONE, ass_{store}, $\varnothing$),
  wobei ass_{store} = {($best$, b, BEFORE), ($best_id$, id, BEFORE)}
- $noop$ = (‘noop’, NONE, $\varnothing$, $\varnothing$)

Der Workflow der Teilnehmers beginnt, wenn die initiale Nachricht empfangen wird. Sodann wird ein Angebot generiert und zum Initiator zurück geschickt. Danach wartet der Teilnehmer eine Zeit lang auf die Zuschlags-Nachricht und behandelt die beiden Fälle entsprechend.

- wf_2 = ({s_3, det_off, $send_2$, g_5, e_{rec}, e_{time}, $accept$, $decline$, e_3}, seq_2, $props_2$),
  wobei seq_2 = {(s_3, det_off, ε), …, ($accept$, e_3, ε)} und $props_2$ = {$sender$, $offer$}

Die einzelnen Flow-Objects des wf_2, in der Form (id, $type$, [Ass,] $Attr$):

- s_3 = (‘s-3’, MESSAGE, {($sender$, $msg1_{snd}$, AFTER)}, {(‘impl’, msg_1)})
- g_5 = (‘g-5’, EVENT, $\varnothing$), e_3 = (‘e-3’, NONE, $\varnothing$, $\varnothing$)
- det_off = (‘det’, SCRIPT, $\varnothing$, {(‘script’, ‘offer = Math.random()’)})
- $send_2$ = (‘send2’, SEND, ass_{send2}, {(‘impl’, msg_2)}),
  wobei ass_{send2} = {($msg2_{rec}$, $sender$, BEFORE), ($msg2_{cnt}$, $offer$, BEFORE)}
- e_{rec} = (‘e-r’, MESSAGE, $\varnothing$, {(‘impl’, msg_3)}),
  e_{time} = (‘e-t’, TIMER, $\varnothing$, {(‘timeout’, $12s$)})
- $accept$ = (‘acc’, SCRIPT, $\varnothing$, {(‘script’, ‘System.out.println(’Won!’)’)})
- $decline$ = (‘dec’, SCRIPT, $\varnothing$, {(‘script’, ‘System.out.println(’Lost’)’)})

Die Nachrichten msg_1, msg_2 und msg_3 sind jeweils Tupel bestehend aus drei Properties, *sender*, *receiver* und *content*, die durch Assignments belegt werden können.

- $\forall i \in \{1, 2, 3\}: \; msg_i = (msgi_{snd}, msgi_{rec}, msgi_{cnt})$

A.1.2. Agenten-Modell

Entsprechend der in dieser Arbeit vorgestellten Abbildung wird von diesem Prozess-Modell ein Agenten-Modell abgeleitet. Hierbei entspricht jeder Participant einer Rolle. Konkrete Agenten werden nicht abgeleitet und müssen explizit hinzugefügt werden. Die Pools entsprechen jeweils einem Plan, und das Message-Start-Event einer speziellen Reaktionsregel.

- $mas = (\text{'mas'},\ \varnothing,\ \{role_1,\ role_2\})$
- $role_1 = (\text{'role1'},\ \{plan_1\},\ \{rule_1\},\ \varnothing),\ \ role_2 = (\text{'role2'},\ \{plan_2\},\ \{rule_2\},\ \varnothing)$

Der Plan $plan_1$ entspricht dem Prozess des Initiators. Der Subprozess mit Event-Handler wird über das $fork$-Behaviour umgesetzt.

- $rule_1 = (\text{'rule1'},\ \text{'Agent wurde gestartet'},\ plan_1,\ \varnothing)$
- $plan_1 = (\text{'plan1'},\ [],\ [],\ \varepsilon,\ \varepsilon,\ script_1)$
- $script_1 = seq(snd_1,\ sub,\ timer,\ join(\text{'sub'}),\ stop(\text{'t1'}),\ cond(\text{'best} > 0\text{'},\ snd_2,\ nop))$, wobei
  - $snd_1 = send(m)$, wobei $m = (self,\ \text{'first-msg'},\ \text{'offer_msg'})$
  - $sub = fork(\text{'sub'},\ while(\top,\ seq(rec,\ cnd)))$
  - $rec = receive(m = (sender,\ self,\ offer)),\ assign(b,\ offer),\ assign(id,\ sender)$
  - $cnd = cond(\text{'b} > \text{best'},\ seq(assign(best,\ b),\ assign(best_id,\ id)),\ nop)$
  - $timer = fork(\text{'t1'},\ seq(wait(\text{'10s'}),\ stop(\text{'sub'})))$
  - $snd_2 = send(m)$, wobei $m = (self,\ best_id,\ \text{'winner!'})$

Der Prozess des Teilnehmers wurde auf den Plan $plan_2$ abgebildet. Das Event-based Gateway wird, genau wie der Subprozess, über ein $fork$ umgesetzt. Außerdem verfügt diese Rolle über eine Reaktionsregel zum Ausführen des Plans, entsprechend des Start-Events.

- $rule_2 = (\text{'rule2'},\ \text{'empfange } m = (*,\ \text{'first-msg'},\ \text{'offer_msg'})\text{'},\ plan_2,\ \{(m,\ msg)\})$
- $plan_2 = (\text{'plan2'},\ [msg],\ [],\ \varepsilon,\ \varepsilon,\ script_2)$
- $script_2 = seq(rec_1,\ offer,\ snd_3,\ loop,\ eh_m,\ eh_t,\ join(\text{'g5'}),\ stop(\text{'e_rec'}),\ stop(\text{'e_time'}),\ branch_m,\ branch_t)$, wobei

- $rec_1 = assign(\text{'sndr'}, msg_{snd})$
- $offer = \text{'offer = Math.random()'}$
- $snd_3 = send(m)$, wobei $m = (self, \text{'sndr'}, \text{'offer'})$
- $loop = fork(\text{'g5'}, while(\top, nop))$
- $eh_m = fork(\text{'e_rec'}, seq(assign(\text{'em'}, \bot), e_{rec}, assign(\text{'em'}, \top), stop(\text{'g5'}))),$ wobei $e_{rec} = receive(m = (\text{'sndr'}, self, \text{'winner!'}))$
- $eh_t = fork(\text{'e_time'}, seq(assign(\text{'et'}, \bot), e_{time}, assign(\text{'et'}, \top), stop(\text{'g5'}))),$ wobei $e_{time} = wait(\text{'12s'})$
- $branch_m = cond(\text{'em'}, \text{'System.out.println('Won!')'}, nop)$
- $branch_t = cond(\text{'et'}, \text{'System.out.println('Lost')'}, nop)$

Dieses Agenten-Modell kann nun auf verschiedene Arten umgesetzt werden, etwa mit einer der drei in dieser Arbeit vorgestellten Implementierungen.

A.2. Erzeugter JIAC V AgentBeans-Quellcode

Im Folgenden soll der vollständige generierte JIAC AgentBeans Code für beide Rollen, Initiator und Teilnehmer, gezeigt werden. Der Code wird genau so gezeigt, wie er erzeugt wurde; lediglich die generierten JavaDoc-Kommentare und die Imports wurden aus Platzgründen entfernt.

Listing A.1 zeigt den Code der Initiator-Rolle. Gut zu erkennen sind die Prozess-Properties, die auf Instanzattribute abgebildet wurden, die Workflow-Methode mit den Event-Handler-Threads sowie die verschiedenen Activity-Methoden. Die beiden Methoden `doStart` und `execute` bewirken zusammen, dass der Prozess entsprechend des Start-Events genau einmal ausgeführt wird, sobald der Agent startet. Der Subprozess wurde auf die eingebettete innere Klasse abgebildet.

Listing A.1: Erzeugte AgentBean: Initiator

```
1 public class Initiator_AuctionProtocol extends AbstractWorkflowBean {
2
3   protected Double best;
4   protected String bestId;
5   protected SimpleMsg message;
6   protected long executed_runOnce;
7
8   public void doStart() throws Exception {
9     super.doStart();
```

```
10      executed_runOnce = 0;
11      join("unicast_message");
12    }
13
14    public void execute() {
15      super.execute();
16      if (executed_runOnce == 0) {
17        executed_runOnce = 1;
18        workflow();
19      }
20    }
21
22    public void workflow() {
23      message = new de.dailab.vsdt.example.protocol.onto.SimpleMsg();
24      message.message = "first";
25      sendMessage();
26      Thread thread_receiveResponses = new Thread(new Runnable() {
27        public void run() {
28          Subprocess_receiveResponses receiveResponses =new Subprocess_receiveResponses();
29          receiveResponses.workflow();
30        }
31      });
32      EventHandler thread_timeout =new TimeoutEventHandler(thread_receiveResponses,10000);
33      try {
34        thread_receiveResponses.start();
35        thread_timeout.start();
36        thread_receiveResponses.join();
37        thread_timeout.stopEventHandler();
38      } catch (InterruptedException e) {
39      }
40      if (best == 0) {
41      } else {
42        sendNotification();
43      }
44    }
45
46    protected void sendMessage() {
47      log.trace("Executing␣Activity␣'sendMessage'");
48      SimpleMsg msg = new SimpleMsg();
49      String __receiver = new String();
50      msg = message;
51      send(msg, "first_message", true);
52    }
53
54    protected void sendNotification() {
55      log.trace("Executing␣Activity␣'sendNotification'");
56      SimpleMsg msg = new SimpleMsg();
57      String __receiver = new String();
58      msg.message = "winner!";
59      msg.number = best;
60      __receiver = bestId;
61      send(msg, __receiver, false);
62    }
```

```
63
64   class Subprocess_receiveResponses {
65     protected Double b;
66     protected String id;
67
68     public void workflow() {
69       best = 0.0;
70       while (true) {
71         if (checkInterrupted()) return;
72         recResponse();
73         if (b > best) {
74           storeNewBest();
75         } else {
76           noOp();
77         }
78       }
79     }
80
81     protected void recResponse() {
82       log.trace("Executing Activity 'recResponse'");
83       SimpleMsg msg = new SimpleMsg();
84       String __sender = new String();
85       IJiacMessage __msg = receiveMessage(SimpleMsg.class, null, -1);
86       if (__msg == null) return; // in case of interruption
87       __sender = __msg.getSender().getName();
88       msg = (SimpleMsg) __msg.getPayload();
89       message = msg;
90       id = __sender;
91       b = msg.number;
92     }
93
94     protected void noOp() {
95       log.trace("Executing Activity 'noOp'");
96     }
97
98     protected void storeNewBest() {
99       log.trace("Executing Activity 'storeNewBest'");
100      best = b;
101      bestId = id;
102    }
103  }
104 }
```

Der Code des Teilnehmers ist in Listing A.2 dargestellt. Die Workflow-Methode ist etwas komplexer und enthält abermals Threads, diesmal für das Event-based-XOR-Gateway. Das Start-Event wurde auf einen `SpaceObserver` abgebildet, der sobald der Agent startet auf eintreffende Nachrichten lauscht und ggf. den Prozess startet.

Listing A.2: Erzeugte AgentBean: Participant

```
1 public class Participant_AuctionProtocol extends AbstractWorkflowBean {
2
```

```
3   protected SimpleMsg message;
4   protected Double n;
5   protected String id;
6
7   public void doStart() throws Exception {
8     super.doStart();
9     join("first_message");
10    SpaceObserver<IFact> messageObserver__Ubl9IKktEeOpeoP = new SpaceObserver<IFact>() {
11      public void notify(SpaceEvent<? extends IFact> event) {
12        if (event instanceof WriteCallEvent) {
13          IJiacMessage message = (IJiacMessage) ((WriteCallEvent<?>) event).getObject();
14          if (message.getPayload() instanceof SimpleMsg && message.getHeader(
15              IJiacMessage.Header.SEND_TO).equalsIgnoreCase("first_message")) {
16            memory.remove(message);
17            workflow(message);
18          }
19        }
20      }
21    };
22    memory.attach(messageObserver__Ubl9IKktEeOpeoP, new JiacMessage());
23  }
24
25  public void workflow(IJiacMessage __msg) {
26    String __sender = __msg.getSender().getName();
27    SimpleMsg msg = (SimpleMsg) __msg.getPayload();
28    message = msg;
29    id = __sender;
30    determineOffer();
31    sendOffer();
32    Thread thread_wait = new Thread(new Runnable() {
33      public void run() {
34        while (true) {
35          try {
36            Thread.sleep(500);
37          } catch (InterruptedException e) {
38            Thread.currentThread().interrupt();
39            return;
40          }
41        }
42      }
43    });
44    EventHandler thread_timeout = new TimeoutEventHandler(thread_wait, 15000);
45    EventHandler thread_recMsg = new MessageEventHandler(thread_wait, null,
46        SimpleMsg.class);
47    try {
48      thread_wait.start();
49      thread_timeout.start();
50      thread_recMsg.start();
51      thread_wait.join();
52      thread_timeout.stopEventHandler();
53      thread_recMsg.stopEventHandler();
54    } catch (InterruptedException e) {
55    }
```

```
56      if (thread_timeout.isTriggered()) {
57        handleDecline();
58      }
59      if (thread_recMsg.isTriggered()) {
60        handleAccept();
61      }
62    }
63
64    protected void determineOffer() {
65      log.trace("Executing␣Activity␣'determineOffer'");
66      java.util.Random rand = new java.util.Random();
67      n = rand.nextInt(100) + 1.0;
68    }
69
70    protected void sendOffer() {
71      log.trace("Executing␣Activity␣'sendOffer'");
72      SimpleMsg msg = new SimpleMsg();
73      String __receiver = new String();
74      msg.number = n;
75      __receiver = id;
76      send(msg, __receiver, false);
77    }
78
79    protected void handleDecline() {
80      log.trace("Executing␣Activity␣'handleDecline'");
81      System.out.println("LOST:␣Rolled␣a␣low␣number:␣" + n);
82    }
83
84    protected void handleAccept() {
85      log.trace("Executing␣Activity␣'handleAccept'");
86      System.out.println("WON:␣Rolled␣the␣highest␣number:␣" + n);
87    }
88  }
```

Schließlich zeigt Listing A.3 die Spring-Konfigurationsdatei um ein einfaches Multiagenten-System mit diesen Rollen zu erzeugen und zu starten, in diesem Falle mit einem Agenten, der die Initiator-Rolle implementiert, und fünf Agenten mit der Teilnehmer-Rolle, die dem Initiator ihre zufällig erzeugten Gebote schicken. Diese Datei wird nicht generiert und muss gesondert erstellt werden.

Listing A.3: Agenten-Konfiguration (nicht erzeugt)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5  ␣␣␣␣␣␣http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7      <import resource="classpath:de/dailab/jiactng/agentcore/conf/AgentNode.xml" />
8      <import resource="classpath:de/dailab/jiactng/agentcore/conf/Agent.xml" />
9
10     <bean name="VsdtAuctionExampleNode" parent="NodeWithJMX">
```

```
11          <property name="agents">
12              <list>
13                  <bean parent="InitiatorAgent" />
14                  <bean parent="ParticipantAgent" />
15                  <bean parent="ParticipantAgent" />
16                  <bean parent="ParticipantAgent" />
17                  <bean parent="ParticipantAgent" />
18                  <bean parent="ParticipantAgent" />
19              </list>
20          </property>
21      </bean>
22
23      <bean name="InitiatorAgent" parent="NonBlockingAgent" scope="prototype">
24          <property name="agentBeans">
25              <list>
26                  <bean name="Initiator_AuctionProtocol" scope="prototype"
27                        class="example.protocol.beans.Initiator_AuctionProtocol">
28                      <property name="executionInterval" value="1000" />
29                  </bean>
30              </list>
31          </property>
32      </bean>
33
34      <bean name="ParticipantAgent" parent="NonBlockingAgent" scope="prototype">
35          <property name="agentBeans">
36              <list>
37                  <bean name="Participant_AuctionProtocol" scope="prototype"
38                        class="example.protocol.beans.Participant_AuctionProtocol">
39                      <property name="executionInterval" value="1000" />
40                  </bean>
41              </list>
42          </property>
43      </bean>
44
45  </beans>
```

Literaturverzeichnis

[1] S. Albayrak, M. Blumendorf, S. Feuerstack, T. Küster, A. Rieger, V. Schwartze, C. Wirth, and P. Zernicke. Ein Framework für Ambient Assisted Living Services. In *Ambient Assisted Living 2009, 2. Deutsche AAL Kongress*, Berlin, 2009. VDE.

[2] T. Allweyer. Kollaborationen, Choreographien und Konversationen in BPMN 2.0 – Erweiterte Konzepte zur Modellierung übergreifender Geschäftsprozesse. Technical report, Fachhochschule Kaiserslautern, Juni 2009.

[3] J. L. Alty, F. Griffiths, N. R. Jennings, E. H. Mamdani, A. Struthers, and M. E. Wiegand. ADEPT – advanced decision environment for process tasks: Overview & architecture. In *Proc. BCS Expert Systems 96 Conferenc*, pages 5–23. Agentlink Clearinghouse [http://eprints.agentlink.org/perl/oai2] (United Kingdom), 1994.

[4] B. Bauer, J. P. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering, First International Workshop, AOSE 2000, Revised Papers*, volume 1957 of *LNCS*, pages 91–104. Springer, 2001.

[5] F. Bellifemine, A. Poggi, and G. Rimassa. JADE – a FIPA-compliant agent framework. Internal technical report, Telecom Italia, 1999. Part of this report has been also published in Proc. of PAAM'99, London, April 1999, pp.97-108.

[6] F. Bergenti, G. Caire, and D. Gotta. Interactive workflows with WADE. *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 0:10–15, 2012.

[7] F. Bergenti, G. Caire, D. Gotta, D. Long, and G. Sacchi. Enacting BPM-oriented workflows with Wade. In *Proc. of the 12th Workshop on Objects and Agents, Rende (CS), Italy*, pages 112–116, July 2011.

[8] C. Bernon, M. Cossentino, M.-P. Gleizes, P. Turci, and F. Zambonelli. A study of some multi-agent meta-models. In *Proceedings of the 5th International Conference on Agent-Oriented Software Engineering*, pages 62–77. Springer, 2004.

[9] E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, and E. Weiss. Graphical definition of in-place transformations in the eclipse modeling framework. In *Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06)*, Genova, Italy, October 2006.

[10] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.

[11] R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming. Languages, Platforms and Applications*. Multiagent Systems, Artificial Societies, and Simulated Organizations. Springer, 2005.

[12] R. H. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni. *Multi-Agent Programming: Languages, Tools and Applications*. Springer, 2009.

[13] R. H. Bordini, J. F. Hübner, and R. Vieira. Jason and the golden fleece of agent-oriented programming. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming*, pages 3–37. Springer, 2005.

[14] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Havard University Press, Cambridge, MA, 1987.

[15] L. Braubach, A. Pokahr, K. Jander, W. Lamersdorf, and B. Burmeister. Go4Flex: Goal-oriented process modelling. In *4th International Symposium on Intelligent Distributed Computing (IDC'2010)*, 2010.

[16] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal representation for BDI agent systems. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, volume 3346 of *LNCS*, pages 44–65. Springer Berlin Heidelberg, 2005.

[17] N. Braun, R. Cissée, and S. Albayrak. An agent-based approach to user-initiated semantic service interconnection. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. P. Sycara, editors, *Service-Oriented Com-*

puting: Agents, Semantics, and Engineering: AAMAS 2007 International Workshop, SOCASE 2007, Honolulu, HI, USA, May 14, 2007. Proceedings, pages 49–62. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[18] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, May 2004.

[19] P. A. Buhler and J. M. Vidal. Towards adaptive workflow enactment using multiagent systems. *Inf. Technol. and Management*, 6:61–87, January 2005. Kluwer Academic Publishers, Hingham, MA, USA.

[20] P. A. Buhler, J. M. Vidal, and H. Verhagen. Adaptive workflow = web services + agents. In *Proc. of Int. Conf. on Web Services*, pages 131–137. CSREA Press, 2003.

[21] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-agents for agile goal-oriented business processes. In M. Berger, B. Burg, and S. Nishiyama, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track*, pages 37–44, Estoril, Portugal, May 12–16 2008. IFAAMAS.

[22] B. Burmeister, H.-P. Steiert, T. Bauer, and H. Baumgärtel. Agile processes through goal- and context-oriented business process modeling. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *LNCS*, pages 217–228. Springer, 2006.

[23] P. Busetta, N. Howden, R. Rönnquist, and A. Hodgson. Structuring BDI agents in functional clusters. In *6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL)*, pages 277–289, London, UK, 2000. Springer.

[24] P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. JACK – components for intelligent agents in Java. Technical report, Agent Oriented Software Pty, Ltd., 1999.

[25] G. Cabri, L. Leonardi, M. Puviani, and R. Quitadamo. Comparing service-oriented computing and agent-oriented programming towards integration. *Web Intelligence and Agent Systems*, 9(2):135–146, 2011.

[26] G. Cabri, M. Puviani, and R. Quitadamo. Connecting methodologies and infrastructures in the development of agent systems. In *Proc. of Int. Multiconf. on Computer Science and Information Technology (IMCSIT 2008)*, pages 17–23, Wisla, Poland, October 2008. IEEE.

[27] G. Caire, D. Gotta, and M. Banzi. WADE: A software platform to develop mission critical applications exploiting agents and workflows. In M. Berger, B. Burg, and S. Nishiyama, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track*, pages 29–36, Estoril, Portugal, May 12–16 2008. IFAAMAS.

[28] G. Caire, E. Quarantotto, M. Porta, and G. Sacchi. WOLF – an Eclipse plug-in for WADE. In *Proc. of ACEC 2008*, 2008.

[29] M. Calisti and D. Greenwood. Goal-oriented autonomic process modeling and execution for next generation networks. In *3rd IEEE International Workshop on Modelling Autonomic Communications Environments (MACE)*. Springer, 2008.

[30] G. Casella and V. Mascardi. From AUML to WS-BPEL. Technical Report DISI-TR-06-01, Dipartimento di Informatica e Scienze dell'Informatione, Università di Genova, 2006.

[31] R. Cervenka and I. Trencansky. Agent-oriented development methodology (ADEM). Technical report, Whitestein Technologies, February 2005.

[32] R. Cervenka and I. Trencansky. *The Agent Modeling Language – AML. A Comprehensive Approach to Modeling Multi-Agent Systems*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Basel, 2007.

[33] B. Chandrasekaran, J. R. Josephson, and V. R. Benjamins. Ontologies: What are they? why do we need them? *IEEE Intelligent Systems and Their Applications*, 14(1):20–26, 1999. Special Issue on Ontologies.

[34] Y.-H. Chen-Burger, D. Robertson, and L. Guo. Business process model based multi-agent system development. In L. Guo, editor, *The 2nd Workshop On Collaboration Agents: Autonomous Agents for Collaborative Environments*, pages 11–19, September 2004.

[35] C. Cheong and M. Winikoff. Improving flexibility and robustness in agent interactions: Extending Prometheus with Hermes. In A. Garcia, R. Choren, C. Lucena,

P. Giorgini, T. Holvoet, and A. Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNCS*. Springer, 2006.

[36] P. R. Cohen and H. J. Levesque. Speech acts and rationality. In W. Mann, editor, *Proc. of 23rd Meeting of the Association for Computational Linguistics*, pages 49–59, Morristown, New Jersey, 1985. Association for Computational Linguistics.

[37] M. Cossentino. From requirements to code with the PASSI methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-orineted Methodologies*. Idea Group Publishing, 2005.

[38] M. Cossentino, V. Hilaire, A. Molesini, and V. Seidita. *Handbook on Agent-Oriented Design Processes*. Springer Publishing Company, Berlin, Heidelberg, 2014.

[39] L. M. Cysneiros and E. Yu. Addressing agent autonomy in business process management – with case studies on the patient discharge process, 2004.

[40] M. Dastani. Programming multi-agent systems. In *12th Int. Workshop on Agent-Oriented Software Engineering (AOSE 2011)*, pages 23–52, Taipei, Taiwan, May 2011.

[41] I. Dickinson and M. Wooldridge. Agents are not (just) web services: Considering BDI agents and web services. In *Proc. of 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE)*, Utrecht, The Netherlands, July 2005.

[42] R. M. Dijkman, M. Dumas, and C. Ouyang. Formal semantics and analysis of BPMN process models using Petri nets, 2008.

[43] R. M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in BPMN. *Information & Software Technology*, 50(12):1281–1294, 2008.

[44] E. W. Dijkstra. Go To statement considered harmful. *Comm. ACM*, 11(3):147–148, 1968. letter to the Editor.

[45] H. Endert, B. Hirsch, T. Küster, and S. Albayrak. Towards a mapping from BPMN to agents. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller,

S. Stoutenburg, and K. P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering – AAMAS 2007 International Workshop, SOCASE 2007, Honolulu, HI, USA, May 14, 2007. Proceedings*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin Heidelberg, 2007.

[46] H. Endert, T. Küster, B. Hirsch, and S. Albayrak. Mapping BPMN to agents: An analysis. In M. Baldoni, C. Baroglio, and V. Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies (AWESOME)*, pages 43–58, 2007.

[47] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall, Indiana, USA, August 2005.

[48] J. Fähndrich, T. Küster, and N. Masuch. Semantic service management and orchestration for adaptive and evolving processes. *International Journal on Advances in Internet Technology*, 9(3&4):75–88, 2016.

[49] J. Fähndrich, T. Küster, and N. Masuch. Semantic service management for enabling adaptive and evolving processes. In *Proc. of 11th Int. Conf. on Internet and Web Applications and Services (ICIW 2016)*, pages 46–53, Valencia, Spain, May 22–26 2016.

[50] J. Fähndrich, N. Masuch, H. Yildirim, and S. Albayrak. Towards automated service matchmaking and planning for multi-agent systems with OWL-S – approach and challenges. In A. R. Lomuscio, S. Nepal, F. Patrizi, B. Benatallah, and I. Brandić, editors, *Service-Oriented Computing – ICSOC 2013 Workshops*, volume 8377 of *LNCS*, pages 240–247. Springer, 2014.

[51] K. Figl, J. Mendling, and M. Strembeck. Towards a usability assessment of process modeling languages. In M. Nüttgens, F. Rump, J. Mendling, and N. Gehrke, editors, *Proc. of 8th Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK 2009)*, volume 554 of *CEUR Workshop Proceedings*, pages 118–136, Berlin, Germany, 26-27 November 2009.

[52] FIPA. FIPA modeling: Interaction diagrams. Technical Report 03-07-02, Foundation for Intelligent Physical Agents, july 2003.

[53] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *ECAI '96: Proc. of Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35, London, UK, 1997. Springer.

[54] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design patterns: Abstraction and reuse of object-oriented design. *Lecture Notes in Computer Science*, 707:406–431, 1993.

[55] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[56] D. Greenwood and M. Calisti. Engineering web service - agent integration. *2004 IEEE International Conference on Systems, Man and Cybernetics*, 2:1918–1925, 2004.

[57] D. Greenwood and G. Rimassa. Autonomic goal-oriented business process management. In *3rd International Conference on Autonomic and Autonomous Systems (ICAS)*, 2007.

[58] T. Gschwind, J. Koehler, and J. Wong. Applying patterns during business process modeling. In *BPM '08: Proc. of 6th International Conference on Business Process Management*, pages 4–19. Springer, 2008.

[59] L. Guo, D. Robertson, and Y.-H. Chen-Burger. Using multi-agent platform for pure decentralised business workflows. *Web Intelligence and Agent Systems*, 6(3):295–311, 2008.

[60] C. Hahn. A domain specific modeling language for multiagent systems. In Pagham, Parkes, Müller, and Parsons, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 233–240, Estoril, Portugal, May 12–16 2008. IFAAMAS.

[61] C. Hahn, C. Madrigal-Mora, and K. Fischer. A platform-independent metamodel for multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):239–266, 4 2009. Springer.

[62] C. Hahn, C. M. Mora, K. Fischer, B. Elvesaeter, A.-J. Berre, and I. Zinnikus. Meta-models, models, and model transformations: Towards interoperable agents.

In *Multiagent System Technologies*, volume 4196/2006 of *LNCS*, pages 123–134. Springer, 2006.

[63] C. Hahn, I. Zinnikus, S. Warwas, and K. Fischer. Automatic generation of executable behavior: A protocol-driven approach. In M.-P. Gleizes and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering X*, volume 6038 of *LNCS*, pages 110–124. Springer Berlin Heidelberg, 2011.

[64] M. Hepp. Ontologies: State of the art, business potential, and grand challenges. *Ontology Management*, I:3–22, 2008.

[65] A. Heßler, B. Hirsch, T. Küster, and S. Albayrak. Agentstore – a pragmatic approach to agent reuse. In F. Dechesne, H. Hattori, A. ter Mors, J. M. Such, D. Weyns, and F. Dignum, editors, *Advanced Agent Technology – AAMAS 2011 Workshops, AMPLE, AOSE, ARMS, DOCM3AS, ITMAS, Taipei, Taiwan, May 2-6, 2011. Revised Selected Papers*, volume 7068 of *LNAI*, pages 128–138. Springer Berlin Heidelberg, May 2012.

[66] B. Hirsch, T. Konnerth, M. Burkhardt, and S. Albayrak. Programming service oriented agents. In M. Calisti, F. P. Dignum, R. Kowalczyk, F. Leymann, and R. Unland, editors, *Service-Oriented Architecture and (Multi-)Agent Systems Technology*, number 10021 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2010. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany.

[67] B. Hirsch, T. Konnerth, and A. Heßler. Merging agents and services – the JIAC agent platform. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 159–185. Springer, 2009.

[68] M. Holtz. Programmierung reaktiver Agenten mit Rule Engines. Bachelor thesis, Technische Universität Berlin, April 2010.

[69] C.-E. Hrabia, T. Küster, M. Voß, F. D. Pozo Pardo, and S. Albayrak. Adaptive multi-stage optimisation for EV charging integration into smart grid control. In Q. Chen, P. Torroni, S. Villata, J. Hsu, and A. Omicini, editors, *PRIMA 2015: Principles and Practice of Multi-Agent Systems – 18th International Conference, Bertinoro, Italy, October 26-30, 2015, Proceedings*, volume 9387 of *LNAI*, pages 622–630. Springer International Publishing, 2015.

[70] C.-E. Hrabia, F. D. Pozo Pardo, T. Küster, and S. Albayrak. Multi-stage smart grid optimisation with a multiagent system (demonstration). In R. H. Bordini, E. Elkind, G. Weiß, and P. Yolum, editors, *Proc. of 14th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, Istanbul, Turkey, May 4–8 2015. IFAAMAS.

[71] M.-P. Huget. Agent UML notation for multiagent system design. *IEEE Internet Computing*, 8:63–71, July 2004. IEEE Educational Activities Department, Piscataway, NJ, USA.

[72] K. Jander, L. Braubach, and W. Lamersdorf. Distributed monitoring and workflow management for goal-oriented workflows. *Concurrency and Computation: Practice and Experience*, 2015.

[73] K. Jander, L. Braubach, A. Pokahr, W. Lamersdorf, and K. Wack. Goal-oriented processes with GPMN. *International Journal on Artificial Intelligence Tools*, 20(6):1021–1041, 2011.

[74] K. Jander and W. Lamersdorf. GPMN-Edit: High-level and goal-oriented workflow modeling. In *Workshops der wissenschaftlichen Konferenz Kommunikation in verteilten Systemen 2011 (WowKiVS 2011)*, 2011.

[75] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296, 2000.

[76] N. R. Jennings, P. Faratin, M. J. Johnson, and M. E. Wiegand. Using intelligent agents to manage business processes. In *1st Int. Conf. on The Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 345–360, 1996.

[77] N. R. Jennings, T. J. Norman, P. Faratin, and B. Odgers. Autonomous agents for business process management. *Journal of Applied Artificial Intelligence*, 14:145–189, 2000.

[78] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:275–306, 1998. Kluwer Academic Puublishers.

[79] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten. Technical Report 89, Universität des Saarlands, Saarbrücken, 1992.

[80] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *CAiSE '00: Proc. of 12th Int. Conf. on Advanced Information Systems Engineering*, pages 431–445, London, UK, 2000. Springer.

[81] M. Klusch. Semantic web service coordination. In M. Calisti, M. Walliser, S. Brantschen, M. Herbstritt, M. Schumacher, H. Schuldt, and H. Helkki, editors, *CASCOM: Intelligent Service Coordination in the Semantic Web*, pages 59–104. Birkhäuser Basel, 2008.

[82] M. Klusch. Semantic web service description. In M. Calisti, M. Walliser, S. Brantschen, M. Herbstritt, M. Schumacher, H. Schuldt, and H. Helkki, editors, *CASCOM: Intelligent Service Coordination in the Semantic Web*, pages 31–57. Birkhäuser Basel, 2008.

[83] J. Koehler and R. Hauser. Untangling unstructured cyclic flows – a solution based on continuations. In R. Meesman and Z. Tari, editors, *CooplIS/DOA/ODBASE 2004*, volume 3290 of *LNCS*, pages 121–138. Springer, 2004. IBM Zurich Research Laboratory.

[84] T. Konnerth, J. Chinnow, S. Kaiser, D. Grunewald, K. Bsufka, and S. Albayrak. Integration of simulations and MAS for smart grid management systems. In *Proc. of 3rd Int. Workshop on Agent Technologies for Energy Systems (ATES 2012)*, pages 51–58, Valencia, Spain, 2012.

[85] S. Krumnow. Modeling and executing ad-hoc subprocesses in different environments. Bachelor's thesis, Hasso Plattner Institut, June 2008.

[86] T. Küster. Development of a visual service design tool providing a mapping from BPMN to JIAC. Diploma thesis, Technische Universität Berlin, April 2007.

[87] T. Küster. *The Visual Service Design Tool, Version 1.4.7.* DAI-Labor, TU Berlin, April 2017. `http://jiac.de/Downloads/toolipse/vsdt-manual_1704.pdf` (2017/06/06).

[88] T. Küster and A. Heßler. Towards transformations from BPMN to heterogeneous systems. In D. Ardagna, M. Mecella, and J. Yang, editors, *Business Process Management Workshops – BPM 2008 International Workshops, Milano, Italy, September 1-4, 2008. Revised Papers*, volume 17 of *LNBIP*, pages 200–211. Springer Berlin Heidelberg, 2009.

[89] T. Küster, A. Heßler, and S. Albayrak. Towards process-oriented modelling and creation of multi-agent systems. In F. Dalpiaz, J. Dix, and M. B. van Riemsdijk, editors, *Engineering Multi-Agent Systems – Second International Workshop, EMAS 2014, Paris, France, May 5-6, 2014, Revised Selected Papers*, volume 8758 of *LNAI*, pages 163–180. Springer International Publishing, 2014.

[90] T. Küster, A. Heßler, and S. Albayrak. Process-oriented modelling, creation, and interpretation of multi-agent systems. *Int. J. Agent-Oriented Software Engineering*, 5(2/3):108–133, 2016.

[91] T. Küster and M. Lützenberger. An overview of a mapping from BPMN to agents (extended abstract). In R. H. Bordini, E. Elkind, G. Weiß, and P. Yolum, editors, *Proc. of 14th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2015)*, pages 1783–1784, Istanbul, Turkey, May 4–8 2015. IFAAMAS.

[92] T. Küster, M. Lützenberger, and S. Albayrak. A formal description of a mapping from business processes to agents. In M. Baldoni, L. Baresi, and M. Dastani, editors, *Engineering Multi-Agent Systems – Third International Workshop, EMAS 2015, Istanbul, Turkey, May 5, 2015, Revised, Selected, and Invited Papers*, volume 9318 of *LNAI*, pages 153–170. Springer International Publishing, 2015.

[93] T. Küster, M. Lützenberger, A. Heßler, and B. Hirsch. Integrating process modelling into multi-agent system engineering. *Multiagent and Grid Systems*, 8(1):105–124, January 2012.

[94] H. Lin, Z. Zhao, H. Li, and Z. Chen. A novel graph reduction algorithm to identify structural conflicts. In *HICSS '02: Proc. of 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, pages 289–298, Washington, DC, USA, 2002. IEEE Computer Society.

[95] R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. In W. M. P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Business Process Management*, volume 3649, pages 268–284, 2005.

[96] M. Lützenberger, T. Konnerth, and T. Küster. Programming of multiagent applications with JIAC. In P. Leitão and S. Karnouskos, editors, *Industrial Agents – Emerging Applications of Software Agents in Industry*, chapter 21, pages 381–400. Morgan Kaufmann, Boston, 2015.

[97] M. Lützenberger, T. Küster, A. Heßler, and B. Hirsch. Unifying JIAC agent development with AWE. In L. Braubach, W. van der Hoek, P. Petta, and A. Pokahr, editors, *Multiagent System Technologies – 7th German Conference, MATES 2009, Hamburg, Germany, September 9-11, 2009. Proceedings*, volume 5774 of *LNAI*, pages 220–225. Springer Berlin Heidelberg, 2009.

[98] M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Heßler, J. Keiser, M. Burkhardt, S. Kaiser, J. Tonn, M. Kaisers, and S. Albayrak. A multi-agent approach to professional software engineering. In M. Cossentino, A. El Fallah Seghrouchni, and M. Winikoff, editors, *Engineering Multi-Agent Systems – First International Workshop, EMAS 2013, St. Paul, MN, USA, May 6-7, 2013, Revised Selected Papers*, volume 8245 of *LNAI*, pages 156–175. Springer Berlin Heidelberg, 2013.

[99] D. Martin, R. Hodgson, I. Horrocks, and P. Yendluri. OWL 1.1 web ontology language, 2006. `http://www.w3.org/Submission/2006/10/` (2017/06/06).

[100] V. Mascardi and G. Casella. Intelligent agents that reason about web services: a logic programming approach. *Proc. of ICLP'06 Workshop Workshop on Applications of Logic Programming in the Semantic Web and Semantic Web Services (ALPSWS 2006)*, 196:55–70, 2006.

[101] N. Masuch, C. Kuster, and S. Albayrak. Semantic service manager – enabling semantic web technologies in multi-agent systems. In *Proc. of Joint Workshops on Semantic Web and Big Data Technologies, INFORMATIK 2014*, volume 232 of *LNI*, Stuttgart, Germany, 2014. GI.

[102] N. Masuch, T. Küster, J. Fähndrich, M. Lützenberger, and S. Albayrak. A multi-agent platform for augmented reality based product-service systems (demonstration). In S. Das, E. Durfee, K. Larson, and M. Winikoff, editors, *Proc. of 16th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, Sao Paulo, Brazil, May 8–12 2017.

[103] J. Mendling, K. B. Lassen, and U. Zdun. On the transformation of control flow between block-oriented and graph-oriented process modelling languages. *International Journal of Business Process Integration and Management (IJBPIM)*, 3(2):96–108, 2008.

[104] J. Mendling, G. Neumann, and M. Nüttgens. Towards workflow pattern support of event-driven process chains (EPC). In *Proc. of 2nd Workshop XML4BPM*, pages 23–38, 2005.

[105] D. Moldt, J. Quenum, C. Reese, and T. Wagner. Improving a workflow management system with an agent flavour. In S. Donatelli, J. Kleijn, R. J. Machado, and J. M. Fernandes, editors, *Recent Advances in Petri Nets and Concurrency*, pages 301–316, 2012.

[106] P. Moraitis and N. I. Spanoudakis. The Gaia2JADE process for multi-agent systems development. *Applied Artificial Intelligence*, 20(2-4):251–273, 2006.

[107] J. P. Müller, M. Pischel, and M. Thiel. Modeling reactive behaviour in vertically layered agent architectures. In M. J. Wooldridge and N. R. Jennings, editors, *Proc. of Int. Workshop on Agent Theories, Architectures, and Languages (ATAL'94)*, pages 261–276, Amsterdam, The Netherlands, August 8–9 1995. Springer Berlin Heidelberg.

[108] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[109] T. J. Norman, N. R. Jennings, P. Faratin, and E. H. Mamdani. Designing and implementing a multi-agent architecture for business process management. In *Intelligent Agents III*, pages 261–275. Springer, 1997.

[110] I. Nunes, E. Cirilo, C. J. P. de Lucena, J. Sudeikat, C. Hahn, and J. J. Gomez-Sanz. A survey on the implementation of agent oriented specifications. In M.-P. Gleizes and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering X: 10th Int. Workshop, AOSE 2009, Budapest, Hungary, May 11-12, 2009, Revised Selected Papers*, volume 6038 of *LNCS*, pages 169–179. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[111] I. Nunes, C. J. de Lucena, and M. Luck. BDI4JADE: a BDI layer on top of JADE. In *Proc. of 9th Int. Workshop on Programming Multi-Agent Systems (PROMAS 2011)*, 2011.

[112] Oasis Committee. Web services business process execution language (WS-BPEL) version 2.0. Technical report, Oasis, 2007.

[113] P. D. O'Brien and M. E. Wiegand. Agent based process management: Applying intelligent agents to workflow. *The Knowledge Engineering Review*, 13(2):1–14, 1998.

[114] J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proc. of Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence*, pages 3–17, 2000.

[115] OMG. MDA guide version 1.0.1. Technical Report omg/2003-06-01, Object Management Group, June 2003.

[116] OMG. Agent metamodel and profile (AMP). Initial Submission ad/2009-08-04, Object Management Group, 2009.

[117] OMG. Business process modeling notation (BPMN) version 1.2. Specification formal/2009-01-03, Object Management Group, January 2009.

[118] OMG. Business process model and notation (BPMN) version 2.0.2. Specification formal/2013-12-09, Object Management Group, December 2013.

[119] OMG. Unified modeling language (UML) version 2.5. Specification formal/15-03-01, Object Management Group, June 2015.

[120] C. Ouyang, M. Dumas, S. Breutel, and A. H. M. ter Hofstede. Translating standard process models to BPEL. In E. Dubois and K. Pohl, editors, *CAiSE*, volume 4001 of *LNCS*, pages 417–432. Springer, 2006.

[121] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling. From business process models to process-oriented software systems. *ACM Transactions on Software Engineering and Methodology*, 19(1):1–37, August 2009.

[122] L. Padgham, J. Thangarajah, and M. Winikoff. Tool support for agent development using the prometheus methodology. In *Proc. of 5th Int. Conf. on Quality Software, 2005. (QSIC 2005).*, pages 383–388, 2005.

[123] L. Padgham, J. Thangarajah, and M. Winikoff. The prometheus design tool – a conference management system case study. In M. Luck and L. Padgham, editors, *Agent-Oriented Software Engineering VIII*, volume 4951 of *LNCS*, pages 197–211. Springer, 2008.

[124] L. Padgham and M. Winikoff. Prometheus: A methodology for developing intelligent agents. In F. Giunchiglia, J. Odell, and G. Weiß, editors, *Agent-Oriented Software Engineering III. Revised Papers and Invited Contributions of the Third Int. Workshop (AOSE 2002)*, volume 2585 of *LNCS*. Springer, 2002.

[125] L. Padgham, M. Winikoff, S. Deloach, and M. Cossentino. A unified graphical notation for AOSE. In M. Luck and J. J. Gomez-Sanz, editors, *Agent-Oriented Software Engineering IX*, pages 116–130. Springer, 2009.

[126] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Bonn: Institut für Instrumentelle Mathematik, 1962.

[127] G. Picard and M.-P. Gleizes. The ADELFE methodology. In F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors, *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, pages 157–175. Springer US, Boston, MA, 2004.

[128] A. Poggi, M. Tomaiuolo, and P. Turci. An agent-based service oriented architecture. In M. Baldoni, A. Boccalatte, F. D. Paoli, M. Martelli, and V. Mascardi, editors, *8th AI*IA/TABOO Joint Workshop "From Objects to Agents": Agents and Industry: Technological Applications of Software Agents (WOA)*, pages 157–165, Genova, Italy, September 2007.

[129] A. Pokahr and L. Braubach. Reusable interaction protocols for workflows. In *Workshop on Protocol based modelling of business interactions*, 2010.

[130] A. Pokahr and L. Braubach. The active components approach for distributed systems development. *IJPEDS*, 28(4):321–369, 2013.

[131] A. Pokahr, L. Braubach, C. Haubeck, and J. Ladiges. Programming BDI agents with pure Java. In J. P. Müller, M. Weyrich, and A. L. C. Bazzan, editors, *Proc. of 12th German Conference on Multiagent System Technologies (MATES 2014)*, volume 8732 of *LNAI*, pages 216–233, Stuttgart, Germany, September 23–25 2014. Springer International Publishing.

[132] A. Pokahr, L. Braubach, and K. Jander. Unifying agent and component concepts – jadex active components. In *Seventh German Conference on Multi-Agent System Technologies (MATES 2010)*, pages 100–112. Springer, 9 2010.

[133] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer, 2005.

[134] K. Raddatz, A.-D. Schmidt, A. Thiele, J. Chinnow, D. Grunewald, and S. Albayrak. Sensor-basierte Erkennung und Reaktion im häuslichen Umfeld. In *Ambient Assisted Living 2012, 5. Deutsche AAL Kongress*, Berlin, 2012. VDE.

[135] I. Raedts, M. Petkovic, Y. S. Usenko, J. M. E. M. van der Werf, J. F. Groote, and L. J. Somers. Transformation of BPMN models for behaviour analysis. In J. C. Augusto, J. Barjis, and U. Ultes-Nitsche, editors, *MSVVEIS*, pages 126–137. INSTICC PRESS, 2007.

[136] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In R. van Hoe, editor, *Agents Breaking Away, 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'96,*, volume 1038 of *LNCS*, pages 42–55, Eindhoven, The Netherlands, January 1996. Springer.

[137] J. C. Recker. BPMN modeling – who, where, how and why. *BPTrends*, 5(3):1–8, March 2008.

[138] J. C. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In T. Latour and M. Petit, editors, *CAiSE 2006 Workshop Proceedings – 11th Int. Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006)*, pages 521–532, 2006.

[139] G. Rimassa, D. Greenwood, and M. E. Kernland. The living systems technology suite: An autonomous middleware for autonomic computing. In *International Conference on Autonomic and Autonomous Systems (ICAS)*, pages 33–38, July 2006.

[140] J. E. Rumbaugh. Notation notes: Principles for choosing notation. *Journal of Object Oriented Programming*, 9(2):11–14, 1996.

[141] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[142] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst, and N. Mulyar. Workflow control-flow patterns, a revised view. Technical report, Workflow Patterns Initiative, 2006.

[143] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and P. Wohed. On the suitability of UML 2.0 activity diagrams for business process modelling. In *APCCM '06: Proc. of 3rd Asia-Pacific conference on Conceptual modelling*, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[144] G. Sacchi, A. Marando, E. Quarantotto, and G. Caire. *WADE Development Tutorial*. Telecom Italia S.p.A., 3.3 edition, November 2013.

[145] W. Sadiq and M. E. Orlowska. Applying graph reduction techniques for identifying structural conflicts in process models. In *CAiSE '99: Proc. of the 11th Int. Conf. on Advanced Information Systems Engineering*, pages 195–209, London, UK, 1999. Springer.

[146] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000. Elsevier Science Ltd., Oxford, UK, UK.

[147] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39:25–31, February 2006. IEEE Computer Society, Los Alamitos, CA, USA.

[148] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[149] R. Sesseler and S. Albayrak. JIAC IV – an open, scalable agent architecture for telecommunications applications. In *Proc. of 1st Int. NAISO Congress on Autonomous Intelligent Systems (ICAIS)*. ICSC Interdisciplinary Research, 2002.

[150] M. P. Singh and M. N. Huhns. Multiagent systems for workflow. *International Journal of Intelligent Systems in Accounting, Finance and Management*, 8:105–117, 1999.

[151] H. Smith and P. Fingar. *Business Process Management: The Third Wave*. Meghan-Kiffer Press, 2003.

[152] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1114, Dec 1980.

[153] P. S. Tan. Automated generation of JIAC agentbeans from BPMN diagrams. Diploma thesis, Technische Universität Berlin, November 2011.

[154] I. Trencansky and R. Cervenka. Agent modeling language (AML): A comprehensive approach to modeling MAS. *Informatica*, 29:391–400, 2005.

[155] A. Urzică, C. Tănase, and A. M. Florea. Bridging the gap between business experts and software agents: BPMN to AUML transformation. *UPB Scientific Bulletin, Series C: Electrical Engineering*, 72(4):49–64, 2010.

[156] W. M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[157] W. M. P. van der Aalst. Formalization and verification of event-driven process chains. Computing Science Reports 98/01, Eindhoven University of Technology, Eindhoven, 1998.

[158] W. M. P. van der Aalst, L. Aldred, M. Dumas, and A. H. M. ter Hofstede. Design and implementation of the YAWL system. In *Proc. of CAiSE*, 2004.

[159] W. M. P. van der Aalst and K. B. Lassen. Translating unstructured workflow processes to readable BPEL: Theory and implementation. *Information Software Technology*, 50(3):131–159, 2008. Butterworth-Heinemann, Newton, MA, USA.

[160] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30:245–275, 2003.

[161] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. Kluwer Academic Publishers. Hingham, MA, USA.

[162] M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In Pagham, Parkes, Müller, and Parsons, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 713–720, Estoril, Portugal, May 12–16 2008. IFAAMAS.

[163] J. M. Vidal, P. A. Buhler, and C. Stahl. Multiagent systems with workflows. *IEEE Internet Computing*, 8(1):76–82, January/February 2004.

[164] M. Voß. Orchestrating multi-agent systems with BPMN by implementing a process executing JIAC agent using the visual service design tool. Master thesis, Humboldt Universität Berlin, May 2014.

[165] T. Wagner and D. Moldt. Integrating agent actions and workflow operations. In J. P. Müller, W. Ketter, G. Kaminka, G. Wagner, and N. Bulling, editors, *Multiagent System Technologies: 13th German Conference, MATES 2015, Cottbus, Germany, September 28 - 30, 2015, Revised Selected Papers*, volume 9433 of *LNCS*, pages 61–78. Springer, 2015.

[166] C. D. Walton. Uniting agents and web services. In *Agentlink News, volume 18*, pages 26–28. AgentLink, 2005.

[167] S. Warwas, C. Hahn, and K. Fischer. A visual development environment for JADE. In D. S. S. Castelfranchi, editor, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 1349–1350, 2009.

[168] G. Weiß. Agent orientation in software engineering. *Knowledge Engineering Review*, 16(4):349–373, 2002.

[169] S. A. White. Introduction to BPMN. Technical report, BP Trends, May 2004. `http://www.bptrends.com/publicationfiles/07-04WPIntrotoBPMN-White.pdf` (2017/06/06).

[170] S. A. White. Process modeling notations and workflow patterns. Technical report, BP Trends, March 2004. `http://www.bptrends.com/publicationfiles/03-04WPNotationsandWorkflowPatterns-White.pdf` (2017/06/06).

[171] S. A. White. Using BPMN to model a BPEL process. Technical report, BP Trends, March 2005. `http://www.bptrends.com/publicationfiles/03-05WPMappingBPMNtoBPEL-White.pdf` (2017/06/06).

[172] M. Winikoff and L. Padgham. *Developing Intelligent Agent Systems: A Practical Guide*. Wiley and Sons, 2004.

[173] P. Wohed, M. Dumas, A. H. M. ter Hofstede, and N. Russell. On the suitability of BPMN for business process modelling. In *Proc. of 4th Int. Conf. on Business Process Management (BPM 2006)*, LNCS, pages 161–176. Springer, 2006.

[174] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell. Pattern-based analysis of BPMN. Technical Report BPM Report BPM-05-26, BPMcenter.org, 2005.

[175] P. Wohed, W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, and N. Russell. Pattern-based analysis of the control-flow perspective of UML activity diagrams. In L. M. L. Delcambre, C. Kop, H. C. Mayr, J. Mylopoulos, and O. Pastor, editors, *Conceptual Modeling – ER 2005: 24th International Conference on Conceptual Modeling, Klagenfurt, Austria, October 24-28, 2005. Proceedings*, volume 3716 of *LNCS*, pages 63–78. Springer, 2005.

[176] P. Y. Wong and J. Gibbons. A process semantics for BPMN. In *Proc. of 10th Int. Conf. on Formal Methods and Software Engineering*, ICFEM '08, pages 355–374. Springer, 2008.

[177] T. Woodley and S. Gagnon. BPM and SOA: Synergies and challenges. In A. H. H. Ngu, M. Kitsuregawa, E. J. Neuhold, J.-Y. Chung, and Q. Z. Sheng, editors, *Web Information Systems Engineering – WISE 2005*, volume 3806 of *LNCS*, pages 679–688. Springer, 2005.

[178] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[179] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, September 2000.

[180] Workflow Management Coalition. WfMC: Process definition language: XPDL 2.0. Specification TC-1025, WfMC, 2005.

[181] E. S.-K. Yu. *Modelling strategic relationships for process reengineering.* PhD thesis, University of Toronto, Toronto, Ont., Canada, 1995.

[182] E. S.-K. Yu. Social modeling and i*. In A. Borgida, V. Chaudhri, P. Giorgini, and E. Yu, editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *LNCS*, pages 99–121. Springer, 2009.

[183] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3):317–370, July 2003.

[184] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Multiagent systems as computational organisations: the Gaia methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent-Oriented Methodologies*, pages 136–171. Idea Group Publishing, 2005.

[185] J. Ziemann and J. Mendling. EPC-based modelling of BPEL processes: a pragmatic transformation approach. In *Proc. of 7th Int. Conf. "Modern Information Technology in the Innovation Processes of the Industrial Enterprises" (MITIP)*, Genova, Italy, September 2005.

[186] I. Zinnikus, C. Hahn, and K. Fischer. A model-driven, agent-based approach for the integration of services into a collaborative business process. In Pagham, Parkes, Müller, and Parsons, editors, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 241–248, Estoril, Portugal, May 12–16 2008. IFAAMAS.

[187] M. zur Muehlen and J. C. Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *20th International Conference on Advanced Information Systems Engineering*, volume 5074, pages 465–479, Montpellier, France, 2008.