Ein MDA-basierter Ansatz zur Entwicklung von Organic Computing Systemen

Holger Kasinger

TR 2005-08

(basierend auf der Diplomarbeit von Holger Kasinger)

Institut für Informatik

Universität Augsburg

April 2005

Institut für Informatik
Universität Augsburg
D-86135 Augsburg, Germany
www.informatik.uni-augsburg.de/vs

Kurzfassung

Die Komplexität heutiger IT-Systeme ist mit der Komplexität der IT-Systeme von vor 40 Jahren kaum mehr zu vergleichen. Schnellere Rechner, größere Speichermedien und höhere Übertragungsgeschwindigkeiten führen zu immer vielseitigeren und umfassenderen Applikationen, deren Komplexität noch weiter steigen wird. Die Anforderungen an die Administration dieser Systeme werden dabei so hoch sein, dass sie von keinem menschlichen Wesen mehr erfüllt werden können. Daher ist es das Ziel, die Administration den Systemen selber zu überlassen, um Eingriffe von außen vermeiden zu können und den Menschen zu entlasten. Solche selbstorganisierenden Systeme werden als Organic Computing Systeme (OCS) bezeichnet, welche durch so genannte Selbst-x Eigenschaften charakterisiert sind. Dazu gehören die Selbst-Konfiguration, die Selbst-Optimierung, die Selbst-Heilung und der Selbst-Schutz des Systems. Eine Schlüsselvoraussetzung für den erfolgreichen Einsatz von Organic Computing Systemen liegt dabei in ihrer Entwicklung. Aus diesem Grund wird eine konsistente und durchgängige Softwareentwicklungsmethodologie benötigt, welche es einem Entwickler ermöglicht, derartig komplexe Systeme zu entwerfen.

Diese Arbeit liefert einen Ansatz für einen modellbasierten Entwicklungsprozess als Grundlage für eine Softwareentwicklungsmethodologie für OCS. Der Ansatz basiert auf dem Framework der Model Driven Architecture, da dieses mit der Definition unterschiedlicher Abstraktionsgrade von Modellen und Modelltransformationen eine geeignete Basis für eine modellbasierte Entwicklung darstellt. Für den Ansatz werden zuerst die Prinzipien der Selbstorganisation an natürlichen, selbstorganisierenden Systemen untersucht, um ein Verständnis für Selbstorganisationsmechanismen zu erhalten. Mit der Analyse eines selbstorganisierenden Produktionsplanungs- und -kontrollsystems aus der Domäne der Fertigungssteuerung werden Anforderungen an die Architektur selbstorganisierender IT-Systeme gesammelt und durch die Untersuchung der technischen Erzeugung von Selbst-x Eigenschaften im Autonomic Computing um konzeptionelle Anforderungen ergänzt. Die gesammelten Anforderungen werden zur Erstellung eines Metamodells für OCS verwendet, in welches auch Konzepte aus Entwicklungsmethodologien für Multi-Agenten-Systeme einfließen. Das Metamodell dient als Fundament des Entwicklungsprozesses, dessen entstehende Modelle so weit wie möglich auf UML 2.0 basieren. Durch eine Evaluierung des Entwicklungsansatzes anhand einer Fallstudie werden abschließend Aspekte identifiziert, welche für die zukünftige Verwendung des erstellten Entwicklungsprozesses in einer Softwareentwicklungsmethodologie zu beachten sind.

Inhaltsverzeichnis

1. Einleitung

Als Gordon Moore im Jahr 1965 sein Gesetz [69] von der Verdopplung der Rechnerleistung alle 18 bis 24 Monaten aufstellte, wusste wohl keiner um die Beständigkeit seines Gesetzes – welches nach seiner aktuellen Schätzung auf der *International Solid-States Circuits Conference 2003* [7] wohl mindestens bis ins Jahr 2013 gültig ist – noch um dessen Konsequenzen. Durch das zusätzliche, starke Wachstum der Kapazität von Speichermedien und dem enormen Anstieg der Bandbreite von Kommunikationskanälen werden immer komplexere Computerarchitekturen entwickelt, welche von noch komplexeren Softwaresystemen gesteuert werden. Bestand eine Applikation im Jahr 1965 aus nur wenigen Zeilen Code, so wird beispielsweise die *Strategic Defense Initiative* [37] der USA wohl aus mehr als 100 Millionen Zeilen Code bestehen – Tendenz steigend.

In dieser Entwicklung liegt das Problem von zukünftigen IT-Systemen begraben. Nach heutigen Schätzungen wird die Anzahl der IT-Fachleute nicht ausreichen, um alle Systeme in Zukunft korrekt am Laufen zu halten. Selbst wenn diese Anzahl aufgebracht werden könnte, würde die Komplexität zukünftiger IT-Systeme über die menschliche Fähigkeit hinausgehen, diese zu administrieren (vgl. [55, S. 5]). Durch die unzähligen und verschachtelten Abhängigkeiten zwischen Systemen und den vielen, interagierenden Applikationen werden administrative Entscheidungen schneller benötigt, als sie jemals ein Mensch treffen könnte. Das Paradoxe an dieser Entwicklung ist jedoch, um die prognostizierte Komplexität in Zukunft für den Menschen beherrschbar zu machen, müssen bereits schon heute noch viel komplexere Systeme entwickelt werden, durch welche dieses Szenario vermieden werden kann.

Das Ziel wird es folglich sein, die Administration der IT-Systeme dem Menschen abzunehmen und dem System selber zur Aufgabe zu machen. Das Prinzip begründet sich auf der Funktionsweise des vegetativen Nervensystems des Menschen. Dieses kontrolliert Herzschlag, Blutzucker und -sauerstoffgehalt und reguliert Körpertemperatur, Lichteinfall und Hunger, ohne dass der Mensch ständig an diese Aufgaben denken muss. IT-Systeme so lebensähnlich – mit anderen Worten organisch – aufzubauen ist ein Ziel des *Organic Computing* (OC) [9]. Intuitiv zu bedienende Computer sollen sich dabei an den Menschen – auch an den IT-Laien – anpassen, lernfähig sein und bei Bedarf benötigte Dienste zur Verfügung stellen. Die Verhaltensweise dieser Computer soll dabei eher intelligenten Assistenten anstatt starren Befehlsempfängern entsprechen.

Organic Computing Systeme (OCS) basieren auf dem Zusammenspiel ihrer einzelnen Komponenten und können adaptiv auf innere und äußere Änderungen reagieren anstatt sich nur an hart codierte Vorgaben zu halten. Die Systeme übernehmen dabei weitere Eigenschaften aus der Natur, z. B. die von dissipativen Strukturen [88], autokatalytischen Zyklen [43] oder Ameisenkolonien (siehe 3.2). Das Verhalten von OCS resultiert zwar indirekt aus dem Verhalten ihrer Einzelkomponenten, lässt sich aus diesen jedoch nicht einfach berechnen. Diese Systeme haben durch ihre evolutiven Lernmechanismen, ihrem rein lokalen, sensorischen Kontextwissen und ihrer Vernetzung die Fähigkeit zur Selbstorganisation (SO). Ein *„organischer Computer“* wird daher definiert als *„ein selbstorganisierendes System, das sich den jeweiligen Umgebungsbedürfnissen dynamisch anpasst. Organische Computer sind selbst-konfigurierend, selbst-optimierend, selbst-heilend und selbst-schützend“* [47, S. 2]. Diese so genannten *Selbst-x Eigenschaften* werden in mancher Literatur um die Eigenschaft zur Selbst-Erklärung erweitert (siehe [70, S. 2]). Die Bedeutung dieser Eigenschaften ist in Tabelle 1.1 aufgeführt.

Eigenschaft	**Bedeutung**
Selbst-Konfiguration	Automatische (Re-)Konfiguration von Komponenten und Systemen nach abstrakten Regeln. Der Rest des Systems passt sich automatisch und nahtlos an die neuen Zustände an.
Selbst-Optimierung	Komponenten und Systeme suchen permanent nach Möglichkeiten, ihre eigene Performance und Effizienz zu verbessern.
Selbst-Heilung	Das System erkennt, diagnostiziert und repariert automatisch lokalisierte Soft- und Hardwareprobleme.
Selbst-Schutz	Das System schützt sich automatisch gegen böswillige Attacken oder Fehlfunktionen. Es verwendet frühzeitige Warnungen um systemweite Fehlfunktionen zu antizipieren und zu verhindern.
Selbst-Erklärung	Das System gibt dem Entwickler Einblick in seine Regeln und lässt Kontrolleingriffe zu.

Tabelle 1.1: Die Selbst-x Eigenschaften des Organic Computing

Trotz ihrer Inspiration durch Ideen aus der Natur basieren OCS aber vorerst auf traditioneller Siliziumtechnologie. Die lebensähnlichen Selbst-x Eigenschaften beziehen sich vielmehr auf den Aufbau des Gesamtsystems sowie das Zusammenspiel seiner Komponenten und sind eher vergleichbar mit den Eigenschaften des Autonomic Computing (AC) [55, 61]. Diese Initiative wird von IBM [4] seit 2001 als Leitprojekt propagiert, bei dem ein Managementsystem für Rechensysteme vorgeschlagen wird, welches auf Veränderungen in der Systemumgebung selbstständig reagiert, um die gewünschte Funktionalität des IT-Systems aufrecht zu erhalten. Die Zusammenfassung der Selbst-x Eigenschaften des AC – welche bis auf die Selbst-Erklärung dieselben sind wie im OC – wird von IBM als Selbstmanagement bezeichnet. Im Gegensatz zu OC ist der Anwendungsbereich des AC in erster Linie auf das Selbstmanagement von Rechenzentren und -netzen ausgelegt (siehe 5.1).

Die zukünftigen Anwendungsszenarien des OC sind laut [47] vielmehr *Smart Factories*, bei denen autonome Roboter durch spontane Vernetzung Föderationen zur Bearbeitung von Aufgaben bilden, *Smart Warehouses*, bei denen sich in intelligenten Warenhäusern Artikel erkennen und überwachen lassen, *Smart Networks/Smart Grids*, welche Netzwerke mit Selbst-x Eigenschaften versehen, oder *vertrauenswürdige Computer*, welche dem Menschen unter anderem die Integrität von Daten sicherstellt. Aktuelle Anwendungsbereiche liegen beispielsweise

in der Automobilindustrie, in welcher der ständig wachsende Anteil von elektronischen Komponenten sowohl in der Entwicklung als auch im Einsatz überschaubar und beherrschbar gehalten werden muss.

1.1 Ziel der Arbeit

Eine wichtige Schlüsselvoraussetzung für den zukünftigen, erfolgreichen Einsatz von OCS liegt in ihrer Entwicklung. Ohne einen geeigneten Ansatz für die Unterstützung der Entwicklung ist es so gut wie unmöglich, mit der Komplexität von OCS – insbesondere mit den systemweiten Selbst-x Eigenschaften – fertig zu werden. Aus diesem Grund wird eine konsistente und durchgängige Softwareentwicklungsmethodologie benötigt, welche es einem Entwickler ermöglicht, derartige Systeme zu entwerfen. Der Bedarf an einer solchen Methodologie wird mit der Zunahme der Systemkomplexität in den nächsten Jahren noch stark steigen.

Eine Softwareentwicklungsmethodologie ist typischerweise durch eine Modellierungssprache und einen Entwicklungsprozess charakterisiert. Die Modellierungssprache wird für die Beschreibung von Modellen verwendet und definiert eine spezifische Syntax (Notation) mit einer assoziierten Semantik der Modellelemente. Der Entwicklungsprozess definiert verschiedene Aktivitäten während der Entwicklung mit deren Ausführung und aktivitätsübergreifenden Beziehungen. Dazu gehören insbesondere Phasen für das Prozess- und Projektmanagement sowie der Qualitätssicherung. Jede definierte Aktivität resultiert dabei in einem oder mehreren Ergebnissen, wie Spezifikationsdokumenten, Modellen, Code, Test- und Performanceberichten etc., welche als Input für nachfolgende Aktivitäten dienen (vgl. [15, S. 1]).

Das Ziel der Arbeit soll es daher sein, einen Ansatz für einen modellbasierten Entwicklungsprozess als Grundlage für eine Softwareentwicklungsmethodologie für Organic Computing Systeme zu erstellen. Dabei soll auf die notwendigen Aktivitäten des Entwicklungsprozesses fokussiert werden und nicht auf das Prozess-, das Projektmanagement oder die Qualitätssicherung. Der Entwicklungsprozess soll auf dem Framework der Model Driven Architecture (MDA) [10] basieren, da dieses bereits mit der Definition von unterschiedlichen Abstraktionsgraden von Modellen und Modelltransformationen eine geeignete Grundlage darstellt.

1.2 Vorgehensweise

Nach der Beschreibung der Grundlagen für diese Arbeit (Kapitel 2) wird zuerst die Selbstorganisation in natürlichen Systemen untersucht (Kapitel 3). Dies soll einen Einblick in die Funktionsweise, die technische Umsetzung und die Verwendung von Selbstorganisationsmechanismen – insbesondere der Stigmergie – ermöglichen.

Im Anschluss daran wird aus der Domäne der Fertigungssteuerung ein selbstorganisierendes Produktionsplanungs- und -kontrollsystem als Beispiel für ein IT-System, welches auf der Stigmergie basiert, analysiert (Kapitel 4). Daraus werden Anforderungen an die Architektur von selbstorganisierenden, stigmergischen IT-Systemen identifiziert.

Daran schließt sich die Untersuchung einer möglichen, technischen Erzeugung von Selbst-x Eigenschaften für IT-Systeme an, um konzeptionelle Anforderungen an IT-Systeme, welche ein Selbst-x Verhalten aufweisen sollen, zu identifizieren (Kapitel 5). Diese Untersuchung wird auf Basis der Konzepte des Autonomic Computing durchgeführt werden.

Auf Grundlage der gesammelten Anforderungen wird anschließend ein Metamodell für Organic Computing Architekturen erstellt, welches als Fundament für den Entwicklungsprozess dient (Kapitel 6). Mit dem Entwicklungsprozess soll es möglich werden, das in Kapitel 4 analysierte IT-System nachzubilden und zusätzlich mit Selbst-x Eigenschaften auszustatten, um so ein stigmergisches Organic Computing System zu erhalten.

Die Evaluierung des Ansatzes soll zeigen, dass der erstellte Entwicklungsprozess nicht nur für die Entwicklung von stigmergischen OCS verwendet werden kann, sondern auch für nicht-stigmergische OCS gültig ist (Kapitel 7). Dazu wird ein weiteres, selbstorganisierendes Produktionsplanungs- und -kontrollsystem als Organic Computing System entwickelt, welches im Gegensatz zu dem ersten System auf Auktionen basiert.

Abschließend werden Aspekte identifiziert, welche für die zukünftige Verwendung des Entwicklungsprozesses in einer Softwareentwicklungsmethodologie zu beachten sind (Kapitel 8).

2. Grundlagen

Dieses Kapitel fasst zuerst den aktuellen Forschungsstand im Organic Computing zusammen. Dabei wird sowohl auf den Stand in der Softwareentwicklung als auch in der Hardwareentwicklung für Organic Computing Systeme eingegangen. Anschließend werden verschiedene Methodologien für die Entwicklung von Agentensystemen vorgestellt, auf die in dieser Arbeit Bezug genommen wird. Abschließend folgt die Beschreibung der Model Driven Architecture und ihrer Konzepte.

2.1 Aktueller Forschungsstand im Organic Computing

2.1.1 Hardwareentwicklung

Die Basis für OCS auf der Hardwareseite liefern so genannte Systems on Chip (SoC). Bei der SoC-Methode werden große Teile eines Chips aus bereits existierenden Funktionsblöcken zusammengebaut. Dabei können Systeme als *Intellectual Property*-Blöcke auf einem einzigen Chip implementiert werden, wohingegen sie noch vor einigen Jahren aus mehreren verteilten Einzelkomponenten aufgebaut waren. Für eine optimale Unterstützung der OC-Anforderungen müssen SoC-Architekturen sowohl Mikroprozessoren als auch rekonfigurierbare Hardwarekomponenten enthalten.

Wesentliche Voraussetzungen für solche SoC-Architekturen liegen unter anderem in der mehrfädigen Prozessorarchitektur [94]. Dabei wird eine überlappte, parallele Ausführung von Befehlen mehrerer Threads in einer Prozessor-Pipeline ermöglicht. Diese Prozessortechnik liefert unter anderem die Basis dafür, dass so genannte Helper Threads gleichzeitig zu einer Anwendung ablaufen können und mittels der *Guaranteed Percentage Scheduling*-Technik [65] einen festen Prozentsatz der Rechenleistung erhalten. Ein Helper Thread läuft dabei mit niedriger Priorität in einem eigenen Thread-Slot konkurrent zur Anwendung und implementiert einen so genannten Autonomic Manager (AM) als organischen Supervisor. Der AM beobachtet die Echtzeitanwendung (Monitoring), durchläuft eine Schleife (Analyze, Plan) und passt die Anwendung gemäß der aktuellen Bedürfnisse und organischen Prinzipien an (Execute), ohne die Echtzeitbedingungen dieser oder anderer Anwendungen zu verletzen.

Beispiele für eine solche SoC-Architektur finden sich in CARUSO [22]. Die Autonomic Manager bieten durch ihre Vernetzung und ihre Kenntnisse des Zustands der lokalen Anwen-

dung die Grundlage für selbstorganisierende Algorithmen auf Middlewareebene. Auf dieser Ebene wird basierend auf den Informationen entschieden, ob Techniken für die Selbst-x Eigenschaften angewandt werden müssen.

2.1.2 Softwareentwicklung

Ein Ziel des OC ist die ingenieurtechnische Beherrschung von OCS. Traditionelle SE-Methoden sind streng hierarchisch und folgen einem Top-Down-Ansatz. Das zugrunde liegende Paradigma dieser Methode ist die komplette Verfeinerung der Spezifikation in detaillierte Entwürfe um eine vollständige Kontrolle des resultierenden Systems zu erhalten. Der Entwickler muss sich infolge dessen allen möglichen Systemzuständen im Vornherein bewusst sein. Mit der Zulassung von emergenten und selbstorganisierenden Systemen wird dieser strikte Top-Down-Entwurf verlassen. Es müssen Systemzustände erreicht werden können, welche vom Entwickler nicht im Vornherein geplant worden sind. Hier liegt aber ein fundamentaler Widerspruch zwischen Top-Down-Kontrolle und kreativem Bottom-Up-Verhalten. Es ist in der Forschungsgemeinschaft noch nicht klar, wie diese gegenläufigen Tendenzen miteinander vereinbart werden können. Ansätze liegen in der Constraint Propagation, dem Einsatz von Assertions und so genannten Observer/Controller-Architekturen (vgl. [70]).

Nachdem eingebettete OCS mehr und mehr sicherheitskritisch werden, sind zurzeit Techniken in Entwicklung, um einzuhaltende Randbedingungen mittels Zusicherungen (Assertions) zu erzwingen [102, 84, 83, 85, 82]. Assertions (nach B. Meyer [68]) können auf diesem Weg für das Monitoring von bestimmten Variablen verwendet werden. Die Beschränkung des emergenten Verhaltens von OCS wird ausschlaggebend für deren technischen Einsatz sein. Daher spielen Constraints eine wichtige Rolle für die Limitierung von lernenden und selbstorganisierenden Systemen. Constraint-Verletzungen führen zu Warnungen und können so für mögliche korrigierende Eingriffe genutzt werden.

Observer/Controller-Architekturen nehmen Anleihen beim makroskopischen Aufbau des Gehirns vor. Hier existieren neben den direkten Reizreaktionsmechanismen zwischengeschaltete Überwachungsapparate, welche den sensorischen Input filtern und die vom bewussten Gehirn vorgeschlagenen Aktionen einer Bewertung unterziehen. Diese Funktion wird vom limbischen System ausgeführt, welches dem bewussten und logischen Denken die emotionale Färbung aufprägt. Die technische Basis für autonome Observer/Controller ist in einzelnen der heute in Entwicklung befindlichen eingebetteten Prozessoren, wie in 2.1.1 beschrieben, bereits vorgesehen.

Trotz dieser unterschiedlichen Ansätze existiert bis zum jetzigen Zeitpunkt noch keine vollständige Softwareentwicklungsmethodologie für Organic Computing Systeme.

2.2 Entwicklungsmethodologien für Multi-Agenten-Systeme

Zur Erstellung eines Entwicklungsprozesses für OCS werden Konzepte aus Softwareentwicklungsmethodologien ähnlicher, bereits bestehender Technologien verwendet. Betrachtet man die Konzepte des Organic Computing, so fallen gewisse Ähnlichkeiten zur Agententechnologie auf. Ein Agent ist durch folgende Eigenschaften charakterisiert (vgl. [107]):

Autonomie Agenten besitzen einen Zustand, auf welchen andere Agenten nicht zugreifen können, und treffen basierend darauf Entscheidungen über ihr Handeln, ohne den direkten Eingriff von Menschen oder anderen Agenten.

Reaktivität Agenten befinden sich in einer Umgebung, welche die physische Welt, ein Benutzer über die graphische Benutzerschnittstelle, eine Ansammlung von anderen Agenten, das Internet oder vielleicht einige von diesen kombiniert sein können. Sie können diese Umgebung durch die Verwendung von Sensoren wahrnehmen und rechtzeitig auf Änderungen reagieren.

Proaktivität Agenten reagieren nicht nur auf ihre Umgebung, sie sind auch dazu in der Lage, zielgerichtetes Verhalten aufzuzeigen, indem sie die Initiative ergreifen.

Soziale Fähigkeit Agenten interagieren mit anderen Agenten über eine Agentenkommunikationssprache und haben typischerweise die Fähigkeit, soziale Aktivitäten zu übernehmen, um ihre Ziele zu erreichen.

Agentensysteme haben somit einige Eigenschaften mit Organic Computing Systemen gemeinsam, wobei letztere durch ihre expliziten Selbst-x Eigenschaften charakterisiert sind. Aufgrund der längeren und größeren Verbreitung von Agentensystemen im Gegensatz zu OCS existieren bereits viele Entwicklungsmethodologien. Nach [14] werden die Methodologien in drei verschiedene Kategorien unterteilt: Knowledge engineering, Agenten-orientierte und Objekt-orientierte Ansätze.

Die wichtigsten Methodologien der ersten Kategorie sind CommonKADS [92], Co-MoMAS [50] und MAS-CommonKADS [58]. Ein bemerkter Mangel der Knowledge engineering Methodologien war jedoch, dass sie nicht zur expliziten Entwicklung von Agentensystemen entwickelt wurden und daher zumeist begrenzte Fähigkeiten hatten, agenten-spezifische Fähigkeiten zu unterstützen (vgl. [14]). Dieser Nachteil wird von den agenten-orientierten Methodologien behoben, vornehmlich von Gaia [108] und ihrer Erweiterung ROADMAP (Role Orientated Analysis and Design for Multi-Agent Programming) [60] sowie von SODA (Societies in Open and Distributed Agent spaces) [81] und ADELFE [18, 19]. Um die Agententechnologie für die Industrie attraktiv zu machen, wurden Methodologien und Notationen entwickelt, welche objekt-orientierte Ansätze verwenden. Zu dieser Kategorie gehören MESSAGE (Methodology for Engineering Systems of Software Agents) [2, 28], Tropos [52, 71, 12], Prometheus [86, 1], MaSE (Multiagent Systems Engineering) [106, 105, 35] und PASSI (Process for Agent Societies Specification and Implementation) [32]. Speziell für diese Methodologien wurde als graphische Notationsunterstützung Agent UML (AUML) [15] entwickelt.

2.3 Model Driven Architecture

Die Model Driven Architecture ist ein von der Object Management Group (OMG) [11] im Jahr 2001 definiertes Framework für modelgesteuerte Softwareentwicklung. Ziel der MDA ist es, Softwaresysteme effizienter und qualitativ hochwertiger zu erstellen. Bei der Spezifikation von Softwaresystemen mittels der MDA wird die Systemfunktionalität von Implementierungsdetails einer bestimmten Technologieplattform getrennt. Dies wird durch eine Fokussierung auf konstruktive Modelle in allen Phasen der Systementwicklung erreicht. Diese Modelle dienen nicht nur der abstrakten Beschreibung von Aspekten des zu realisierenden Systems, sondern werden zur automatischen Gewinnung weiterer Modelle desselben Systems oder zur Erzeugung von Komponenten eines Systems verwendet. Die Grundlage für dieses Paradigma bildet die Transformation eines plattformunabhängigen Modells höheren Abstraktionsgrades in ein oder mehrere plattformspezifische Modelle niedrigeren Abstraktionsgrades.

Die MDA ermöglicht zudem die Spezifikation von Plattformen, die Auswahl einer oder mehreren Plattformen zur Realisierung des Softwaresystems und die Transformation der Systemspezifikation in eine plattformspezifische Spezifikation. Dadurch können Spezifikationen und ihre Modelle für unterschiedliche Anforderungen und Problemstellungen wieder verwendet werden. Das Framework basiert auf Modellierungsstandards der OMG (MOF [74], XMI [80], UML [77] und CWM [73]) und schließt einige UML-Profile, wie z. B. Enterprise Computing [79] oder Real-Time Computing [78], mit ein.

2.3.1 Adressierte Problemfelder der Softwareentwicklung

Der messbare Fortschritt in der Softwareentwicklung ist abhängig von der Möglichkeit, komplexere und größere Systeme mit geringerem finanziellen oder zeitlichen Einsatz entwickeln zu können. Dabei trifft die SE auf einige Problemfelder, welche von der MDA gelöst werden:

Produktivität Die Artefakte der verschiedenen Phasen eines typischen Softwareentwicklungsprozesses sind unabhängig von der inkrementellen oder iterativen Version des Prozesses zumeist nicht mehr als „Papier“. Da sie zu Beginn jeder neuen Phase immer mehr an Wert verlieren, verzichten Entwicklungsprozesse wie Extreme Programming (XP) [16] oder Agile Software Development [30] bereits fast gänzlich auf Entwicklungsdokumente.

Portabilität Durch das permanente Entstehen neuer und besserer Technologien sind Unternehmen aus Wettbewerbsgründen ständig gezwungen, ihre bestehende Software zu portieren oder an eine neue Version anzupassen. Investitionen in frühere Technologien verlieren dadurch sehr schnell an Wert.

Interoperabilität Existierende Softwaresysteme sind häufig als monolithische Insellösungen entwickelt worden und laufen in hermetischer Isolation. Durch den parallelen Einsatz alter und neuer Technologien ist eine Kommunikation zwischen verschiedensten Systemen jedoch zwingend notwendig.

Pflege und Dokumentation Da die Implementierung eines Systems allgemein für wichtiger erachtet wird, erfolgt die verlangsamende Dokumentation aus Zeit- und Kostengründen oftmals erst am Ende eines Entwicklungsprozesses. Die Folgen sind nicht aktualisierte Dokumentationen niedriger Qualität.

2.3.2 Softwareentwicklungsprozess

Der Softwareentwicklungsprozess der MDA liefert drei wesentliche, formale Modelle als Artefakte: Das *Computational Independent Model* (CIM), das *Platform Independent Model* (PIM) und das *Platform Specific Model* (PSM). Diese Modelle bilden eine Architekturbeschreibung des existierenden oder zu entwickelnden Systems von unterschiedlichen Sichtpunkten (Viewpoints) bzw. Abstraktionsgraden aus und können im Gegensatz zu traditionellen Artefakten von Tools verarbeitet werden. Abbildung 2.1 stellt den Zusammenhang dieser Modelle dar, welche im Folgenden näher erläutert werden.

2.3.2.1 Computational Independent Model

Das CIM dient der Beschreibung eines Unternehmens mit seinen Geschäftsprozessen und verwendet ein Vokabular, welches den Anwendern aus der entsprechenden Domäne vertraut

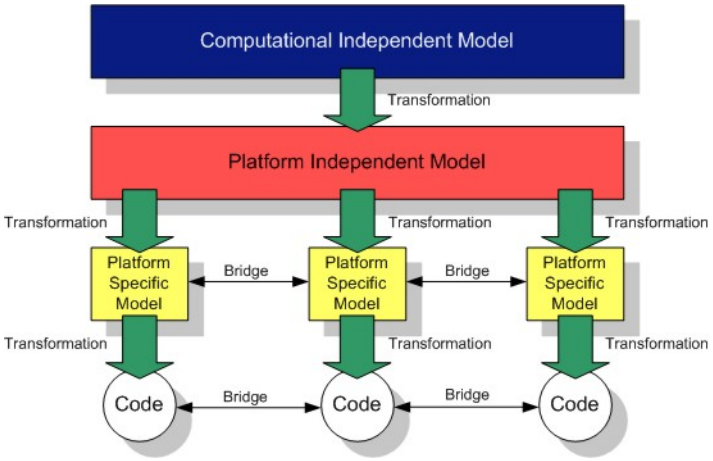

Abbildung 2.1: Der MDA-Prozess

ist. Dieses Vokabular ist nicht nur nützlich für das Problemverständnis, sondern dient auch als Grundwortschatz für die weiteren Modelle. Ein CIM sagt nicht notwendigerweise etwas über verwendete Softwaresysteme im Unternehmen aus, sondern ist selber softwareunabhängig und wird daher vereinzelt auch als *Domain* oder *Business Model* bezeichnet. Sobald einige oder alle Teile eines CIM von einem Softwaresystem unterstützt werden sollen, muss dafür ein spezifisches Softwaremodell entwickelt werden.

Das CIM modelliert solch ein Softwaresystem in seiner aktuellen oder zukünftigen Einsatzumgebung. Es beschreibt die abstrakte Verwendung des Systems, die Anforderungen an seine Umgebung sowie den Nutzen, welchen die Umgebung aus dem System zieht, ohne Details der Systemstrukturen preiszugeben. Die in einem CIM modellierten Anforderungen sollten nachvollziehbar im PIM und PSM umgesetzt werden und von dort auch wieder zurückverfolgt werden können (*Traceability*). Ein CIM kann mehrere, vorwiegend in UML modellierte, Modelle enthalten, welche mehr oder weniger Details der Domäne offen legen bzw. auf verschiedene Belange des softwareunabhängigen Viewpoint fokussieren.

2.3.2.2 Platform Independent Model

Das PIM modelliert ein System im Gegensatz zum CIM aus einem plattformunabhängigen Viewpoint, so dass das System zwar beschrieben wird, jedoch keine Details für die Verwendung auf einer bestimmten Plattform preisgibt. Es zeigt genau denjenigen Teil einer kompletten Systemspezifikation, welcher sich bei einem Wechsel auf eine andere Plattform nicht verändert. Das PIM soll dem Grundsatz der MDA nach möglichst automatisch durch Transformation aus dem CIM erzeugt werden können und dieses verfeinern. Zum heutigen Zeitpunkt ist diese Transformation jedoch noch nicht vollständig möglich.

Um eine Plattformunabhängigkeit zu erreichen, bieten sich klassische Vorgehensweisen, wie die Basierung auf einer virtuellen Maschine (vgl. [76]) oder die Nutzung einer generischen Plattform (vgl. [21]), an. Dabei kann auf eine allgemein anerkannte Grundmenge von Elementen und Diensten, so genannte Pervasive Services [76], Bezug genommen werden (siehe [21]). Allerdings fehlt für diese Dienste bisher ein eindeutiger OMG-Standard für plattformunabhängige Modelle. Das PIM als Kernstück des MDA-Prozesses kann ebenfalls aus mehreren (UML-)Modellen bestehen, welche im Vergleich zu den Modellen aus dem CIM einen höheren Detaillierungsgrad besitzen.

2.3.2.3 Platform Specific Model

Ein PSM zeigt die Spezifikation eines PIM mit genauen Details über die Realisierung auf einer oder mehreren konkreten Soft- oder Hardwareplattformen. Durch diesen plattformabhängigen Viewpoint kann ein PSM anschließend direkt in maschinenlesbaren Code transformiert werden und als fertiges Softwaresystem in Betrieb gehen. Für jede spezifische Technologieplattform wird dabei ein eigenes PSM generiert. Da sich die meisten Systeme heutzutage über mehrere Technologien erstrecken, wird für jede Plattform eine separates PSM erzeugt. Aufgrund dessen existieren oft mehrere PSM zu einem einzigen PIM.

Die dadurch notwendig gewordene Kommunikation zwischen den einzelnen PSM bzw. Plattformen wird von der MDA mit der Generierung von so genannten Bridges sichergestellt. Dabei werden Konzepte einer Plattform in verwendete Konzepte anderer Plattformen transformiert wodurch die Interoperabilität gewährleistet wird.

Das aus einem PIM durch Transformation abgeleitetes PSM stellt zwar eine Implementierung des abstrakteren Modells dar, kann aber noch von einer Reihe von Implementierungsdetails abstrahieren und als Ausgangspunkt für weitere Transformationen in verfeinerte PSM dienen. Aus der Sicht der verfeinerten PSM würde das ursprüngliche PSM hingegen die Rolle eines PIM einnehmen. Diese Verfeinerung kann beliebig weit fortgesetzt werden, wobei die Effizienz im Auge behalten werden muss.

2.3.2.4 Code

Nachdem ein PSM sehr technologienah modelliert wird, ist diese Transformation in maschinenlesbaren Code relativ unkompliziert. Auch die auf PSM-Ebene eingeführten Bridges werden auf die Code-Ebene mit übernommen.

2.3.2.5 Transformationen

Die Transformationen zwischen den unterschiedlichen Modellen bilden das Systemkonstruktionsparadigma der MDA und ermöglichen durch die maschinelle Verarbeitung der Modelle die Unterstützung des Entwicklungsprozesses durch Tools. Die MDA gibt unabhängig von den zu transformierenden Modellen, der Art und Weise der Transformation und den eingesetzten Modellierungssprachen ein generelles Muster für die Transformation vor (siehe Abbildung 2.2).

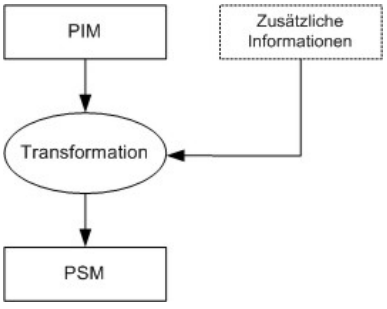

Abbildung 2.2: Modeltransformationsmuster der MDA [75]

Ausgangspunkt für den Transformationsprozess sind immer ein gegenüber dem Transformationsergebnis plattformunabhängiges Modell und zusätzliche, ergänzende Informationen, wie beispielsweise ein Platform Model (PM). Ein PM ist eine technische Spezifikation, welche die Struktur der Bestandteile einer Plattform und die durch die Plattform angebotenen

Dienste präzise beschreibt. Das Transformationsergebnis wird – bezogen auf die ergänzenden Plattforminformationen – als plattformspezifisch angesehen. Transformationen sind dabei folgendermaßen aufgebaut [101, S. 24]:

- *„Eine Transformation ist die automatische Generierung eines Zielmodells aus einem Quellmodell, gemäß einer Transformationsdefinition.“*
- *„Eine Transformationsdefinition ist eine Menge von Transformationsregeln welche gemeinsam beschreiben, wie ein Modell in der Quellsprache in ein Modell in der Zielsprache transformiert werden kann.“*
- *„Eine Transformationsregel ist eine Beschreibung dessen, wie ein oder mehrere Konstrukte aus der Quellsprache in ein oder mehrere Konstrukte in der Zielsprache transformiert werden können.“*

3. Selbstorganisation als natürliches Vorbild

Selbstorganisierende IT-Systeme – insbesondere Organic Computing Systeme – beziehen ihre Konzepte aus natürlichen, selbstorganisierenden Systemen. Dieses Kapitel soll das Verständnis für selbstorganisierende Systeme vertiefen und deren Charakteristika, theoretischen Anforderungen und wesentlichsten Mechanismen beschreiben, welche von diesen Systemen zur Selbstorganisation genutzt werden. Einer dieser Mechanismen – die Stigmergie – wird dabei am Beispiel der Nahrungssuche in Ameisenkolonien näher untersucht. Eine wichtige Voraussetzung für die Nutzung der Selbstorganisation spielt die technische Umsetzung der Mechanismen in Algorithmen, welche von Computern verstanden werden können. Daher wird in diesem Kapitel zusätzlich die technische Umsetzung der Stigmergie durch Ameisenalgorithmen genau erläutert und den daraus gewonnenen Nutzen für die Lösung von verschiedenen kombinatorischen Optimierungsproblemen beschrieben.

3.1 Der Begriff der Selbstorganisation

Die Wissenschaft der letzten vier Jahrhunderte, insbesondere die der Physik, beruhte in erster Linie auf Newtons Prinzipien der Mechanik [72]. Darin konnte jedes auftretende Phänomen letztlich auf eine Ansammlung von Atomen oder Körpern reduziert werden, deren Bewegungen von deterministischen Naturgesetzen geregelt wurden. Die neuere Wissenschaft hingegen ist nach und nach zu dem Ergebnis gekommen, dass es mit diesem Ansatz nicht möglich sein wird, alle auftretenden Phänomene vollständig zu erklären. Seit Mitte des letzten Jahrhunderts haben Forscher aus verschiedenen Bereichen daher mit der Untersuchung von Phänomenen begonnen, welche von inhärenter Kreativität, von spontanem Auftreten neuartiger Strukturen oder von autonomen Anpassungen an eine sich verändernde Umgebung geregelt zu werden scheinen. Die verschiedenen Beobachtungen und die daraus gewonnenen Ergebnisse begannen langsam, sich zu einem neuen Ansatz zusammenzufügen – der Wissenschaft der Selbstorganisation.

3.1.1 Charakteristika selbstorganisierender Systeme

Die unterschiedlichen Untersuchungen der letzten fünfzig Jahre haben eine Vielzahl von fundamentalen Eigenschaften und Kennzeichen hervorgebracht, welche selbstorganisierende

Systeme von den eher traditionellen Systemen unterscheidet. Heylighen identifiziert in [54] folgende Charakteristika selbstorganisierender Systeme:

Globales Verhalten aus lokalen Interaktionen Das globale Verhalten selbstorganisierender Systeme ist nicht nur die Vereinigung der Verhalten der einzelnen Systemkomponenten. Die Interaktionen zwischen den Systemkomponenten erzeugen vielmehr ein größeres System als es alle einzelnen Komponenten könnten.

Verteilte Kontrolle Die Kontrolle in selbstorganisierenden Systemen ist verteilt und nicht zentralisiert. Jede Systemkomponente ist für sich selbst verantwortlich und kontrolliert sich selber. Die verschiedenen Komponenten haben kein Wissen über die Ziele und Zustände anderer Komponenten.

Robustheit und Belastbarkeit Selbstorganisierende Systeme sind so aufgebaut, dass der Ausfall oder das Entfernen von Komponenten das gesamte System nicht wesentlich beeinträchtigt. Die Systeme sind daher unanfällig gegenüber Fehlern und haben die Fähigkeit, sich selbst wiederherzustellen.

Nichtlinearität und Feedback Der Zusammenhang zwischen Ursache und Wirkung ist bei selbstorganisierenden Systemen im Gegensatz zu traditionellen Systemen nicht linear, d. h. kleine Ursachen können eine große Wirkung besitzen und umgekehrt. Die Reaktion auf ein Ereignis kann zudem unterschiedlich sein und führt nicht immer zum gleichen Ergebnis.

Organisatorische Geschlossenheit, Hierarchie und Emergenz Selbstorganisierende Systeme bilden eine geschlossene, geordnete Organisation um eine Funktion zu erfüllen. Daraus resultieren emergente Eigenschaften. Emergenz ist generell als ein Prozess verstanden, welcher zur Entstehung einer Struktur führt, die nicht direkt von den existierenden Bedingungen und den momentanen Kräften, die eine System kontrollieren, beschrieben werden kann (vgl. [34]).

Verzweigungen und Symmetriebrechung Für selbstorganisierende Systeme existieren mehrere stabile Zustände, in welche sie übergehen können. Es ist in gewisser Weise jedoch unvorhersagbar, in welchen Zustand das System aufgrund von welchen Einflüssen übergeht.

Weit vom Gleichgewicht entfernte Dynamiken Durch das Hinzufügen und Entfernen von Komponenten sind selbstorganisierende Systeme ständig in Bewegung und beeinflussen ihre Umgebung, wie sie auch von dieser beeinflusst werden.

3.1.2 Theoretische Anforderungen an selbstorganisierende Systeme

Um die genannten Charakteristika aufweisen zu können und selbstorganisierendes Verhalten möglich zu machen, müssen selbstorganisierende Systeme wenigstens vier theoretische Anforderungen erfüllen. Die Anforderungen wurden von Prigogine und seinen Kollegen mathematisch abgeleitet (siehe [48]):

1. Mindestens zwei Komponenten im System müssen *gegenseitig kausal* sein. Ein System besitzt gegenseitige Kausalität, wenn mindestens zwei der Komponenten eine zirkuläre Beziehung haben, bei welcher jede die andere beeinflusst.

2. Mindestens eine Komponente im System muss eine *Autocatalyse* besitzen. Ein System besitzt eine Autocatalyse, wenn mindestens eine der Komponenten kausal von einer anderen Komponente beeinflusst wird, was zu einer Intensivierung der Komponente führt.

3. Das System muss unter *weit vom Gleichgewicht entfernten Bedingungen* operieren. Ein System ist als „weit vom Gleichgewicht entfernt“ definiert, wenn es eine große Menge Energie von außerhalb des Systems aufnimmt und diese Energie dazu verwendet, seine eigenen Strukturen zu erneuern (Autopoesie) und die anwachsende Unordnung (Entropie) mehr in die Umgebung zurückzugeben als sie zu speichern.

4. Um *morphogenetische Veränderungen* zu besitzen, muss mindestens eine der Komponenten des Systems offen für externe, zufällige Änderungen außerhalb des Systems sein.

3.1.3 Selbstorganisationsmechanismen

Solche selbstorganisierenden Systeme können nach Di Marzo Serugendo [40] in drei verschiedene Kategorien aufgeteilt werden: Physikalische Systeme, lebende Systeme und soziale Systeme. Alle drei Kategorien besitzen verschiedene Selbstorganisationsmechanismen, deren wichtigsten im Folgenden vorgestellt werden.

3.1.3.1 Magnetisierung

Ein wichtiger Mechanismus der Selbstorganisation physikalischer Systeme ist die Magnetisierung, welche mittlerweile ausführlich untersucht wurde. Ein Stück möglicherweise magnetischen Materials, z. B. Eisen, besteht aus einer Vielzahl von kleinen Magneten, so genannten Spins. Jeder Spin hat seine eigene, geographische Orientierung entsprechend der Richtung seines magnetischen Feldes (siehe linke Seite der Abbildung 3.1).

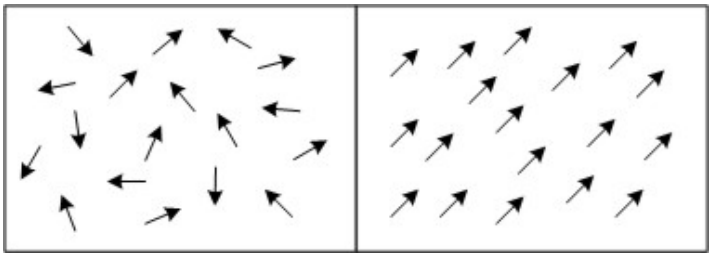

Abbildung 3.1: Anordnung der Spins in einem Metallstück: Ungeordnet und geordnet

Die Ausrichtungen der Spins sind aufgrund der zufälligen Bewegungen der Moleküle generell unterschiedlich, so dass sich ihre magnetischen Felder in Summe gegenseitig aufheben. Je höher jedoch die Temperatur des Materials ansteigt, desto stärker werden die zufälligen Bewegungen der Moleküle. Diese molekularen Bewegungen beeinflussen die Spins, wodurch es für sie immer schwieriger wird, sich geordnet auszurichten. Sobald die Temperatur einen gewissen Punkt erreicht hat, richten sich die Spins spontan von selber aus, so dass sie alle in dieselbe Richtung zeigen (siehe rechte Seite der Abbildung 3.1). Anstatt sich gegenseitig aufzuheben, ergänzen sich nun die magnetischen Felder zu einem einzigen, großen Magneten. Der Grund dafür ist, dass Spins eine geordnete Ausrichtung bevorzugen, da sich Spins mit entgegen gesetzten Ausrichtungen abstoßen. Die Abstoßung ist vergleichbar mit der Abstoßung beim aneinander halten der Nordpole von zwei Magneten. Die Magnetisierung ist

eine deutliche Form der Selbstorganisation, da die Ausrichtung der Spins variabel ist und von der lokalen Nachbarschaft abhängt. Ein ähnliches Phänomen dieser Art kann bei der Kristallisation flüssiger Materialien beobachtet werden.

3.1.3.2 Neuronale Netzwerke

Neuronale Netzwerke nach Kohonen [62] – auch bekannt als Self Organizing Maps (SOM) – sind unüberwachte Lernalgorithmen, welche Strukturen in Daten aufdecken und für Anwendungen mit Clustering geeignet sind. Sie haben ihre Inspiration aus den Gehirnzellen (lebende Systeme), welche abhängig vom Ort eines wahrgenommenen Objekts aktiviert werden. Ein Kohonennetz besteht aus zwei Schichten von Neuronen, einer Eingabe- und einer Ausgabeschicht – auch als Kohonenschicht bezeichnet –, in der eine topologische Karte und damit eine zweidimensionale Visualisierung der Daten entsteht. Die Anzahl der Neuronen in der Eingabeschicht ergibt sich aus der Dimensionalität der Eingabedaten. Die Struktur der Ausgabeschicht wird vom Anwender festgelegt.

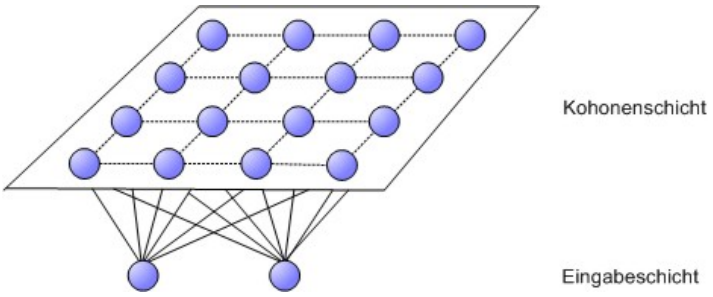

Abbildung 3.2: Neuronales Netzwerk nach T. Kohonen

Abbildung 3.2 zeigt ein Kohonennetzwerk mit zweidimensionaler Anordnung der Ausgabeschicht. Jedes Neuron der Eingabeschicht ist mit allen Neuronen der Ausgabeschicht verbunden und jedem Kohonenneuron ist ein Gewichtsvektor von der Größe des Eingabevektors zugeordnet (nicht eingezeichnet). Zudem sind die Kohonenneuronen untereinander verbunden. Ein Ähnlichkeitsmaß, wie z. B. die euklidische Distanz, vergleicht die Eingabevektoren mit den Gewichtsvektoren. Das Neuron mit der geringsten Distanz oder der höchsten Ähnlichkeit zum Eingabemuster gewinnt und erhält die gesamte Aktivierung. Die Gewichtungen des Gewinner-Neurons zu der Eingabeschicht werden so modifiziert, dass die Ähnlichkeit weiter steigt. Im Gegensatz zu anderen Clustering-Verfahren, die bis hierhin ähnlich funktionieren, verändern sich im Kohonennetzwerk auch die Gewichtsvektoren der Nachbarneuronen des Gewinners, um eine topologische Struktur zu erzeugen. Je näher ein benachbartes Neuron dem aktivierten Neuron ist, desto stärker wird sein Gewichtsfaktor adaptiert. Die Vektoren sehr naher Neuronen werden somit immer in ähnliche Richtungen verschoben, wodurch Cluster mit Abbildungen in ähnliche Muster entstehen.

3.1.3.3 Stigmergie

Der Begriff Stigmergie (Zusammensetzung aus dem Griechischen: *stigma* = Zeichen und *ergon* = Arbeit) wurde zum ersten Mal von Grassé [51] im Jahre 1959 verwendet und deutet an, dass Aktivitäten von gewissen Einheiten über äußerliche Zeichen ausgelöst werden. Der französische Biologe untersuchte das Verhalten einer Termitenart während dem Nestbau und bemerkte, dass das Verhalten der Arbeitertermiten während dem Bauprozess von der Struktur des Nestbaus selber beeinflusst wurde. Stigmergie beschreibt eine Form von asynchroner

Interaktion und dem Informationsaustausch zwischen Insekten (soziale Systeme) über eine aktive Umgebung. Untersuchungen biologischer Insektenvölker, wie z. B. Bienen-, Wespen-, oder Ameisenkolonien, zeigen, dass sich diese Tiere selbst organisieren, indem sie ein dissipatives Feld in ihrer Umgebung erstellen. Die Umgebung wird auf diese Weise als Schreibmedium verwendet, in dem vergangene Verhaltenseffekte eingetragen werden um zukünftiges Verhalten zu beeinflussen. Dieser Mechanismus definiert den so genannten selbstkatalytischen Prozess, das bedeutet, je öfter ein Prozess vorkommt, desto größer ist seine Chance in Zukunft erneut vorzukommen. Dieser Mechanismus zeigt allgemein, wie leicht Systeme einen großen Bereich von komplexem, koordiniertem Verhalten kreieren können, indem sie den Einfluss ihrer Umgebung ausnutzen.

3.2 Nahrungssuche in Ameisenkolonien

Ameisenkolonien sind ein geeignetes Beispiel, in dem das Prinzip der Stigmergie deutlich wird. Beobachtet man in der Natur Ameisen, welche Futter von einer Futterquelle zu einem Ameisenhügel transportieren, so ist deutlich zu erkennen, dass sie sich zwischen diesen beiden Orten auf einem festgelegten Weg, den Ameisenstraßen, zu bewegen scheinen. Diese Straßen verlaufen beinahe direkt zwischen Quelle und Ziel. Fraglich ist jedoch, wie die Ameisen es aufgrund ihrer bodennahen Position schaffen, überhaupt Futter zu finden, da sie bekanntlich nicht weit sehen können und jeder Stein oder Grashalm ein großes Hindernis darstellt? Ameisen haben dazu mittels einer Drüse am Hinterleib die Möglichkeit, so genannte Pheromone (Kunstwort aus dem Griechischen: *pherein* = übertragen und *horman* = erregen) auf ihrem Weg zu hinterlassen. An diesen chemischen Lockstoffen ihrer Vorgänger orientieren sich nachfolgende Ameisen und wählen mit einer gewissen Wahrscheinlichkeit den am stärksten markierten Weg. Doch um zu klären, warum durch diese Pheromone ein zumeist direkter Weg zwischen Futterquelle und Ameisenhügel entsteht, kann man das Verhalten an einem Modell näher betrachten. Ein ähnliches Modell wurde von den Biologen Deneubourg, Goss und Pasteels [87, 49, 36] verwendet, um mit einem gefangenen Ameisenvolk der Spezis Lasius niger in einem Glasröhrensystem das Verhalten der Ameisen nachzuweisen (so genanntes *Double-Bridge-Experiment*).

3.2.1 Modell der Nahrungssuche

Abbildung 3.3 beschreibt die Ausgangssituation des Modells, bei der zwei Ameisen zum Zeitpunkt $t = 0$ Minuten einen Weg von einem Ameisenhügel zu einer Futterquelle suchen. Zwischen diesen beiden Punkten befindet sich ein Hindernis, welches von den Ameisen nicht überschaut werden kann und um welches zwei verschiedene Wege führen. Für Weg A werden zwei Minuten (T_A) benötigt, für Weg B, der halb so lang sei, nur eine Minute (T_B). Die beiden Ameisen sind die ersten, die eine neue Futterquelle suchen, dass bedeutet, es existieren noch keine Pheromone auf den beiden Wegen ($P_A = P_B = 0$).

Es sei unterstellt, dass aufgrund der nicht vorhandenen Pheromone eine der beiden Ameisen den längeren Weg wählt und die andere Ameise den kürzeren. Zudem sei vorausgesetzt, dass alle zwei Minuten zwei weitere Ameisen den Ameisenhügel verlassen um Nahrung zu suchen. Mit diesen Prämissen sieht die Situation zum Zeitpunkt $t = 1$ Minute so aus, wie in Abbildung 3.4 dargestellt. Während sich die Ameise, welche den längeren Weg eingeschlagen hat, gerade erst auf halber Wegstrecke befindet, hat die andere Ameise die Futterquelle bereits erreicht und macht sich schon wieder auf den Rückweg. Nach einer weiteren Minute liegt die in Abbildung 3.5 dargestellte Situation vor.

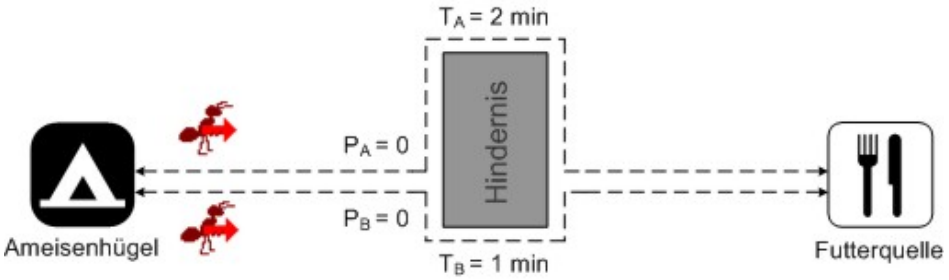

Abbildung 3.3: Modell der Nahrungssuche bei Ameisen zum Zeitpunkt $t = 0$ Minuten

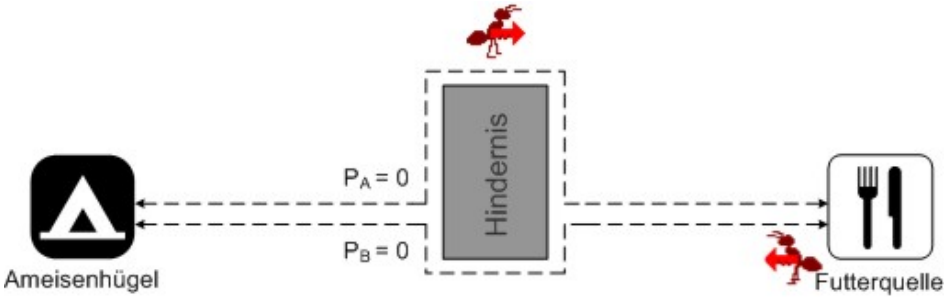

Abbildung 3.4: Modell der Nahrungssuche bei Ameisen zum Zeitpunkt $t = 1$ Minuten

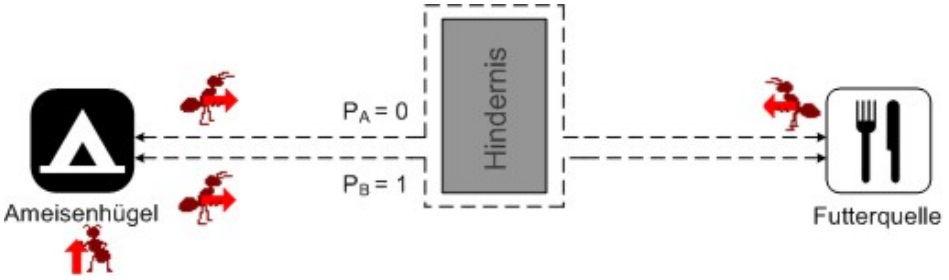

Abbildung 3.5: Modell der Nahrungssuche bei Ameisen zum Zeitpunkt $t = 2$ Minuten

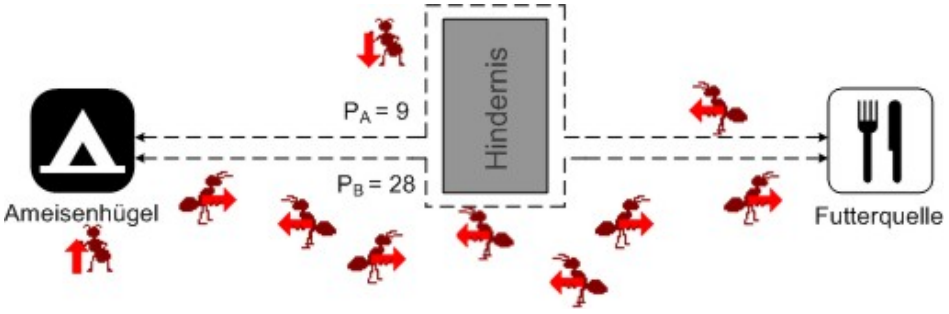

Abbildung 3.6: Modell der Nahrungssuche bei Ameisen zum Zeitpunkt $t = 20$ Minuten

In dem Moment, in dem die Ameise auf dem längeren Weg die Futterquelle erreicht, ist die andere Ameise bereits schon wieder zum Ameisenhaufen zurückgekehrt. Auf ihrem Rückweg hat diese neben dem Tragen der Nahrung auch noch eine Pheromonspur gelegt ($P_B = 1$), welche den zu diesem Zeitpunkt startenden Ameisen als Informationsquelle dient. Da letztere einen markierten Weg vorfinden, werden sie mit hoher Wahrscheinlichkeit den kürzeren Weg B einschlagen. Indem sie dann auf ihrem Rückweg ebenfalls wieder Pheromone hinterlassen, steigern sie dementsprechend die Konzentration der Pheromone auf diesem Weg, weshalb verhältnismäßig mehr nachfolgende Ameisen denselben Weg einschlagen werden. Nach 20 Minuten könnte die Situation die in Abbildung 3.6 dargestellte sein.

Durch den wenn auch zu anfangs sehr kleinen Unterschied in der Pheromonkonzentration beider Wege werden immer ein wenig mehr Ameisen auf den kürzeren der beiden Wege gelockt, welcher sich dadurch zu einer großen Ameisenstraße entwickeln wird. Wesentlich bei diesem Auswahlmechanismus ist jedoch, dass sich die Ameisen mit einer der Stärke der Pheromonkonzentration korrespondierenden Wahrscheinlichkeit für einen der beiden Wege entscheiden. Indem manche Ameisen vom Weg abweichen und einen neuen Weg beschreiten, können unter Umständen Abkürzungen gefunden werden. Da auf diesem neuen Weg in derselben Zeiteinheit nun mehr Ameisen hin und her laufen können als auf dem alten Weg, wird sich folglich die Pheromonkonzentration dort schneller erhöhen und über kurz oder lang eine neue Ameisenstraße entstehen.

3.2.2 Rolle der Pheromone

Die Pheromone nehmen bei der Nahrungssuche der Ameisen die Rolle einer Art kollektiven Gedächtnisses der Kolonie ein, welches die vergangenen Wegentscheidungen speichert. Dies stärkt die Robustheit der Ameisenkolonie gegenüber äußeren Einflüssen aus ihrer Umgebung. Zusätzlich wird über die Pheromone ein wichtiges Prinzip der Stigmergie illustriert: Globale Information (der Ort an dem Futter gefunden werden kann) wird lokal verfügbar gemacht. Überall dort, wo in der Umgebung einer Ameise Pheromonspuren existieren, lernt die Ameise über die Verfügbarkeit von Futter an entfernten Orten. Eine wichtige Eigenschaft von Pheromonen ist dabei auch, dass sie mit der Zeit verdunsten. Falls keine neuen Pheromone auf eine Spur gelegt werden, da die Futterquelle beispielsweise erschöpft ist, beginnen nachfolgende Ameisen mit der Suche nach neuen Wegen zu neuen Futterquellen anstatt fehlgeleitet zu werden. Des Weiteren übernehmen die Pheromone das positive Feedback (Verstärkung der Pheromonspuren von angezogenen Ameisen) für die Emergenz der Ordnung bei der Nahrungssuche ohne globale Koordination.

3.3 Von Ameisenkolonien zu Ameisenalgorithmen

Aufgrund der Tatsache, dass künstliche Ameisen nur schwer dazu bewegen sind, reale Pheromone in einem Computer abzulegen, mussten Wege gefunden werden, die Prinzipien der Stigmergie durch so genannte Ameisenalgorithmen technisch nachzubilden. Ameisenalgorithmen modellieren das Verhalten von natürlichen Ameisen bei der Nahrungssuche. Diese Nahrungssuche kann in der Informatik als Wegsuche auf einem Graphen beschrieben werden. Daher können mit Ameisenalgorithmen Probleme gelöst werden, die sich als Wegsuche auf Graphen formulieren lassen. Solche Probleme werden durch die folgenden Parameter beschrieben:

- Eine Menge von Randbedingungen, welche die Eigenschaften des Problems beschreiben (z. B. „Alle Knoten des Graphen müssen besucht werden“).
- Eine endliche Menge von Knoten, welche Teilprobleme darstellen.
- Nachbarschaftsbeziehungen zwischen den Knoten, welche mögliche Übergänge zwischen den Knoten beschreiben.
- Eine Lösung, welche durch einen Weg durch den Graphen beschrieben wird, der alle Randbedingungen des Problems erfüllt.
- Eine Kostenfunktion, welche jeder Lösung ihre Kosten zuweist.

Der erste, dem die technische Umsetzung des Verhaltens der Ameisen gelang, indem er die Nahrungssuche der Ameisen auf die Lösungssuche in kombinatorischen Optimierungsproblemen übertrug, war 1991 der italienische Mathematiker Dorigo. Das von ihm entwickelte Ant System (AS) [42] war der erste Ameisenalgorithmus, der zur Lösung des Travelling Salesman Problem (TSP) [99] genutzt wurde. Trotz zu anfangs nicht sehr überzeugender Ergebnisse im Vergleich zu spezialisierten Algorithmen hatte Ant System eine sehr wichtige Rolle: Es stimulierte weitere Forschungsarbeiten zu algorithmischen Varianten mit deutlich besserer Performance und es inspirierte eine Vielzahl neuer Anwendungen von Ameisenalgorithmen. Dorigo und Gambardella erweiterten Ant System 1995 um Q-Learning [103], einer Technik für Reinforcement Learning, zu Ant-Q [45] und verbesserten 1996 dessen Performance mit der Entwicklung des Ant Colony System (ACS) [46]. Alle Anwendungen, welche Ameisenalgorithmen zur Lösung von kombinatorischen Optimierungsproblemen verwenden, fallen in den Bereich der Ant Colony Optimization (ACO).

Aufgrund der wichtigen Ergebnisse von Dorigos Ansätzen für viele weitere Forschungen, wird die Funktionsweise von Ameisenalgorithmen anhand des Beispiels erklärt, wie sich das TSP mittels des ACS lösen lässt.

3.3.1 Travelling Salesman Problem

Das Travelling Salesman Problem (Handlungsreisendenproblem) zählt zu den bekanntesten kombinatorischen Optimierungsproblemen. Seinen Namen hat das Problem aus dem Handbuch von Voigt [66] für Handlungsreisende von 1832, welches die älteste bekannte Quelle für die Beschreibung des Problems ist. Darin gibt der Autor Hinweise, wie ein Handelsvertreter seine Rundreise durch eine Reihe von Städten zu seinen Kunden planen soll. Gesucht ist die kürzeste Rundreise, bei der alle Städte bzw. die dortigen Kunden genau ein einziges Mal besucht werden. Die im Buch vorgeschlagene Rundreise durch 45 Städte in Süddeutschland und der Schweiz erweist sich beim Nachrechnen als nicht optimal, jedoch bleiben in der modernen Fassung damalige Bedingungen unberücksichtigt.

Mathematisch kann das Problem folgendermaßen beschrieben werden: Sei $V = \{v_1, \ldots, v_n\}$ eine Menge von Städten, $A = \{(r, s) : r, s \in V\}$ die Kantenmenge und d_{rs} die einer Kante $(r, s) \in A$ zugeordneten Kosten. Das TSP besteht darin, eine geschlossene Tour minimaler Länge zu finden, bei der jede Stadt genau einmal besucht wird. Falls die Städte $v_i \in V$ mit ihren Koordination angegeben sind und d_{rs} die euklidische Distanz zwischen r und s ist, dann besteht ein euklidisches TSP. Ist $d_{rs} = d_{sr}$, so spricht man von einem symmetrischen TSP; gilt $d_{rs} \neq d_{sr}$ für auch nur eine Kante $(r, s) \in A$, so spricht man vom asymmetrischen TSP (ATSP).

3.3.2 Ant Colony System

Im ACS wird nur das ATSP betrachtet und das TSP als Spezialfall dessen angesehen. Der ACS-Algorithmus ist in Pseudonotation am Anhang A.1 auf Seite 93 dargestellt. Eine künstliche Ameise ist dabei ein Agent, welcher sich auf dem TSP-Graphen von Stadt zu Stadt bewegt. Der Agent wählt die Stadt, zu der er sich bewegen wird, durch eine wahrscheinlichkeitstheoretische Funktion aus akkumulierten Spuren auf Kanten und einem heuristischen Wert, welcher als Funktion über die Kantenlänge gewählt wurde. Künstliche Ameisen bevorzugen mit einer gewissen Wahrscheinlichkeit Städte, welche über Kanten mit einer hohen Pheromonkonzentration verbunden sind und welche nahe beieinander liegen.

Zunächst werden m künstliche Ameisen auf zufällig ausgewählte Städte gesetzt. Zu jedem Zeitschritt bewegen sie sich zu neuen Städten und modifizieren dabei die Pheromonspur auf den benutzen Kanten – Dorigo bezeichnet dies als lokale Aktualisierung der Spur. Sobald alle Ameisen ihre Tour abgeschlossen haben, modifiziert diejenige Ameise, welche die kürzeste Tour vollzogen hat, die Kanten auf ihrer Tour – als globale Aktualisierung der Spur bezeichnet – indem eine Menge von Pheromonen auf die Spur gelegt wird, welche umgekehrt proportional zu der Länge der Tour ist (vgl. [46]).

3.3.2.1 Übergangsfunktion

Sei k ein Agent im ACS, dessen Aufgabe es ist, eine Tour zu erarbeiten, bei der er alle Städte besucht und zur Ausgangsstadt zurückkehrt. Dabei besitzt k eine Liste $J_k(r)$ von Städten, die er noch zu besuchen hat, wobei r die Stadt ist, in der er sich gerade befindet (anders herum kann man sagen, dass sich Agent k die bereits besuchten Städte merkt). Ein in Stadt r befindlicher Agent k bewegt sich nach folgender Regel in eine Stadt s:

$$s = \begin{cases} \arg\max_{s \in J_k(r)} \left\{ [\tau(r,s)] \cdot [\eta(r,s)]^{\beta} \right\} & , q \leq q_0 \\ S & , sonst \end{cases} \tag{3.1}$$

wobei

- $\tau(r,s)$ eine positive reale Zahl ist, welche der Kante (r,s) zugeordnet ist und welche im ACS-Algorithmus zu den von Ameisen deponierten Pheromonen korrespondiert.
- $\eta(r,s)$ eine heuristische Funktion ist, welche den Nutzen einer Bewegung von r nach s auswertet. Beispielsweise ist im ATSP der Wert $\eta(r,s)$ gerade das Inverse der Distanz zwischen den Städten r und s.
- β ein Parameter ist, welcher die Relevanz der Heuristik abwiegt.
- q eine Zufallsvariable aus dem Intervall [0,1] und $q_0 \, (0 \leq q_0 \leq 1)$ ein Parameter ist. Je kleiner dabei q_0 gewählt ist, desto größer ist die Wahrscheinlichkeit eine zufällige Wahl zu treffen.
- S eine entsprechend gewählte Zufallsvariable aus der folgenden statistischen Auswahlfunktion ist, welche die Wahrscheinlichkeit p widerspiegelt, mit der sich ein Agent k in Stadt r in die Stadt s bewegt:

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{z\in J_k(r)}[\tau(r,s)]\cdot[\eta(r,s)]^\beta} & , s \in J_k(r) \\ 0 & , sonst \end{cases} \tag{3.2}$$

Die Zustandsübergangregel favorisiert Übergange zu Knoten, die über kurze Kanten verbunden sind und eine große Menge von Pheromonen aufweisen.

3.3.2.2 Lokale Aktualisierung der Pheromonspuren

Eine Pheromonspur kann sowohl lokal als auch global verändert werden. Nach Dorigo ist die Intention der lokalen Aktualisierung, sehr starke Kanten als lokale Optima zu vermeiden, welche von allen Ameisen immer wieder gewählt werden. Während Ameisen eine Lösung für das TSP suchen, verändern sie die Pheromonmenge auf den besuchten Kanten nach der folgenden lokalen Aktualisierungsregel:

$$\tau(r,s) := (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \tag{3.3}$$

Der Faktor ρ spiegelt in der Formel die natürliche Verdunstung des Pheromons wieder und τ_0 ist ein beliebiger Parameter, meist als sehr kleine Konstante gewählt. Somit wird jedes Mal, wenn eine Kante (r,s) von einer Ameise benutzt wird, deren Pheromonmenge durch Anwendung der lokalen Aktualisierungsregel verringert und die Kante für nachfolgende Ameisen mehr und mehr unattraktiv. Dies steigert die Erforschung von unbekannten Kanten.

3.3.2.3 Globale Aktualisierung der Pheromonspuren

Die Globale Aktualisierung sorgt für eine größere Anzahl von Pheromonen auf kürzeren Touren. Sobald alle Ameisen ihre Lösungen fertig gestellt haben, deponiert die Ameise mit der kürzesten Tour Pheromone auf den von ihr besuchten Kanten (r,s) nach folgender Aktualisierungsregel:

$$\tau(r,s) := (1-\alpha)\cdot\tau(r,s) + \alpha\cdot(L_{best-iter})^{-1} \tag{3.4}$$

In gewisser Weise entspricht diese Formel dem *Reinforcement Learning*-Schema, bei dem bessere Lösungen eine höhere Bewertung bekommen. Dorigo fand zwei verschiedene Möglichkeiten, die beste Ameise für die globale Aktualisierung zu bestimmen: Die iterations-beste Aktualisierung oder die global-beste Aktualisierung. Bei der iterations-besten Methode wird die Ameise mit der kürzesten Tour innerhalb einer Iteration L_{iter} gewählt, wogegen bei der global-besten Methode die Ameise gewählt wird, welche die kürzeste Tour seit Beginn der Lösungssuche L_{best} berechnete (in Experimenten wurde später immer L_{best} verwendet). Die Anzahl der von der besten Ameisen deponierten Pheromone $(L_{best-iter})^{-1}$ auf jeder besuchten Kante (r,s) ist indirekt proportional zu der Länge der Tour: Je kürzer die Tour ist, desto größer ist die Anzahl der deponierten Pheromone auf deren Kanten. Der Faktor α entspricht der Verdunstung der Pheromone.

3.3.3 Evaluierung des Ant Colony Systems

In [46] finden sich Performance-Ergebnisse von Experimenten mit ACS zur Lösung des TSP mit mehr als 1500 Städten. Vergleiche mit anderen natürlich inspirierten Algorithmen

in [41] über die durchschnittlich kürzesten, berechneten Wege zeigen die Stärke des gefunden ACS-Algorithmus. In Dorigos künstlicher Ameisenkolonie wurden drei Ideen aus dem natürlichen Verhalten von Ameisen transferiert:

- Die Präferenz für einen Weg mit einer hohen Pheromonkonzentration
- Die größere Wachstumsrate der Anzahl an Pheromonen auf kürzeren Pfaden
- Die Kommunikation der Ameisen über Pheromone

Zusätzlich wurden die künstlichen Ameisen mit einigen Fähigkeiten ausgestattet, welche kein natürliches Vorbild besitzen, jedoch für die TSP-Lösung gut geeignet sind. Künstliche Ameisen können bestimmen, wie weit Städte voneinander entfernt liegen, und sind mit einem Speicher J ausgestattet, in dem alle bereits besuchten Städte gespeichert werden. Zu Beginn jeder Tour wird dieser Speicher geleert und bei jedem Zeitschritt durch Hinzufügen einer neu besuchten Stadt aktualisiert.

3.4 Ameisenalgorithmen zur Lösung von kombinatorischen Optimierungsproblemen

Ameisenalgorithmen werden in erster Linie zur Lösung kombinatorischer, NP-harter Optimierungsprobleme eingesetzt. Generell können Anwendungen auf Basis von Ameisenalgorithmen zwei unterschiedlichen Klassen kombinatorischer Optimierungsprobleme zugeteilt werden, zu statischen und dynamischen Problemen. Statische Probleme sind solche, deren Topologie und Kosten während der Problemlösung konstant bleiben; ein Beispiel hierfür ist das Travelling Salesman Problem. Dagegen können sich in dynamischen Problemen Topologie und Kosten während des Lösungsprozesses verändern; ein Beispiel hierfür ist das Routing in Netzwerken (siehe unten). Im Folgenden werden verschiedene kombinatorische Optimierungsprobleme vorgestellt, bei deren Lösung sehr gute Ergebnisse mit Ameisenalgorithmen erzielt wurden:

Sequential Ordering Problem (SOP) Das sequentielle Ordnungsproblem mit Vorrangsbedingungen kann für Produktionsplanungssysteme oder Transportprobleme in flexiblen Produktionssystemen verwendet werden und wurde zum ersten Mal 1988 mit dem Ziel formuliert, eine Prozesssequenz zu finden, welche die gesamte Produktionsdauer minimiert und den Vorrangsbedingungen unterliegt. Das SOP kann als ein allgemeiner Fall des ATSP angesehen werden, indem man die Gewichtung von den Knoten wegnimmt und anstatt dessen die Kanten gewichtet. Dorigo und Gambardella entwickelten zur Lösung des SOP den Ameisenalgorithmus ACS-SOP mit ähnlicher Funktionalität wie ACS. Zusätzlich wurde dieser um die Fähigkeit zur lokalen Suche zu HAS-SOP (Hybrid Ant System for the Sequential Ordering Problem) [44] erweitert.

Vehicle Routing Problem (VRP) Das VRP ist ein sehr kompliziertes Problem, welches seit den späten fünfziger Jahren des letzten Jahrhunderts untersucht wird, da es eine zentrale Bedeutung im Verteilungsmanagement besitzt. Ziel dabei ist es, Fahrzeugrouten mit minimalen Kosten zu finden, wobei (1) jeder Kunde genau einmal von genau einem Fahrzeug besucht wird, (2) alle Fahrzeugrouten am Depot beginnen und dort wieder enden, (3) für jede Fahrzeugroute die gesamten Aufträge nicht die Ladekapazität

Q eines Fahrzeug überschreiten und (4) für jede Fahrzeugroute die gesamte Routenlänge (incl. Servicezeiten) eine festgelegte Grenze L nicht überschreitet. Ein wichtiger Lösungsansatz für das VRP kommt von Bullnheimer [23, 24], welcher jedoch sehr stark vom Ant System-Algorithmus von Dorigo beeinflusst wurde.

Job Shop Scheduling Problem (JSP) Das Problem beim JSP besteht beispielsweise darin, jeden Prozessschritt für die Fertigung eines Produktes auf eine Maschine aufzuteilen (Routing Problem) und die Prozessschritte auf einer Maschine zu ordnen (Scheduling Problem), so dass die maximale Bearbeitungszeit für alle Prozessschritte minimiert wird. In [31] präsentieren Colorni und Dorigo einen ameisenbasierten Ansatz, wiederum aufbauend auf dem Algorithmus aus Ant System.

Quadratic Assignment Problem (QAP) Das quadratische Zuordnungsproblem kann am besten beschrieben werden, als das Problem vom Zuordnen einer Menge aus n Objekten auf eine Menge von n Orten. Dabei ist jede Zuordnung mit Kosten belegt, die jeweils von der Entfernung und dem Fluss zwischen den Objekten abhängen. Das Ziel ist es dann, die Objekte so auf die Orte zu platzieren, dass die gesamten Zuordnungskosten minimal sind. In [67] und [93] präsentieren Stützle und Dorigo den HAS-QAP-Algorithmus, der wie der HAS-SOP-Algorithmus zusätzlich noch über die Möglichkeit zur lokalen Suche verfügt.

Graph Coloring Problem (GCP) Beim GCP geht es darum, einen Graphen so zu färben, dass zwei beliebige aneinander liegende Knoten verschiedene Farben besitzen. Zur Lösung dieses Problems entwickelten Costa und Herz AntCol [33]. Der darin enthaltene Ameisenalgorithmus bildet bei jeder Iteration eine mögliche Lösung für die Färbung des Graphen.

Routing in Netzwerken Die Hauptaufgabe eines Routingalgorithmus ist es, den Datenfluss von einem Quell- zu einem Zielknoten in einem Graphen – wie z. B. dem Internet – zu dirigieren und dabei die Netzwerkperformance zu maximieren (vgl. [39]). Di Caro und Dorigo präsentieren mit AntNet [38] einen Routingalgorithmus, bei dem jede künstliche Ameise einen Weg von ihrem Quellknoten zu ihrem Zielknoten bildet.

4. Selbstorganisierende IT-Systeme auf Basis der Stigmergie

Ameisenalgorithmen werden nicht nur für die Lösung kombinatorischer Optimierungsprobleme verwendet, sondern auch zur erfolgreichen Realisierung von größeren, selbstorganisierenden IT-Systemen in der Industrie. Dieses Kapitel beschreibt aus der Domäne der Fertigungssteuerung ein Produktionsplanungs- und -kontrollsystem, welches auf der Stigmergie basiert und sich bereits im Einsatz befindet. Eine anschließende Analyse dieser Fallstudie wird zeigen, wie die Prinzipien der Stigmergie im Produktionsplanungs- und -kontrollsystem umgesetzt wurden und welche Vorteile die Verwendung der Stigmergie für das System hat. Aufgrund dieser Analyse werden Anforderungen und Technologien für die Architektur von selbstorganisierenden IT-Systemen auf Basis der Stigmergie identifiziert. Diese Ergebnisse können für die Entwicklung von Organic Computing Systemen verwendet werden.

Die heutige Produktionsindustrie sieht sich einem bedeutenden Wandel von einem Lieferantenmarkt hin zu einem Kundenmarkt gegenüber (vgl. [26, S. 1]). Der wachsende Überschuss industrieller Kapazität bietet dem Kunden eine größere Auswahl und steigert den Wettbewerb zwischen den Lieferanten. Ihrer Macht bewusst werden Kunden immer anspruchsvoller und weniger loyal gegenüber bestimmten Produkten. Als Resultat müssen die Unternehmen Produktlebenszyklen verkürzen, Produkte schneller auf den Markt bringen, die Produktvielfalt steigern und schneller die Nachfrage erfüllen, während die Qualität erhalten und die Investitionskosten gesenkt werden müssen. Dies ist eine große Herausforderung für den Produktionsprozess an sich. Er muss flexibler, robuster und noch skalierbarer werden.

Valckenaers und seine Kollegen [97, 96, 95, 53] aus diesem Grund ein selbstorganisierendes Produktionsplanungs- und -kontrollsystem auf Basis der Stigmergie entwickelt, welches Produktionssysteme von Unternehmen logistisch unterstützt.

4.1 Fallstudie I: Rechnergestützte Produktionsplanung und -kontrolle

Das Produktionsplanungs- und -kontrollsystem basiert auf der PROSA-Referenzarchitektur [98] und ist als Multi-Agenten-System realisiert. Die Agenten werden als intelligente und flexible Einheiten eingesetzt, welche ihre Umgebung beobachten. Sie führen dabei so autonom

wie möglich bestimmte Aufgaben aus und verändern dadurch ihre Umgebung auf eine gezielte Art und Weise. Eine Schlüsseleigenschaft des Systems ist es, die zukünftige Auslastung der Produktionsanlagen exakt zu planen. Zusätzlich werden die Routen von Produktinstanzen durch das Produktionssystem berechnet, das Starten von Prozessen an noch nicht fertig gestellten Produktinstanzen ermittelt, Maschinenausfälle kompensiert und viele weiteren Anforderungen an moderne Produktionsplanungs- und -kontrollsysteme erfüllt.

4.1.1 Systemarchitektur

Die wesentlichsten Komponenten des Systems sind Agenten und eine dissipative Umgebung. Letztere ist eine wichtige Einrichtung, um die Aktivitäten der Agenten zu leiten und um Informationen über laufende Aktivitäten der gesamten Agentengemeinschaft zu sammeln.

4.1.1.1 Dissipative Umgebung

Abbildung 4.1 zeigt die Topologie einer exemplarischen Produktionsstraße mit den Transportbändern B1 bis B5, den Kreuzungen X1 und X2, und den Ressourcen (Produktionsanlagen) R1, R2 und R3. Die dissipative Umgebung des Planungs- und Kontrollsystems repräsentiert diese physikalische Struktur intern als eine Menge von Pheromonstellen. Jedes physikalische Element ist mit einer Pheromonstelle im System verbunden. Die Pheromonstellen sind auf die gleiche Weise miteinander verbunden wie die physikalischen Elemente.

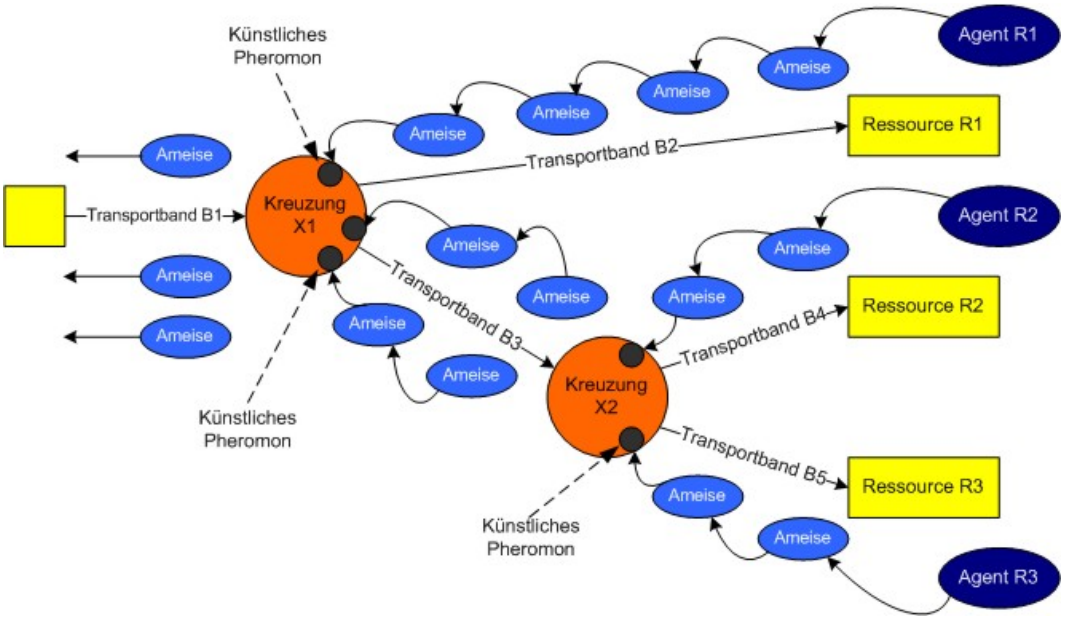

Abbildung 4.1: Dissipative Umgebung des Produktionsplanungs- und -kontrollsystems [97]

4.1.1.2 Agenten

Die Agenten sind einfach gebaut, reaktiv, umgebungsorientiert und haben keine Kenntnis über andere Agenten oder die Ziele der Agentengemeinschaft. Die zugrunde liegende PROSA-Referenzarchitektur ist auf drei verschiedenen Typen von Basisagenten aufgebaut:

Ressourcenagent Ein Ressourcenagent reflektiert einen physischen Teil – nämlich eine Produktionsressource des Produktionssystems – und einen informationsverarbeitenden Teil,

welcher die Ressource kontrolliert. Der Agent bietet die Produktionskapazität und -funktionalität den umgebenden Agenten an und besitzt Methoden zur Kontrolle dieser Produktionsressourcen, um die Produktion durchzuführen. Ein Ressourcenagent ist eine Abstraktion der Produktionsmittel, wie Maschinen, Schmelzöfen, Transportbänder, Paletten, Personal, usw.

Zur Implementierung des stigmergischen Ansatzes stellt ein Ressourcenagent auf der anderen Seite eine Art „Schwarzes Brett“ als Pheromonstelle bereit, auf dem sich Informationen befinden können und für interessierte Agenten verfügbar gemacht werden. Überdies bieten Ressourcenagenten einen Graphen an, welcher die lokalen Verbindungen zu benachbarten, physischen Ressourcen reflektiert. Dieser Graph erlaubt es anderen Agenten virtuell durch die Produktionsstraße zu reisen.

Produktagenten Ein Produktagent besitzt das Prozess- und Produktwissen um die korrekte Erstellung von Produkten mit ausreichender Qualität zu sichern. Ein Produktagent besitzt konsistente und aktuelle Informationen über den Lebenszyklus, die Kundenanforderungen, Design, Prozesspläne, Stückliste, Qualitätssicherung, Prozeduren, usw. eines Produktes. Als solches enthält er das Produktmodell eines Produktes, jedoch nicht das Produktzustandsmodell einer physischen Produktinstanz, welche produziert wird. Der Produktagent agiert lediglich als Informationsprovider für andere Agenten.

Auftragsagent Ein Auftragsagent repräsentiert einen Auftrag im Produktionssystem. Er ist für die zeitliche und korrekte Ausführung des Auftrags verantwortlich und verwaltet das produzierte, physische Produkt, das Produktzustandsmodell und alle logistischen Informationen für die Ausführung des zugeordneten Auftrags. Ein Auftragsagent kann Kundenaufträge, Serienaufträge, Prototypaufträge, Reparaturaufträge o. Ä. repräsentieren. Oftmals kann ein Auftragsagent als das Werkstück selber mit einem zusätzlichen Kontrollverhalten betrachtet werden, um es auf dem Weg durch die Produktionsstraße zu verwalten, mit anderen Worten, um mit anderen Teilen und Ressourcen zu verhandeln, so dass das Werkstück produziert wird.

Zusätzlich werden Ameisenagenten mit fachspezifischem Wissen zur Unterstützung der Basisagenten hinzugefügt. Durch ihre Aktivitäten erstellen die Ameisenagenten dissipative Felder in der Umgebung, welche das Verhalten gegenseitig beeinflussen und wie bei den realen Ameisen bestimmte Verhaltensmuster auftreten lassen.

4.1.2 Koordinationsmechanismus

Der Koordinationsmechanismus des Systems ist inspiriert von der Nahrungssuche in Ameisenkolonien und läuft auf drei verschiedenen Kontrollstufen (*Layers*) ab, bei denen Informationen propagiert werden (vgl. [53]):

4.1.2.1 Feasibility Layer

Beginnend bei der Ressource, welche zu dem Ende der Produktionsstraße korrespondiert, werden Ameisenagenten *feasibility ants* in einer bestimmten Frequenz erzeugt. Diese Frequenz wird als *refresh rate* bezeichnet. Eine *feasibility ant* zeigt folgendes Verhalten:

- Zuerst verlangt die Ameise eine Beschreibung der Produktionsfähigkeiten der Ressource, auf der sie sich momentan befindet.

- Anschließend wird diese Beschreibung mit den bisher gesammelten Informationen über Produktionsfähigkeiten zusammengebracht, welche zu anfangs leer sind.
- Danach klont sich die Ameise so oft wie Eingänge bei der aktuellen Ressource vorhanden sind, so dass für jeden Eingang eine Ameise zur Verfügung steht. Jede Ameise navigiert nun in der virtuellen Kopie der zugrunde liegenden Produktionsstraße zu dem korrespondierenden Ausgang jeder vorausgehenden Ressource (siehe Abbildung 4.1). Dabei haben Ressourcen, welche zu dem Anfang der Produktionsstraße korrespondieren, keine Eingänge.
- Bei der Ressource angekommen, überprüft die Ameise diejenigen Produktionsfähigkeiten, welche auf dem am Ausgang der Ressource platzierten Informationsraum (Schwarzes Brett) von ihrer eigenen Generation abgelegt wurden, und fügt diese mit ihren Informationen zusammen (Ameisen können über verschiedene Routen denselben Ausgang erreichen). Sind die eigenen Informationen bereits verfügbar, so stirbt die Ameise. Andernfalls deponiert die Ameise die neuen Informationen durch Überschreiben der alten und beginnt mit dem ersten Schritt ihres Verhaltensmusters bei dieser Ressource.

Durch die Deponierung von Informationen entstehen auf den Informationsräumen an den Ausgängen der Ressourcen so genannte *subnet processing capabilities* (siehe [63]). Abbildung 4.2 zeigt die Topologie der Produktionsstraße mit explizit markierten Teilnetzen. Die Ressourcen R1, R2 und R3 haben überlappende Fähigkeiten C1 bis C5. Die abgelegten Informationen bleiben jedoch nur für eine kleine Anzahl von Auffrischungszyklen gültig.

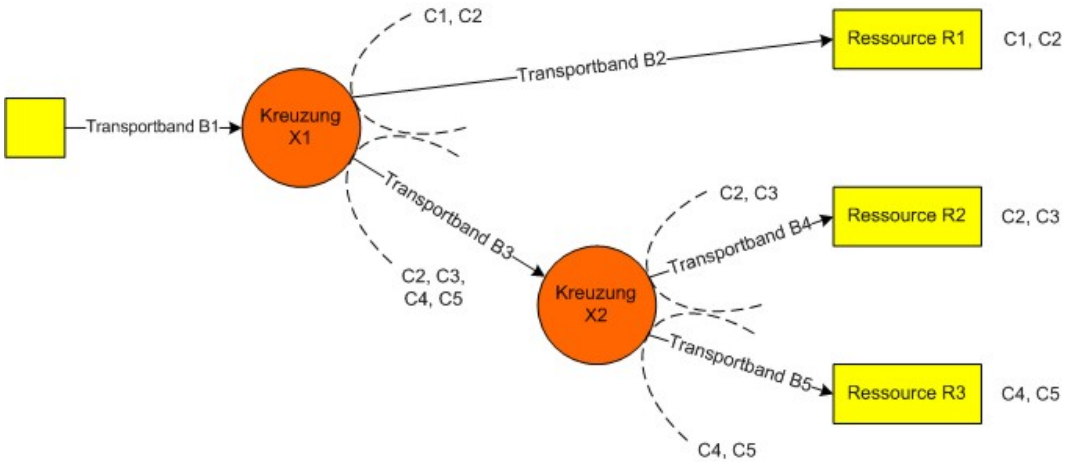

Abbildung 4.2: *Subnet processing capabilities* des Produktionsplanungs- und -kontrollsystems [97]

Wenn die Ameisenagenten der Auftragsagenten vorwärts durch die Produktionsstraße reisen, nehmen sie die an den Ausgängen abgelegten Informationen auf und leiten diese an den zuständigen Produktagenten weiter, welcher entscheidet, ob dieser Ausgang für die Umsetzbarkeit in Frage kommt (alle benötigten Prozessschritte müssen in einer passenden Sequenz vorkommen, ohne dabei zwangsweise die Optimalität zu garantieren). Somit begrenzt die Feasibility Layer den Suchraum auf Routen, welche technisch umsetzbar sind.

4.1.2.2 Exploring Layer

Im Folgenden wird angenommen, dass ein Auftragsagent einem einzelnen Werkstück zugeordnet werden kann. Ein Auftragsagent muss sein Werkstück so durch die Produktionsstraße leiten, dass es allen benötigten Prozessschritten unterzogen wird. Um die verfügbaren Möglichkeiten zu erforschen, erzeugen die Auftragsagenten mit einer geeigneten Frequenz Ameisenagenten (*exploring ants*), welche virtuell durch die Produktionsstraße reisen, angefangen an der aktuellen Position des Werkstücks bis zum Ende der Straße. Jede *exploring ant* erforscht die erwartete Performance einer einzelnen umsetzbaren Route für die Produktinstanz durch die Produktionsstraße, wobei sich diese Ameisen im Gegensatz zu den *feasibility ants* nicht selber klonen.

Während die Ameisen virtuell durch die Produktionsstraße reisen und virtuell produziert werden, fragen die *exploring ants* bei den Ressourcen, welchen sie auf ihrem gewählten Weg begegnen, nach deren erwarteter Performance (Auslastung, frühester Prozessstart, ...). Sobald die Ameisen am Ende angekommen sind, berichten sie die erforschte Route an ihre Auftragsagenten, welche die erwartete Performance der gesamten Route berechnen. Die Auftragsagenten speichern eine Menge von geeigneten Routen, welche kürzlich erforscht wurden.

4.1.2.3 Intention Layer

Hat ein Auftragsagent genügend Informationen über geeignete Routen gesammelt, wählt er die beste Route aus. Diese Route wird dann zu seiner Intention. In gleichmäßigen Intervallen erzeugt der Auftragsagent Ameisenagenten (*intention ants*), welche die Ressourcenagenten entlang dieser Route über die Intention ihres Auftragsagenten informieren. Eine *intention ant* unterscheidet sich dabei in zwei Punkten von einer *exploring ant*. Auf der einen Seite hat sie eine feste Route, der sie folgen muss, auf der anderen Seite tätigt sie Reservierungen bei der Ressource. Durch die Reservierungen können Ressourcenagenten lokale Auslastungsprofile erstellen, welche die nahe Zukunft betreffen. Diese Profile werden dazu verwendet, *exploring ants* und auch *intention ants* genaue Performancedaten zu geben.

Der so genannte *refresh-and-forget*-Mechanismus der natürlichen Ameisen wird bei der Propagation von Intentionen ebenfalls angewandt. Sobald es ein Auftragsagent verpasst, seine Reservierung zu erneuern, wird sie aus dem lokalen Auslastungsprofil der Ressource entfernt. Überdies hält dieser Mechanismus die Auftragsagenten über die erwartete Performance ihrer vorgesehenen Route informiert. Kommt eine *intention ant* am Ende der Produktionsstraße an, so berichtet sie die Performance der aktuellen Intention an ihren Auftragsagenten zurück. Sollte sich die Situation in der Produktionsstraße verändern, z. B. durch einen Maschinenausfall oder durch das Eintreffen von Aufträgen mit höherer Priorität, so kann sich die Performance der Route positiv oder negativ ändern. Da das Erforschen neuer Routen parallel zu der Propagation von Intentionen läuft, könnten die *exploring ants* attraktivere Routen zurückmelden. Ein Auftragsagent kann in diesem Fall seine Intention verändern.

4.1.3 Produktionsvorgang

Ein Auftragsagent beobachtet den Fortschritt der korrespondierenden, physischen Produktinstanz und führt, wenn die Zeit gekommen ist, den nächsten Schritt seiner Intention aus. Das sozial verträgliche Verhalten der Agenten impliziert dabei, dass eine kurzfristige Abweichung von der Intention nur in sehr ernsthaften Situationen geschehen darf. Das Ändern der Intention ist für Situationen akzeptabler, welche weit in der Zukunft liegen, so dass Auftragsagenten die Möglichkeit haben, darauf zu reagieren.

4.2 Analyse der Fallstudie I

4.2.1 Umsetzung der Prinzipien der Stigmergie

Änderungen in der Produktionsstraße, wie z. B. der Ausfall einer bestehenden Ressource oder das Hinzukommen einer neuen Ressource, werden im System durch den *refresh-and-forget*-Mechanismus geregelt. Indem die Produktionsfähigkeiten in abgelegten Pheromonen immer wieder erneuert werden müssen (Verdunstung), bleiben keine alten bzw. falschen Informationen erhalten, so dass auch neue Informationen ohne Probleme hinzugefügt werden können. Zudem speichern die Auftragsagenten nur die kürzlich erforschten Routen (Verdunstung). Ältere Routen könnten schon nicht mehr attraktiv sein und werden so durch neuere ersetzt.

Die Systemarchitektur weist die in Abbildung 4.3 dargestellten Konzepte auf, welche für die Umsetzung der Stigmergie mittels Ameisenalgorithmen in selbstorganisierenden IT-Systemen notwendig sind:

- Eine dissipative Umgebung als verteilte Infrastruktur für die Propagation von Pheromonen und die Erschaffung eines künstlichen, dissipativen Felds
- Agenten als agierende Individuen[1]
- Pheromone als Informationsträger

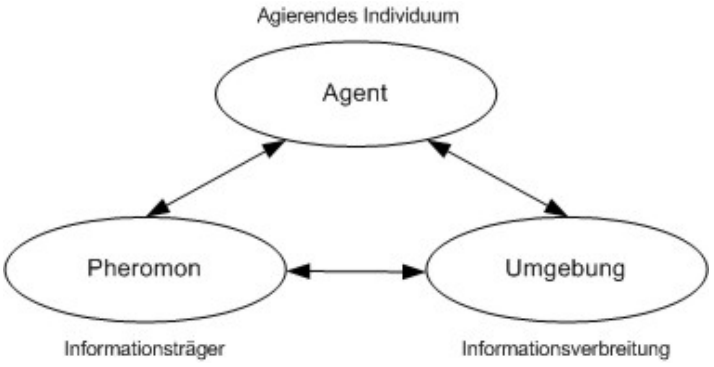

Abbildung 4.3: Elemente eines stigmergischen Systems

Die dissipative Umgebung ist eine essentielle Komponente des beschriebenen Systems. Sie stellt Mechanismen für die Ameisenagenten bereit, aufgrund derer sich letztere durch das virtuelle System bewegen und Informationen innerhalb des Systems verbreiten und erhalten können. Eine Umgebung für stigmergische IT-Systeme muss daher folgende architektonische Konzepte unterstützen:

- Pheromonstellen, an denen sich Agenten befinden und Informationen ablegen oder aufspüren können
- Ein Managementmechanismus, um künstliche Pheromonobjekte an diesen Stellen zu erhalten und zu speichern

[1] Die Eingrenzung auf Agenten als agierende Individuen ist nicht zwingend notwendig. Diese Rolle kann prinzipiell auch von anderen Technologien übernommen werden, welche hier jedoch nicht untersucht wurden.

- Ein Propagationsmechanismus, um den Agenten eine Bewegung innerhalb einer solchen Topologie von Pheromonstellen zu ermöglichen

Die Struktur der dissipativen Umgebung ist abhängig von der Anwendung und kann beliebig gewählt werden.

Das globale Verhalten des Systems entsteht aus den Aktivitäten relativ einfacher Agenten. Die wichtigsten Eigenschaften der Agenten in stigmergischen IT-Systemen sind daher:

Kein Wissen über die gesamte Systemkomplexität Agenten agieren auf Basis von Informationen, welche an ihrem momentanen Standort verfügbar sind. Global notwendige Informationen werden anhand von Verteilungs- und Propagationsmechanismen lokal verfügbar gemacht und so ein dissipatives Feld geschaffen. Kein Agent besitzt dadurch ein Wissen über die übergeordneten Ziele des Systems.

Selbstreflektion auf die Umgebung Agenten sind in keine direkte Kommunikation verwickelt und sind sich keiner Partneragenten bewusst. Die Aktivitäten innerhalb einer stigmergischen Agentengemeinschaft werden von den Agenten koordiniert, indem sie ihre Intentionen in der dissipativen Umgebung ablegen und andere, daran interessierte Agenten diese Informationen aufspüren und daraufhin selber Informationen ablegen.

4.2.2 Technologien für die Realisierung stigmergischer Systeme

Stigmergische IT-Systeme müssen auf einer Technologie basieren, welche diese Anforderungen unterstützen. Für die Implementierung werden daher Technologien benötigt, welche auf Modellen mit generativer Kommunikation basieren. Dies sind Technologien, welche generell durch Daten- oder Tupel-Spaces realisiert werden und im Linda System [29] ihren Ursprung haben. Entwicklungen mit ähnlichen Ansätzen sind Swarm [89], CHAM [20], Gamma [13] oder die Jini Technologie von Sun Microsystems [100].

4.2.3 Vorteile durch die Verwendung der Stigmergie

Die Verwendung der Stigmergie im Produktionsplanungs- und -kontrollsystem weist Vorteile gegenüber traditionellen Systemen auf, welche nicht auf der Stigmergie basieren:

- Die Agenten kommunizieren indirekt miteinander und interagieren nur mit der Umgebung nach standardisierten Protokollen. Die Agenten können sich dadurch auf das Abgeben und Aufspüren von Informationen konzentrieren und müssen sich nicht um den Zustand anderer Agenten kümmern.

- Diese einfache Kopplung zur Agentengemeinschaft ermöglicht es Agenten, in die Gemeinschaft einzutreten oder sie zu verlassen, ohne sie zu beeinträchtigen. Zudem besteht keine Veranlassung, andere Agenten über das Eintreten oder das Verlassen zu informieren. So können beispielsweise Ressourcen in die physische Produktionsstraße eingefügt oder herausgenommen werden, ohne dass das System neu gestartet werden muss.

- Die Erforschung neuer Routen und die Propagation von Intentionen der Auftragsagenten geschieht in viel höherer Geschwindigkeit als in der realen, physischen Produktion. Mit anderen Worten produziert das Produktionsplanungs- und -kontrollsystem das benötigte

Produkt virtuell weit bevor die tatsächliche Produktionsaktivität eingeleitet wird. Wie viel Rechenaufwand dafür betrieben wird, hängt von dem wirtschaftlichen Wert besserer Routen und anderer Sequenzentscheidungen für ein Unternehmen ab.

- Das System führt automatisch zu optimierten Lösungen. Die Erforschung von Wegen durch Agenten aufgrund der teilweise zufälligen Routenwahl garantiert, dass die Gemeinschaft ein robustes Verhalten aufzeigt. Die Verbreitung von globalen Informationen durch die Verweise auf Ressourcen und die Rückmeldung auf das Verhalten der Agentengemeinschaft durch die Anziehungskraft besserer Auslastungsprofile führt das System zu einer Lösung, welche optimiert ist, jedoch nicht zwangsweise optimal. Als Resultat wird die Gemeinschaft daher als agierende Entität eine optimierte Lösung finden, ohne Robustheit oder Anpassungsfähigkeit zu verlieren.

5. Erzeugung von Selbst-x Eigenschaften für IT-Systeme

Die in Kapitel 4 gesammelten Anforderungen an die Architektur von selbstorganisierenden IT-Systemen auf Basis der Stigmergie reichen noch nicht aus, um sinnvolle Organic Computing Systeme zu entwickeln. OCS sind zwar selbstorganisierende IT-Systeme, jedoch besitzen sie wie in Kapitel 1 beschrieben gewisse Selbst-x Eigenschaften, welche nicht allein durch die Verwendung eines Selbstorganisationsmechanismuses wie der Stigmergie entstehen. Für die Erstellung des Entwicklungsprozesses für OCS fehlen noch Informationen über die technische Erzeugung von Selbst-x Eigenschaften für IT-Systeme.

In diesem Kapitel wird daher eine mögliche Erzeugung von Selbst-x Eigenschaften genau untersucht. Die Untersuchung findet auf Basis der Referenzarchitektur für Autonomic Computing Systeme [56] statt. Dieses Vorgehen ist möglich, da sich die Selbst-x Eigenschaften des Autonomic Computing von denen des Organic Computings bis auf wenige Punkte nur im Anwendungsbereich unterscheiden, nicht jedoch im Prinzip der technischen Erzeugung. Der Vorteil der Untersuchung der Erzeugung im AC liegt darin, dass aufgrund der erheblich größeren Forschungsgemeinschaft im Vergleich zum OC bereits detailliertere Informationen darüber vorliegen, wie die in 2.1 vorgestellten Ansätze des OC für Helper Threads bzw. Observer/Controller-Architekturen konkret umgesetzt wurden. Aus dieser Untersuchung werden konzeptionelle Anforderungen an IT-Systeme identifiziert, um ein systemweites Selbst-x Verhalten zu ermöglichen.

5.1 Konzepte des Autonomic Computing

Autonomic Computing Systeme tasten ähnlich wie OCS ihre Umgebung ab, analysieren ihr eigenes Verhalten und vollziehen Aktionen, um die Umgebung oder ihr Verhalten zu verändern. Zur technischen Erzeugung der Selbst-x Eigenschaften für diese Systeme definiert die Architektur Konzepte wie *Managed Resource*, *Touchpoint*, *Touchpoint Autonomic Manager*, *Orchestrating Autonomic Manager* und *Integrated Solution Console*, deren Zusammenhang in Abbildung 5.1 dargestellt ist. Für die Realisierung dieser Konzepte werden verschiedene Technologien (Common Information Model, Simple Network Management Protocol (SNMP),

Web Services (WS), Java Management Extensions (JMX), Open Grid Service Architecture (OGSA), ...) verwendet.

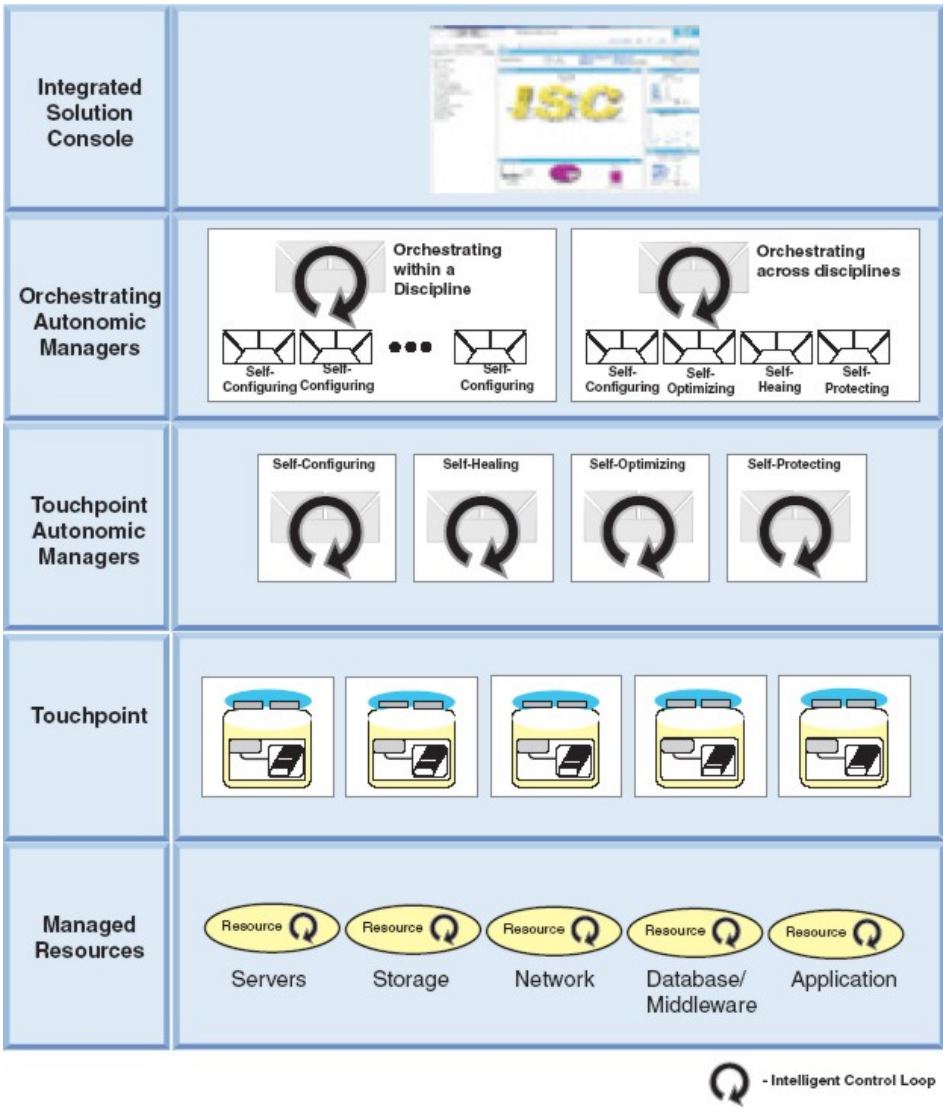

Abbildung 5.1: Autonomic Computing Referenzarchitektur [56]

Die Referenzarchitektur enthält auf der untersten Schicht so genannte Managed Resources, welche als Systemkomponenten die IT-Infrastruktur ergeben und auch selber Fähigkeiten zum Selbst-Management enthalten können. Die nächste Schicht enthält konsistente, standardisierte *Manageability Interfaces*, um die Managed Resources zu kontrollieren und einen Zugriff auf diese zu ermöglichen. Diese Interfaces werden jeweils durch einen so genannten Touchpoint implementiert. Eine konkrete Ressource kann einen oder mehrere so genannte Touchpoint Autonomic Managers besitzen, von denen jeder eine relevante Kontrollschleife implementiert und so eine Managed Resource mit einer Selbst-x Eigenschaft ausstattet. In Abbildung 5.1 wird jede Selbst-x Eigenschaft durch einen einzigen Autonomic Manager dargestellt, jedoch könnte auch ein einziger Autonomic Manager diese Aufgaben übernehmen. Die vierte Schicht enthält Autonomic Managers, welche die Touchpoint Autonomic Managers orchestrieren. Diese so genannten Orchestrating Autonomic Managers sorgen für die systemweiten, autonomen Fähigkeiten, indem sie Kontrollschleifen implementieren, welche den breitesten Überblick über

die gesamte Infrastruktur besitzen. Die oberste Schicht sorgt für ein einheitliches System-Management für die IT-Entwickler durch eine so genannte Integrated Solutions Console. Da diese Schnittstelle nur für das Management des Systems vorhanden ist, wird es im Folgenden nicht näher betrachtet.

5.1.1 Managed Resource

Eine Managed Resource ist im Wesentlichen äquivalent zu einer Komponente, die in gewöhnlichen, nichtautonomen Systemen Anwendung findet. Sie kann jedoch für das Monitoring und die Kontrolle durch einen Autonomic Manager angepasst werden. Managed Resources können sowohl einzelne Hardwareressourcen, wie Speicher, CPU oder Drucker, sowie einzelne Softwareressourcen, wie Datenbanken oder Verzeichnisdienste, oder aber eine Sammlung von Ressourcen, wie Serverpools oder ganzen Unternehmensanwendungen, sein. Zusätzlich können bereits auf dieser Schicht Kontrollschleifen für ein Selbst-Management in den Managed Resources enthalten sein. Die Details dieser Kontrollschleife sind jedoch meist anbieterabhängig und nach außen hin nicht sichtbar.

5.1.2 Touchpoint

Ein Touchpoint implementiert das Verhalten von Sensor und Effektor in einem Manageability Interface für meist mehrere nach außen angebotene Methoden von Managed Resources. Der Sensor bietet dabei Methoden für die Sammlung von Informationen über den Zustand einer Managed Resource an, der Effektor Methoden für die Veränderung von Zuständen der Managed Resource. Die Manageability Interfaces verwenden für die Bereitstellung dieser Methoden Mechanismen wie Logdateien, Events, Befehle, APIs oder Konfigurationsdateien. Diese Mechanismen unterstützen verschiedene Wege, um Details wie Identifikationen, Zustände, Metriken, Konfigurationen oder auch Beziehungen zu anderen Managed Resources zu sammeln, sowie das Verhalten oder die Zustände einer Managed Resource zu verändern.

5.1.3 Touchpoint Autonomic Manager

Autonomic Managers implementieren intelligente Kontrollschleifen, welche verschiedene Kombinationen von Aufgaben automatisieren und dadurch Selbst-x Eigenschaften ermöglichen. Touchpoint Autonomic Managers sind dabei diejenigen, welche direkt mit einer oder mehreren Managed Resources über deren jeweiligen Touchpoint zusammenarbeiten. Das Konstrukt aus einem Autonomic Manager und seinen Managed Resources wird auch als Autonomic Element bezeichnet. Autonomic Managers verwenden Policies, welche unter anderem Ziele für eine Kontrollschleife vorgeben. Autonomic Managers können verschiedene Anordnungen von Managed Resources kontrollieren, z. B. nur eine einzelne Ressource, eine homogene oder heterogene Gruppe von Ressourcen oder aber auch eine Sammlung von heterogenen Gruppen als gesamtes Businesssystem. Autonomic Managers jeglicher Art besitzen ähnlich den Managed Resources ebenfalls einen Sensor und einen Effektor, so dass deren orchestrierende Autonomic Managers mit ihnen interagieren können. Letztere verwenden dazu eine Schnittstelle, welche ähnlich dem Manageability Interface einer Managed Resource ist.

5.1.4 Orchestrating Autonomic Manager

Ein einzelner Touchpoint Autonomic Manager, welcher in Isolation arbeitet, kann nur ein autonomes Verhalten für die Ressourcen erzeugen, welche er kontrolliert. Orchestrierende Autonomic Managers hingegen koordinieren das Verhalten der einzelnen Touchpoint Autonomic Managers und erreichen durch Interaktionen mit diesen und anderen orchestrierenden

Autonomic Managers ein systemweites Selbst-x Verhalten. Dabei kann unterschieden werden, ob die orchestrierenden Autonomic Manager die Touchpoint Autonomic Manager nach ihren Fähigkeiten gruppiert oder über ihre Fähigkeiten hinweg kontrollieren.

5.2 Funktionsweise eines Autonomic Managers

Da ein Autonomic Manager der zentrale Bestandteil für die technische Erzeugung von Selbst-x Eigenschaften ist, wird für die gezielte Entwicklung der Eigenschaften die Funktionsweise im Inneren eines Autonomic Managers noch detaillierter betrachtet.

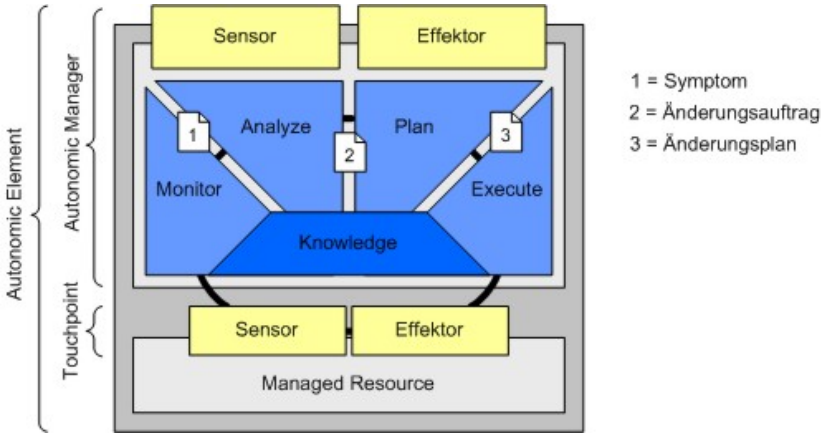

Abbildung 5.2: Logischer Aufbau eines Autonomic Elements

Wie bereits erwähnt, implementiert ein Autonomic Manager eine intelligente Kontrollschleife. Um ein Selbst-Management zu ermöglichen, muss diese Kontrollschleife automatisiert werden, das bedeutet, der Autonomic Manager benötigt automatische Mechanismen, welche die benötigten Details des Systems sammeln, die Details auf Notwendigkeiten für Veränderungen analysieren, Pläne oder Aktionssequenzen für diese Veränderung generieren und darin enthaltene Aktionen durchführen. Aus diesem Grund wird ein Autonomic Manager in die vier Komponenten Beobachtung (*Monitor*), Analyse (*Analyze*), Planung (*Plan*) und Ausführung (*Execute*) untergliedert, welche sich eine gemeinsame Wissensbasis (*Knowledge*) teilen und durch ihre Zusammenarbeit die Kontrollschleife (dicke Linie) bilden (siehe Abbildung 5.2). Die Darstellung gibt dabei mehr die strukturelle Anordnung als einen strikten Kontrollfluss wieder. Durch das Anbieten von Sensor und Effektor (oben) ähnlich denen der Touchpoints einer Managed Resource (unten) können die Autonomic Managers in einer verteilten Infrastruktur zusammengefügt werden und ermöglichen es so orchestrierenden Autonomic Managers, deren Managementfunktionen zu erfüllen.

Obwohl ein Autonomic Manager die vier Komponenten der Kontrollschleife automatisieren kann, kann ein Entwickler einen Autonomic Manager so konfigurieren, dass dieser nur bestimmte Bereiche seiner automatisierten Kontrollschleife ausführen soll. So kann ein Autonomic Manager beispielsweise auf das Monitoring beschränkt werden, so dass er nur Nachrichten über beobachtete Situationen an eine andere Instanz meldet, ohne weitere Schritte der Kontrollschleife zu tätigen. Durch andere Konfigurationen können weitere Bereiche der Kontrollschleife zugelassen werden (vgl. [56]). Jede der fünf Komponenten eines Autonomic Managers hat dabei seine eigene Funktion bzw. Aufgabe:

Monitorfunktion Der Monitor eines Autonomic Managers sammelt in erster Linie relevante Details (Topologieinformationen, Metriken, Konfigurationseinstellungen, ...) über Managed Resources, welche von da ab auch anderen Funktionen zur Verfügung stehen. Die Informationssammlung geschieht entweder durch das Empfangen eines Events, welches die Managed Resource über ihren Touchpoint an den Autonomic Manager schickt (*receive-notification style*), oder durch den Aufruf von angebotenen „get"-Methoden auf dem Touchpoint einer Manager Resource (*retrieve-state style*). Die aus den Details gewonnenen Daten (Status, Konfiguration, angebotene Kapazität, Durchsatz, ...) werden aggregiert und gefiltert, bis ein Symptom festgestellt wird, welches analysiert werden muss. Ein ermitteltes Symptom wird an die Analyse weitergereicht.

Analysefunktion Die Analyse bietet auf Basis der Symptome Mechanismen für die Analyse von Situationen an, um zu bestimmen, ob bestimmte Änderungen vorgenommen werden müssen. Eine Notwendigkeit für eine Änderung kann festgestellt werden, wenn beispielsweise gültige Policies nicht mehr erfüllt werden. Die Analyse ist auch für die Erhaltung von Policies verantwortlich. Sollten Änderungen notwendig sein, übergibt die Analyse einen Änderungsauftrag an die Planung. Der Änderungsauftrag beschreibt die Modifikationen, welche die Analysekomponente für notwendig oder wünschenswert hält. In vielen Fällen modelliert die Analysefunktion komplexes Verhalten, so dass sie Vorhersagetechniken verwendet, um über ihre IT-Umgebung zu lernen und zukünftiges Verhalten vorhersagen zu können.

Planungsfunktion Die Planung wählt oder erstellt aufgrund des Änderungsauftrags einen passenden Ablauf, um eine gewünschte Modifikation in einer Managed Resource zu erzeugen. Sie kann dabei auf viele Ablaufformen zurückgreifen, angefangen bei einem einzigen Befehl bis hin zu einem kompletten Workflow. Die Planung übergibt darauf einen geeigneten Änderungsplan mit den ermittelten Abläufen an die Ausführung.

Ausführungsfunktion Die Ausführung liefert den Mechanismus für die Koordination der Abläufe im Änderungsplan mittels bestimmter Aktionen. Diese Aktionen werden einer Managed Resource über den Effektor ihres Touchpoints mitgeteilt und dann von ihr ausgeführt (*perform-operation style*). Die Ausführung eines Änderungsplans kann eine Aktualisierung der Wissensbasis des Autonomic Managers bewirken, indem der Effekt einer Aktion vom Monitor beobachtet und gespeichert wird.

Wissensbereitstellung Die von den vier Funktionen verwendeten Informationen werden als gemeinsames Wissen in der Wissensbasis gespeichert. Diese enthält somit Daten wie Topologieinformationen, historische Logs, Metriken, Symptome und Policies. Ein Autonomic Manager kann das verwendete Wissen auf drei verschiedene Arten empfangen: (1) Das Wissen wird über den Effektor eingeführt. Auf diesem Weg kann ein Autonomic Manager z. B. Policies vom Entwickler oder von orchestrierenden Autonomic Managers erhalten. (2) Das Wissen wird durch einen externen Informationsdienst geliefert. Auf diesem Weg kann ein Autonomic Manager z. B. Logdateien mit historischem Wissen über eine Managed Resource erhalten. Die Logdatei kann beispielsweise Einträge mit Events enthalten, welche in einer Komponente oder in einem System früher aufgetreten sind. (3) Ein Autonomic Manager kann das Wissen durch seine Funktionen wie beschrieben selber erstellen. Die Autonomic Computing Referenzarchitektur besitzt dabei drei Arten von Wissen (siehe Tabelle B.1 im Anhang auf Seite 94).

5.3 Konzeptionelle Anforderungen an IT-Systeme für die Ermöglichung eines Selbst-x Verhaltens

Die beschriebene Architektur des Autonomic Computing und die Funktionsweise eines Autonomic Managers ermöglichen die Identifizierung von konzeptionellen Anforderungen an IT-Systeme, um ein systemweites Selbst-x Verhalten zu ermöglichen. Die Anforderungen können in verschiedene Bereiche untergliedert werden:

Systemarchitektur Die Systemarchitektur muss derart gestaltet sein, dass Selbst-x Eigenschaften für das ganze System ermöglicht werden. Dazu müssen Regeln gefunden werden, wie ein System in Managed Resources und Autonomic Managers aufgeteilt werden kann. Eine Herausforderung wird dabei sein, welche Autonomic Managers welche Managed Resources bzw. andere Autonomic Managers verwalten müssen, um bestimmte Selbst-x Eigenschaften zu erhalten.

Informationsaustausch Die gefundenen Managed Resources besitzen Fähigkeiten, um die ihnen gestellten Aufgaben erfüllen zu können. Für die Verwendung in OCS müssen sie jedoch eine Schnittstelle für Autonomic Manager anbieten, durch welche letztere auf diese Fähigkeiten zugreifen können. Dazu gehören auch Mechanismen, welche den Austausch von Nachrichten möglich machen. Dasselbe gilt für den Nachrichtenaustausch zwischen den internen Komponenten eines Autonomic Managers bzw. den Austausch von Nachrichten zwischen mehreren Autonomic Managers.

Interaktionen Für die Kommunikation zwischen den einzelnen Systemkomponenten müssen Modelle und Protokolle entwickelt werden, welche den Austausch von Informationen, Events und Wissen definieren. Benötigt werden dabei Standards, durch welche Autonomic Manager auch aus verschiedensten Domänen miteinander kommunizieren können. Dabei müssen auch theoretische Grundlagen für die Kommunikation zwischen den Autonomic Managers entwickelt werden, unter welchen Bedingungen bilaterale, multilaterale oder verkettete Verhandlungen angewendet werden sollen und welche Auswirkungen diese Interaktionen haben.

Monitorfunktion Ein Monitor benötigt Regeln, welche Daten er zu beobachten hat. Es müssen Wege gefunden werden, Symptome zu definieren, so dass ein Monitor beobachtete Daten in diese unterteilen kann.

Analysefunktion Die Analyse benötigt Regeln, Metriken, Schwellen und Konfigurationen, um Symptome analysieren zu können und notwendige Aktionen in die Wege zu leiten.

Planungsfunktion Die Planung muss mit initialen Plänen ausgestattet werden, um erhaltene Änderungsanforderungen umsetzen zu können. Dazu werden auch Informationen über die Fähigkeiten einer Managed Resource benötigt.

6. Entwicklungsansatz für Organic Computing Systeme

Die Untersuchungen in den Kapiteln 3, 4 und 5 haben zum einen gezeigt, wie stigmergische IT-Systeme konstruiert sein müssen, um Charakteristika der Selbstorganisation aufzuweisen, zum anderen, welche konzeptionellen Anforderungen an IT-Systeme gestellt werden, um ein Selbst-x Verhalten zu ermöglichen. Aufbauend auf diesen Ergebnissen wird in diesem Kapitel ein Metamodell für Organic Computing Systeme beschrieben, welches die architektonischen Anforderungen an stigmergische IT-Systeme mit Selbst-x Eigenschaften erfüllt. Dieses Metamodell dient als Fundament des modellbasierten Entwicklungsprozesses für Organic Computing Systeme.

Der Entwicklungsprozess ermöglicht es folglich, stigmergische Organic Computing Systeme zu entwerfen. Dies wird exemplarisch an einer vereinfachten Entwicklung eines stigmergischen Produktionsplanungs- und -kontrollsystems gezeigt, welches annähernd dieselben Grundfunktionalitäten und Eigenschaften besitzt, wie das System aus Fallstudie I[1]. Im Gegensatz zum Originalsystem wird das hier entwickelte System jedoch zusätzlich über Selbst-x Eigenschaften verfügen, welche das System zu einem Organic Computing System werden lassen. Anhand dieses Beispiels wird der Entwicklungsprozess und die verwendete Modellierungssprache erklärt. Abschließend werden Ansätze für Transformationen zwischen verschiedenen Modellen des Entwicklungsprozesses vorgestellt, um eine Unterstützung für das Framework der MDA zu liefern.

6.1 Metamodell für Organic Computing Architekturen

Um die in den letzten Kapiteln identifizierten Anforderungen erfüllen zu können, werden für das OCS-Metamodell unterschiedliche Konzepte verwendet, welche sich in ihrem jeweiligen Anwendungsgebiet bereits bewährt haben. Neu ist hingegen die Kombination dieser Konzepte in einem einzigen Metamodell. Ein Teil der Konzepte entstammt aus Metamodellen und Methodologien der Agententechnologie, da Organic Computing Systeme wie in 2.2 beschrieben

[1] Die Entwicklung wird aus Platzgründen nur die wesentlichsten Funktionen berücksichtigen und von Details abstrahieren

Ähnlichkeiten mit diesen Systemen aufweisen. Ein anderer Teil der Konzepte fließt aus dem Autonomic Computing mit ein, um die Anforderungen an ein Selbst-x Verhalten zu erfüllen.

Für die Identifizierung geeigneter Konzepte aus der Agententechnologie wurden einige Ansätze aus 2.2 miteinander verglichen. Ein Vergleich der Metamodelle von ADELFE, Gaia und PASSI findet sich auch bei Bernon [17]. Sie fasst die ihr am besten erscheinenden Konzepte – für ein umfassendes Agentenmetamodell und nicht für ein OCS-Metamodell – der drei Methodologien zu einem einzigen Metamodell zusammen. Aufgrund der enormen Größe des entstandenen Metamodells kommt sie zu dem Schluss, dass eine Entwicklungsmethodologie für solch ein Modell nicht oder nur in einzelnen Teilen möglich ist. Das OCS-Metamodell verwendet aus diesem Grund nur die wirklich notwendigen Konzepte, so dass alle gestellten Anforderungen erfüllt werden.

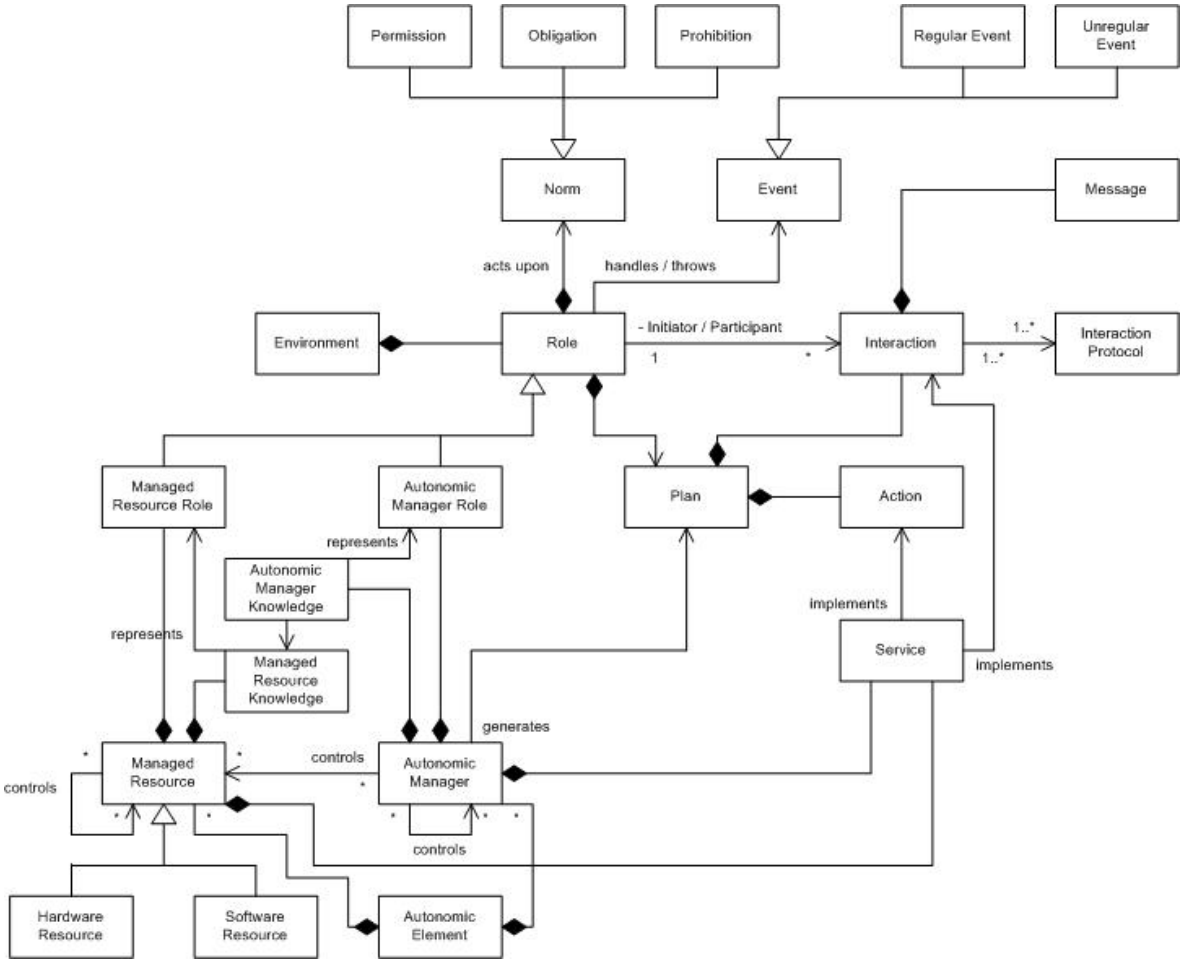

Abbildung 6.1: Metamodell für OCS-Architekturen

Im OCS-Metamodell (siehe Abbildung 6.1) ist wie in vielen Agentenmethodologien (vgl. [60, 2, 32, 106]) die Rolle (*Role*) das zentrale Element. Alle Rollen ergeben gemeinsam die gesamte Umgebung (*Environment*) des Systems. Eine Rolle kann mit anderen Rollen interagieren (*Interaction*) und tauscht dazu Nachrichten (*Messages*) nach einem spezifizierten Protokoll (*Interaction Protocol*) aus (ähnlich [108]). Die initiierende Rolle wird dabei als *Initiator* bezeichnet, die partizipierende Rolle als *Participant*. Die Interaktion kann dann je nach

Protokoll durch eine direkte Kommunikation zwischen zwei Rollen als auch indirekt durch die Kommunikation über die Umgebung (Stigmergie) realisiert werden.

Rollen werden in *Managed Resource Roles* bzw. *Autonomic Manager Roles* unterteilt, welche dann von *Managed Resources* bzw. *Autonomic Managers* übernommen werden. Dabei kann ein Autonomic Manager mehrere Autonomic Manager Roles übernehmen und weitere Autonomic Managers bzw. Managed Resources kontrollieren. Managed Resources können ebenfalls mehrere Managed Resource Roles übernehmen und mehrere Managed Resources kontrollieren. Letztere können wiederum in *Hardware* und *Software Resources* unterteilt werden. Managed Resources und Autonomic Managers ergeben Autonomic Elements. Dabei kann ein Autonomic Element aus einer oder mehreren Managed Resource, einem oder mehreren Autonomic Managers oder auch aus einer Mischung beider Konzepte bestehen. Eine Managed Resource muss infolgedessen nicht immer von einem Autonomic Manager kontrolliert werden. Dieses Konzept entspricht der Referenzarchitektur des Autonomic Computing und schafft so die Basis für Selbst-x Eigenschaften.

Sowohl *Managed Resources* als auch *Autonomic Managers* besitzen Wissen. Dieses Wissen enthält im Fall des *Managed Resource Knowledge* Repräsentationen von Vorstellungen über sich selber, ihre Umgebung sowie andere Managed Resource Roles. Das *Autonomic Manager Knowledge* enthält zusätzlich noch das Wissen über kontrollierte Managed Resources (Selbst-Konfiguration). Das Wissenskonzept entspricht sowohl dem Konzept von *beliefs* [18] in Agenten als auch dem des Autonomic Computing.

Das Verhalten einer Rolle ist durch Normen (*Norms*) festgelegt. Dieses Konzept ist der normativen Agentenarchitektur (NoA) [64] entnommen, welche erstellt wurde, um die Entwicklung von Agentengemeinschaften zu unterstützen, welche von Normen anstatt von Zielen geleitet werden. Normen können Obligationen (*Obligations*), Erlaubnisse (*Permissions*) oder Verbote (*Prohibitions*) sein. Obligations motivieren Agenten entweder einen Zustand zu erreichen oder eine bestimmte Aktion auszuführen. Prohibitions schränken das Verhalten eines Agenten ein, wohingegen Permissions es einem Agenten erlauben, bestimmte Aktivitäten zu tätigen. Durch die Verwendung dieses Konzepts kann das Verhalten einer Rolle im Gegensatz zur einfachen Verwendung von Zielen noch feiner gesteuert werden.

Der Lebenszyklus einer Rolle entspricht trotz des Konzepts der Normen dennoch dem klassischen Lebenszyklus von Rollen bei Agenten. Eine Rolle erkennt Situationen, trifft Entscheidungen und führt Aktionen aus. Das Erkennen von Situationen beruht in diesem Metamodell auf Ereignissen (*Events*). Dieses Konzept ist für die Verwendung in einem Metamodell im Vergleich zu den Agentenmethodologien neu und nimmt zusätzlich das Konzept der so genannten NCS (*Non-Cooperative Situations*) aus [18] und die Funktionsweise der Kommunikation einer Managed Resource und einer Autonomic Managers im Autonomic Computing mit auf. Events können gewöhnlich (*Regular Events*) oder ungewöhnlich sein (*Unregular Events*). Gewöhnliche Events sind solche, welche eine Rolle von Anfang an besitzt oder durch Lernen (Adaptivität) hinzugewinnt. Sie werden von einer Rolle ähnlich dem *Publish-Subribe*-Pattern [25] geworfen und von einer anderen oder derselben Rolle empfangen und behandelt. Aufgrund des Empfanges eines Events können Normen aktiviert oder deaktiviert werden. Events können das Eintreffen einer Nachricht signalisieren, das Erreichen eines Zustands melden oder auch aufgrund einer unbekannten Situation oder Fehlers ausgelöst werden. Im letzten Fall kommt dies dem Werfen einer Exception wie in Java [8] gleich und wird als ungewöhnliches Event eingeordnet. Ungewöhnliche Events könnten in verschiedene weitere Kategorien unter-

teilt werden[2]. Dazu gehören Teile der NCS von ADELFE, wie *Incromprehension* (eine Rolle versteht eine empfangene Nachricht nicht), *Ambiguity* (eine Rolle hat mehrere Interpretationsmöglichkeiten für eine empfangene Nachricht), *Incompetence* (eine Rolle kann die Anfrage einer anderen Rolle nicht erfüllen), *Unproductivness* (eine Rolle erhält eine bereits bekannte Information oder eine Information, welche ihr nichts nützt), *Concurrence* (mehrere Rollen wollen gleichzeitig Zugriff auf denselben kritischen Bereich einer Rolle erhalten), *Uselessness* (eine Rolle führt Aktionen durch, welche zu nichts führen).

Darüber hinaus können ungewöhnliche Events *Internal Exceptions* sein, falls eine Fehlfunktion innerhalb einer Rolle vorliegt (Selbst-Heilung), oder *Intrusion Detections*, falls ein Angriff von außen stattfindet (Selbst-Schutz). Dazu können beliebig viele weitere Events definiert oder erlernt werden. In allen Fällen ist eine Rolle dazu gezwungen, ein empfangenes Event zu kategorisieren (ähnlich den Symptomen im Autonomic Computing) und nach bestehenden Normen und Plänen zu behandeln oder bei Bedarf neu Pläne zu generieren, um die Situation zu lösen (Adaptivität).

Pläne bestehen aus Aktionen (*Actions*) – die internen Aktivitäten einer Rolle – und Interaktionen – die externen Aktivitäten einer Rolle. Die explizite Modellierung von Zielen und Plänen fehlt in ADELFE, Gaia und PASSI, ist aber in anderen Methodologien relevant (MASSAGE, Tropos). Für die Adaptivität von OCS ist die Modellierung von Plänen allerdings unvermeidlich (Selbst-Optimierung). Pläne helfen bei der Erreichung von Zielen in Normen und haben bestimmte Effekte (siehe [64]). Die Aktionen und Interaktionen einer Rolle werden durch Dienste (*Services*) eines Autonomic Managers oder einer Managed Resource implementiert.

Durch das OCS-Metamodell werden die identifizierten Anforderungen erfüllt. Das Konzept der Rollen unterstützt zusätzlich die Selbst-x Eigenschaften des Systems. Würden die Managed Resources und Autonomic Managers ohne Rollen direkt die gesamte Umgebung bilden, gäbe es z. B. zur Laufzeit keine Möglichkeit, eine Rolle eines Autonomic Managers auf einen anderen Autonomic Manager zu übertragen. Beim Ausfall eines Autonomic Managers oder einer Managed Resource wäre das System nicht mehr voll funktionsfähig. Ist es jedoch möglich, die Rollen zur Laufzeit auf andere Elemente zu übertragen, ist nicht nur die Selbst-Heilung gewährleistet.

6.2 Entwicklungsprozess

Ein Softwareentwicklungsprozess ist definiert als ein Prozess, d. h. eine Sequenz von Schritten, durch den Benutzeranforderungen in ein Softwareprodukt umgewandelt werden (vgl. [57]). Der folgende Entwicklungsprozess definiert eine Sequenz von Schritten (Aktivitäten), welche für die Entwicklung von Organic Computing Systemen, die auf dem OCS-Metamodell (siehe Abbildung 6.1) basieren, ausgeführt werden müssen. In Klammern sind jeweils die Modelle angegeben, welche nach Ausführung einer Aktivität entstehen. Die Analysephase wird durch die Aktivitäten 1–5, die Designphase durch die Aktivitäten 6–19 gebildet. Eine Implementierungsphase ist in dieser Arbeit noch nicht berücksichtigt, kann aber in Zukunft ohne Weiteres an die bestehenden Aktivitäten angehängt werden. Die Abfolge der Schritte ist unabhängig vom Vorgehensmodell und kann beispielsweise durch das Wasserfallmodell oder den Rational Unified Process (RUP) [6] realisiert werden.

[2] Aus Platzgründen nicht dargestellt

Die Aktivitäten des Entwicklungsprozesses sind im Einzelnen:

1. Definition des Geschäftsumfeldes (Business Context Model)
2. Bestimmung der zu unterstützenden Geschäftsprozesse (Business Process Model)
3. Charakterisierung der Umgebung (Environment Model)
4. Aufstellung von Anwendungsfällen (Use Case Model)
5. Zusammenfassen des Grundwortschatzes (Ontology Model)
6. Identifizierung von Managed Resource Rollen (Role Model)
7. Spezifikation von Normen für Managed Resource Rollen (Norm Model)
8. Entwicklung von Plänen für Managed Resource Rollen (Plan Model)
9. Identifizierung der Interaktionspattern zwischen Managed Resource Rollen (Interaction Model)
10. Identifizierung der Dienste für Managed Resource Rollen (Service Model)
11. Identifizierung von Autonomic Manager Rollen (Autonomic Manager Role Model)
12. Spezifikation von Normen für Autonomic Manager Rollen (Autonomic Manager Norm Model)
13. Entwicklung der Analyse für Autonomic Manager Rollen (Autonomic Manager Analyze Model)
14. Entwicklung von Plänen für Autonomic Manager Rollen (Autonomic Manager Plan Model)
15. Identifizierung der Interaktionspattern zwischen Autonomic Manager Rollen und weiteren Rollen (Autonomic Manager Interaction Model)
16. Identifizierung der Dienste von Autonomic Manager Rollen (Autonomic Manager Service Model)
17. Entwicklung von Interaktionsprotokollen (Interaction Protocol Model)
18. Identifizierung von Autonomic Elements (Autonomic Element Model)
19. Verteilung der Autonomic Elements (Autonomic Elements Instance Model)

Von den Methodologien für Agentensysteme besitzt lediglich ADELFE einen konsistenten Entwicklungsprozess, alle anderen Ansätze modellieren nur Fragmente eines Systems. Das Vorkommen bestimmter Modelle in Agentenmethodologien hängt davon ab, welche Konzepte das jeweilige Metamodell der Methodologie verwendet, jedoch entält keine Methodologie Modelle der Aktivitäten 11-19, da bekanntlich noch keine Methodologie für die Entwicklung von Selbst-x Eigenschaften existiert. Wie zu erkennen ist, wird jeder Aspekt des OCS-Metamodells von mindestens einer Aktivität im Entwicklungsprozess berücksichtigt.

6.3 Modelle und Notationen

Die durch den Entwicklungsprozess entstehenden Modelle basieren so weit wie möglich auf UML 2.0[3]. Für manche Modellierungsaspekte reicht dieser Standard allerdings nicht aus, weswegen in diesen Fällen eine eigene Notation definiert wird, die zum Teil bereits für die Modellierung von Konzepten in Agentenmethodologien verwendet wird, zum Teil aber auch völlig neu ist. Die Aufteilung der Modelle auf die Ebenen der MDA und deren Zusammenhang ist in Abbildung 6.2 dargestellt.

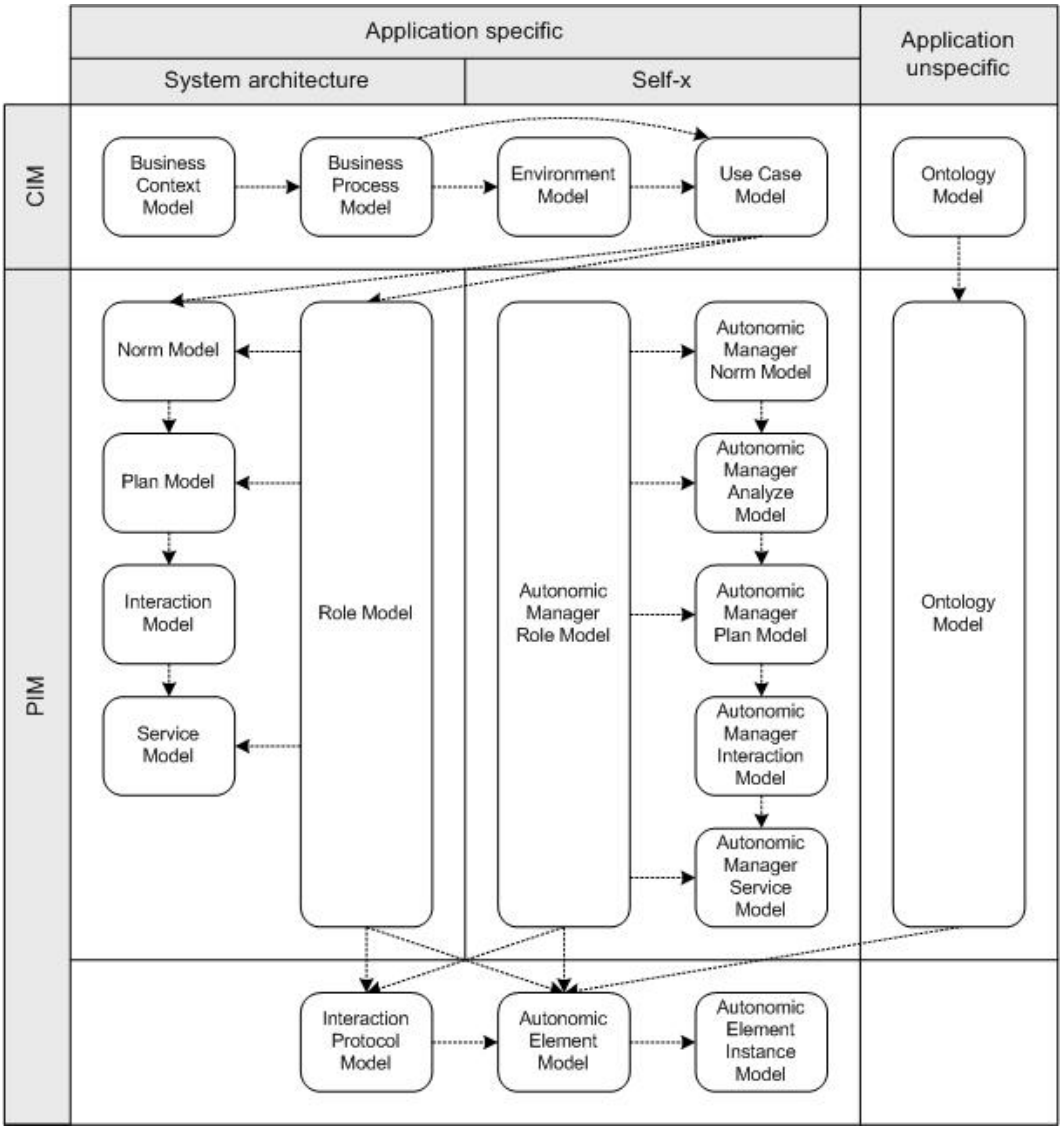

Abbildung 6.2: Modelle des Entwicklungsprozesses für Organic Computing Systeme

Mit der Definition der ersten fünf Aktivitäten des Entwicklungsprozesses bzw. deren Modelle im CIM werden so viele Informationen wie möglich für die weitere Verwendung in tieferen Schichten verfügbar gemacht, ohne auf Details eines Systems einzugehen. Die Untergliederung in einen applikationsspezifischen bzw. domänenspezifischen Modellierungsteil ist aus Gründen der Wiederverwendbarkeit entstanden, so dass domänenspezifische Modelle auch für andere Applikationsentwicklungen genutzt werden können. Die Modelle im PIM verwenden Informa-

[3]Da noch keine Spezifikation der OMG für UML 2.0 verfügbar ist, wird [59] als Grundlage verwendet

tionen aus dem CIM, um das System unabhängig von der am Ende eingesetzten Technologie und bis Aktivität 16 zunächst auch unabhängig von der Art der Kommunikation zwischen den einzelnen Systemkomponenten zu modellieren. Die Kommunikationsart wird erst in Aktivität 17 festgelegt. Aufgrund der noch nicht berücksichtigten Implementierungsphase existieren im PSM keine Modelle. Vorstellbar wären hier Modelle, welche auf den vorgestellten Technologien in 4.2.2 und 5.1 basieren.

Die gestrichelten Linien in Abbildung 6.2 geben an, welche Informationsflüsse zwischen den Modellen bestehen. Dabei bedeutet ein Pfeil von Model A nach Model B, dass Informationen aus A direkt in B verwendet werden bzw. nach B transformiert werden können. Ansätze für diese Transformationen werden in 6.4 vorgestellt. Im Folgenden werden die Modelle des Entwicklungsprozesses und die darin verwendeten Notationssymbole ausführlich erklärt und die Informationsflüsse zwischen den Modellen informell beschrieben.

6.3.1 Business Context Model

Die Definition des Geschäftsumfeldes geschieht im Business Context Model. Dabei wird die Einbettung des zukünftigen Systems in den Kontext des gesamten Unternehmens beschrieben. Das Modell abstrahiert von konkreten Geschäftsprozessen und betrachtet nur den groben Zusammenhang von übergeordneten Prozessen, zumeist von funktionalen Einheiten des Unternehmens. Dieses Modell dient lediglich dem Verständnis für Entwickler und hat nur geringen Verwendungszweck für die folgenden Modelle. Die Modellierung erfolgt in einem Aktivitätsdiagramm.

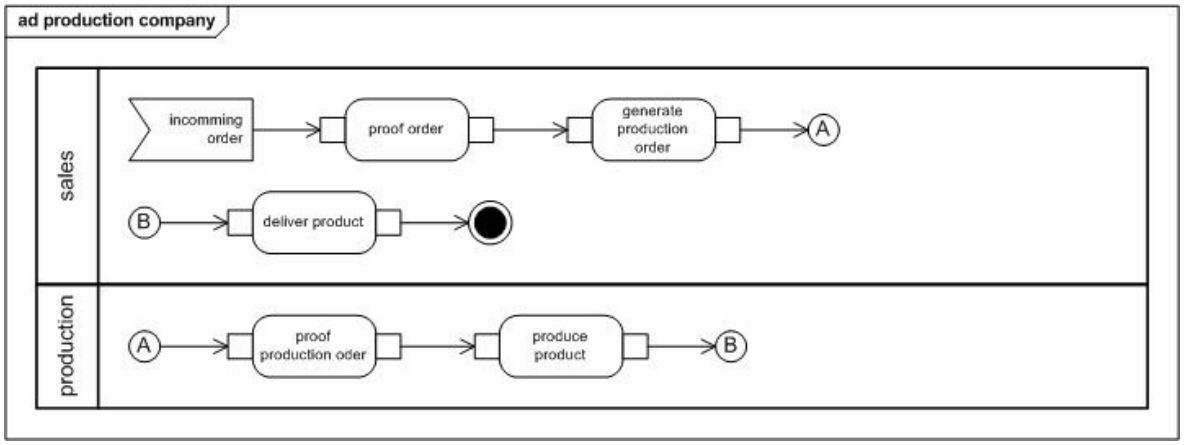

Abbildung 6.3: Business Context Model

Wie bereits erwähnt, soll parallel zur Beschreibung der Modelle und Notation illustrierend ein Produktionsplanungs- und -kontrollsystem (*PPCS*) entwickelt werden, welches ähnlich dem System aus Fallstudie I ist. Da von dem analysierten System nur Informationen über die Systemarchitektur und die Kommunikationsmechanismen vorliegen, werden die Modelle im CIM nach bestem Wissen modelliert. Abbildung 6.3 zeigt einen Ausschnitt aus dem möglichen Business Context Model eines produzierenden Unternehmens. Das Modell beschreibt zwei Unternehmensbereiche, Vertrieb (*sales*) und Produktion (*production*). Trifft ein Auftrag im Vertrieb ein (*incoming order*), so wird der Auftrag geprüft (*proof order*), eine Produktionsauftrag erstellt (*generate production order*) und dieser an die Produktion weitergeleitet. Diese prüft den Produktionsauftrag (*proof production order*), erstellt das Produkt (*produce product*) und leitet es an den Vertrieb zur Auslieferung weiter (*deliver product*).

6.3.2 Business Process Model

Das Business Process Model beschreibt den Zusammenhang der Geschäftsprozesse, welche durch ein zu entwickelndes OCS unterstützt werden sollen, indem es bestimmte (Teil-)Prozesse aus dem Business Context Model verfeinert. Die Geschäftsprozesse werden dabei den ausführenden Akteuren zugeordnet. Zu beachten ist, dass die Modellierung zwar auf die Unterstützung eines IT-Systems hin abzielt, jedoch noch keine Details des Systems aufzeigt. Die Modellierung erfolgt ebenfalls durch ein Aktivitätsdiagramm.

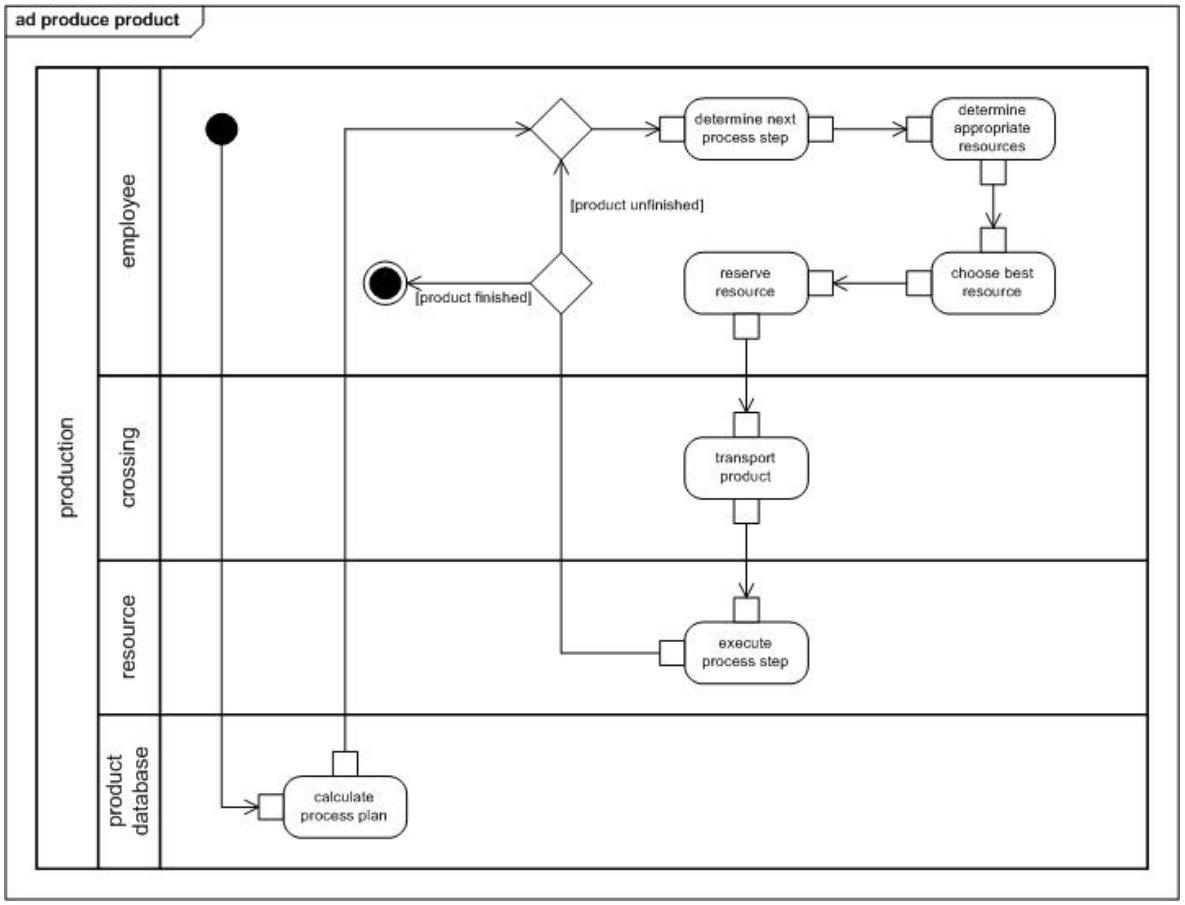

Abbildung 6.4: Business Process Model als Verfeinerung des Prozesses *produce product*

Das in Abbildung 6.4 gezeigte Business Process Model verfeinert den Prozess *produce product* aus dem Business Context Model für die Unterstützung durch ein PPCS. Es untergliedert die Produktionsabteilung in die Partitionen Angestellter (*employee*), Kreuzung (*crossing*), Ressource (*Resource*) und Produktdatenbank (*product database*), in welcher die Prozesspläne für alle Produkte des Unternehmens hinterlegt sind. Die Produktion eines Produktes beginnt mir dem Berechnen des Prozessplans (*calculate process plan*) für das zu erstellende Produkt, das die Produktdatenbank auf Geheiß des Angestellten hin übernimmt. Auf Basis dieses Prozessplans wird anschließend vom Angestellten der nächste, auszuführende Prozessschritt ermittelt (*determine next process step*) und alle Ressourcen bestimmt (*determine appropriate resources*), welche diesen Prozessschritt ausführen können. Liegt die Auswertung vor, wird die am besten bewertete Ressource nach hier nicht näher bestimmten Kriterien ausgewählt (*choose best resource*) und für die Ausführung reserviert (*reserve resource*). Nachdem die Ressource reserviert ist, wird das unfertige Produkt über Transportbänder zu dieser Ressource

transportiert(*transport product*) und dort entsprechend der Reservierung bearbeitet (*execute process step*). Da die Steuerung der Transportbänder von Kreuzungen übernommen wird, enthält diese Partition die Aktivität. Ist nach der Beendigung der Ausführung das Produkt fertig gestellt, so ist der Prozess *produce product* beendet. Andernfalls wird wieder mit der Berechnung des nächsten, auszuführenden Prozessschritts begonnen.

6.3.3 Environment Model

Das Environment Model charakterisiert Objekte genauer, welche für das zu entwickelnde System relevant sind. Diese Objekte können von beliebiger Natur sein (Mensch, Maschine, Datenbank, ...) und werden unter anderem aus dem Business Process Model abgeleitet. Alle darin enthaltenen Unterpartitionen im Aktivitätsdiagramm werden zu Klassen im Environment Model. Durch die Verwendung eines Klassendiagramms zur Modellierung können detaillierte Informationen zu einem Objekt hinzugefügt werden.

Das Environment Model in Abbildung 6.5 enthält einen Angestellten (*Employee*) mit einem zugeteilten Produktionsauftrag (*AssignedProductionOrder*), welchen er zeitlich und fachlich korrekt erfüllen muss. Der Produktionsauftrag wird auf einer oder mehreren Ressourcen (*Resource*) ausgeführt, welche beispielsweise über eine Kapazitätsgrenze (*capacity*), ein Auslastungsprofil (*loadProfile*) und verschiedene Produktionsfähigkeiten (*capabilities*) verfügen. *Resources* sind über ein Transportband (*Conveyour*) mit einer Kreuzung (*Crossing*) verbunden, welche über weitere Transportbänder wiederum mit anderen *Resources* verbunden ist. Zu erkennen ist, dass die Klassen *AssignedProductionOrder* und *Conveyour* nicht aus dem Business Process Model abgeleitet werden konnten, über den Zusammenhang der Objekte in der Umgebung des Systems jedoch aufschlussreich sind.

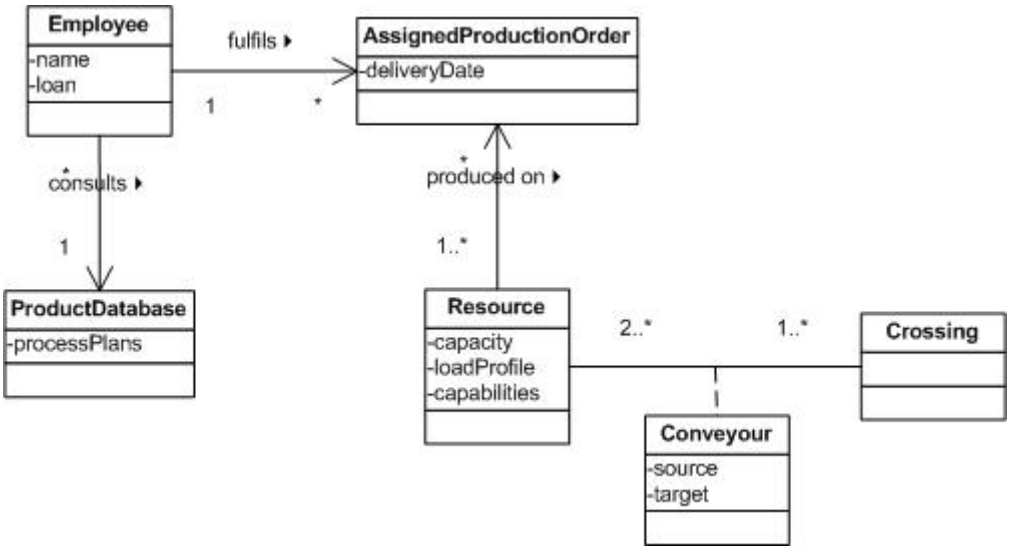

Abbildung 6.5: Environment Model

6.3.4 Use Case Model

Das Use Case Model beschreibt abstrakt den Nutzen des Systems für die Umgebung und definiert die Verwendung des Systems durch Anwendungsfälle. Dabei werden bereits primäre Akteure mit Anwendungsfällen und daran beteiligten sekundären Akteuren identifiziert. Die Modellierung verwendet ein Use Case Diagramm.

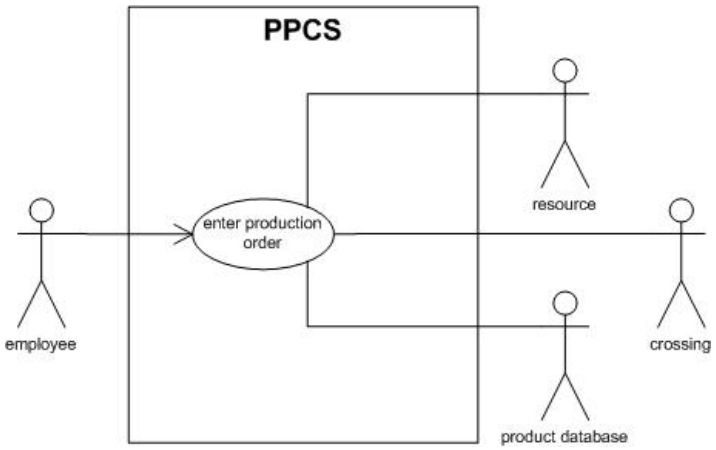

Abbildung 6.6: Use Case Model

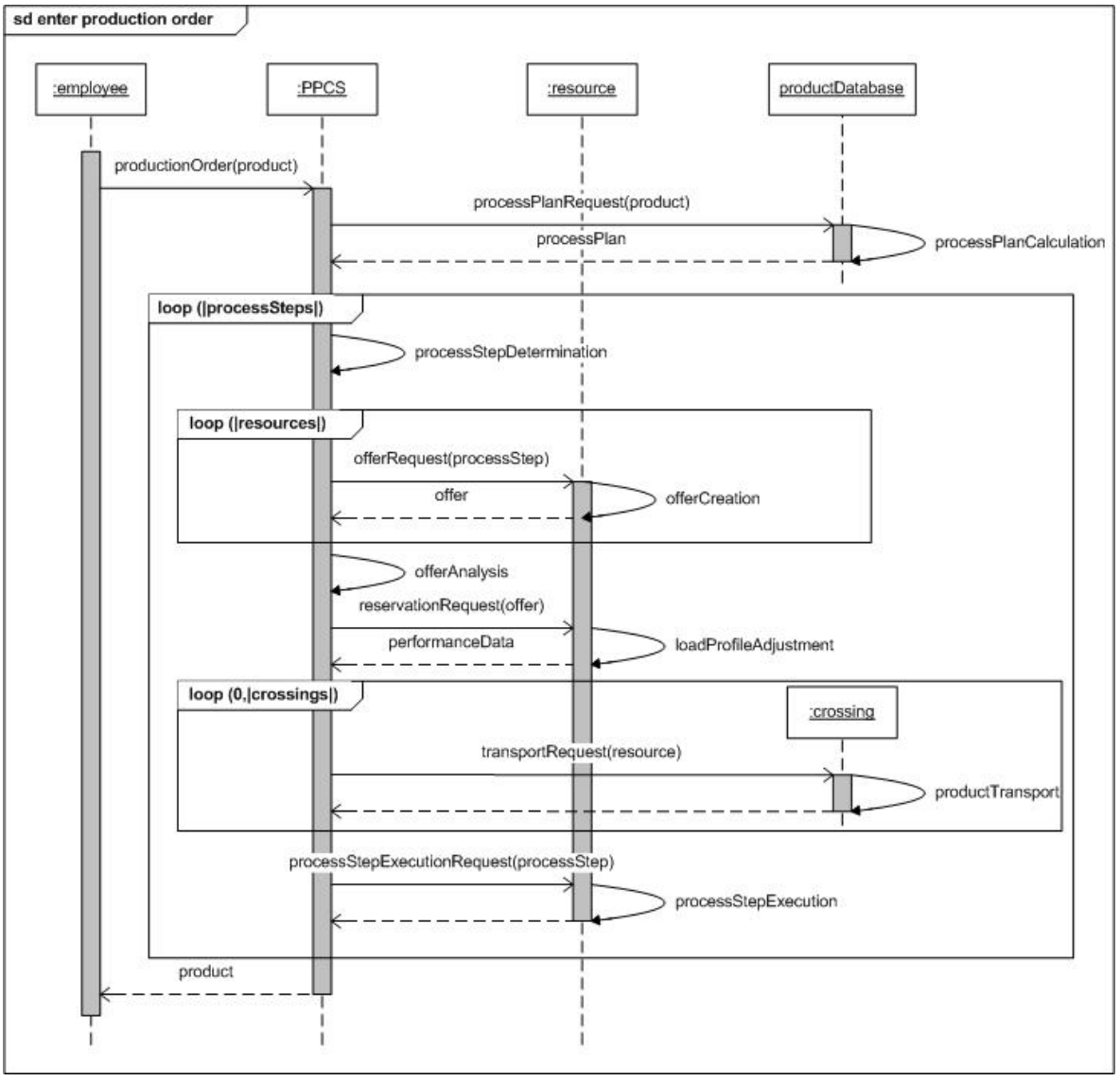

Abbildung 6.7: Sequenzdiagramm zur Verfeinerung des Use Case Models

Das Use Case Model aus Abbildung 6.6 zeigt den einzigen Anwendungsfall für ein gewünschtes PPCS, das Eingeben eines Produktionsauftrages (*enter production order*) durch den Angestellten. Die durch diesen Anwendungsfall betroffenen sekundären Akteure sind die Ressourcen, die Kreuzungen sowie die Produktdatenbank. Das Use Case Model kann durch die Beschreibung des Nachrichtenaustausches zwischen den beteiligten Akteuren mittels Sequenzdiagrammen verfeinert werden. Dazu werden die Geschäftsprozesse aus dem Business Process Model wieder aufgegriffen und der dadurch bedingte Nachrichtenaustausch zwischen Akteuren chronologisch aufgegliedert.

Das Sequenzdiagramm in Abbildung 6.7 zeigt den Nachrichtenaustausch zwischen den Akteuren an, so wie er aufgrund der bisherigen Modelle, vor allem des Business Process Model, stattfindet bzw. im PPCS stattfinden soll. Gemäß dem Use Case gibt der Angestellte den *productionOrder* für das gewünschte Produkt in das PPCS ein. Das PPCS sendet daraufhin eine Anfrage (*processPlanRequest*) an die *productDatabase* welche den *processPlan* zusammenstellt (*processPlanCalculation*) und zurückschickt. Solange noch nicht alle Prozessschritte erledigt sind, wird immer wieder der nächste, auszuführende Prozessschritt berechnet (*processStepDetermination*) und eine Anfrage für ein Angebot an alle Ressourcen geschickt, den berechneten Prozessschritt auszuführen (*offerRequest*). Die Ressourcen erstellen ein Angebot (*offerCreation*) und senden es an das PPCS zurück. Darauf hin wird das beste Angebot ausgewählt (*offerAnalysis*) und dafür eine Reservierungsanfrage an die beste Ressource geschickt (*reservationRequest*). Da sich seit der Abgabe des Angebots jedoch die Auslastung verändert haben könnte, schickt die Ressource nach der Anpassung des Auslastungsprofils (*loadProfileAdjustment*) aktuelle Daten (*performanceData*) zurück. Anschließend werden solange Transportanfragen (*transportRequest*) an die Kreuzungen gestellt, bis diese das Produkt über die Transportbänder (hier nicht gezeigt) zu der richtigen Ressource geleitet haben (*productTransport*). An diese wird dann die Aufforderung zur Ausführung des Prozessschrittes gestellt (*processStepExecutionRequest*), welches dann von dieser erfüllt wird (*processStepExecution*). Sobald kein Prozessschritt mehr zu erfüllen ist, wird das Produkt an den Angestellten fertig zurückgeliefert.

6.3.5 Ontology Model

Das Ontology Model dient der Definition des Grundwortschatzes. Hier werden alle notwendigen Wissensbausteine modelliert, welche für weitere Modelle notwendig sind. Dazu gehören beispielsweise Prozesspläne für Produkte, Maschinenfähigkeiten, Auftragsarten, etc. Die Modellierung kann hier durch ein Klassendiagramm erfolgen. Zu beachten ist, dass die im Ontology Model der CIM-Ebene definierten Begriffe nur dem Verständnis der Prozesse und Anforderungen dienen. Auf tieferen Ebenen des Ontology Models können neue Begriffe hinzukommen bzw. auch manche nicht mehr beachtet werden. Für Ontology Modelle erfolgt aufgrund der Trivialität auf werder auf CIM- noch auf PIM-Ebene ein Beispiel.

6.3.6 Role Model

Das Role Model ist das erste Modell der PIM-Ebene und definiert die Managed Resource Rollen, welche aus dem Use Case Model abgeleitet werden können. Die Ableitung der Rollen kann durch eine Abbildung der Objekte aus dem Sequenzdiagramm des Use Case Models erfolgen. Dabei wird jedes Objekt zu einer Rolle mit Ausnahme des zu entwickelnden Systems an sich. Da letzteres nur einen Rahmen vorgibt, wird in diesem Fall die aufrufende Nachricht des Anwendungsfalls zu einer Rolle.

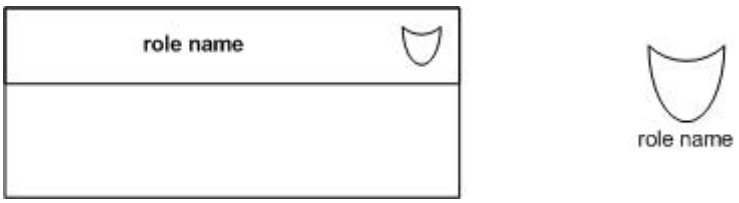

Abbildung 6.8: Notationsmöglichkeiten einer Rolle im Role Model

In UML 2.0 ist weder ein Modell noch eine Notation für Rollen vorgesehen. Daher wird hier ein eigenes Notationssymbol für eine Rolle verwendet, welches auch in [2] angewandt wurde. Abbildung 6.8 zeigt zwei Notationsmöglichkeiten für eine Rolle im Role Model. Die erste Möglichkeit (links) funktioniert ähnlich dem Kompositionsstrukturdiagramm des UML-Standards, indem weitere Eigenschaften oder Attribute innerhalb des Rahmens der Rolle notiert werden können. Die zweite Möglichkeit (rechts) funktioniert ähnlich dem Klassendiagramm des UML-Standards, indem weitere Eigenschaften oder Attribute über Kanten mit der Rolle in Verbindung gebracht werden können. In beiden Fällen wird eine Rolle durch ein schild-ähnliches Symbol repräsentiert, um die Wiedererkennung in Modellen zu erleichtern. Ebenso wäre – wie auch für die Symbole in späteren Modellen – eine Realisierung mit Stereotypen denkbar.

Im Gegensatz zu den meisten Agentenmethodologien erfolgt in diesem Ansatz die Modellierung in allen Modellen durch graphische Notationselemente. Daher musste ein Weg gefunden werden, wie Zusammenhänge über mehrere Modelle hinweg modelliert werden, ohne dass die Übersichtlichkeit oder semantische Informationen verloren gehen. Aus diesem Grund werden Referenzen zwischen Modellen definiert, welche in UML 2.0 bereits möglich sind, jedoch in dieser Art bisher noch nicht verwendet wurden. Für die Verwendung von Referenzen eignet sich die linke Notationsweise einer Rolle besser und wird daher im Folgenden verwendet.

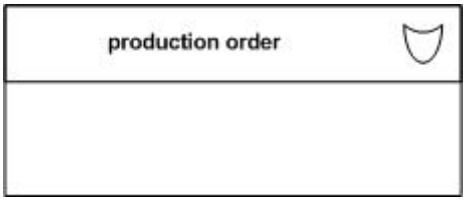

Abbildung 6.9: Rolle *production order* im Role Model

Abbildung 6.9 zeigt die noch leere Rolle *production order* im Role Model für das PPCS. Diese Rolle spiegelt den beschriebenen Fall wider, dass die den Anwendungsfall initiierende Nachricht zur Rolle wird. Abbildung B.1 im Anhang zeigt das Role Model mit allen Rollen, welche für das PPCS identifiziert wurden. Dies sind gerade die Objekte aus dem Sequenzdiagramm aus Abbildung 6.7.

6.3.7 Norm Model

Die Spezifikation von Normen für Rollen geschieht im Norm Model. Die Normen ergeben sich in erster Linie aus den Nachrichten im Sequenzdiagramm des Use Case Models. Diese können noch nicht vollautomatisch in Normen transformiert werden, sondern müssen manuell vervollständigt werden. Jeder interne oder externe Nachrichtenaustausch im Sequenzdiagramm wird dabei zu einer Norm für diejenige Rolle, dessen korrespondierendes Objekt im

Sequenzdiagramm den Austausch initiiert. Vom Entwickler muss eine Norm dann nach logischen Gesichtspunkten gefüllt werden. Dazu ist die Angabe von Zielen (<<*achieve*>>) bzw. auszuführenden Aktionen (<<*perform*>>), von Aktivierungsereignissen (<<*activation*>>) und Deaktivierungsereignissen (<<*expiration*>>) nötig. Diese müssen entsprechend der Informationen aus dem Business Process Model und kausalen Zusammenhängen im Use Case Model geeignet formuliert werden, so dass die Funktionsweise des Systems für diesen Use Case gesichert wird. Da für die Notation einer Norm nach [64] ebenfalls keine Notationsvorschrift in UML 2.0 existiert, wird hier erneut eine eigene Notation definiert (siehe Abbildung 6.10).

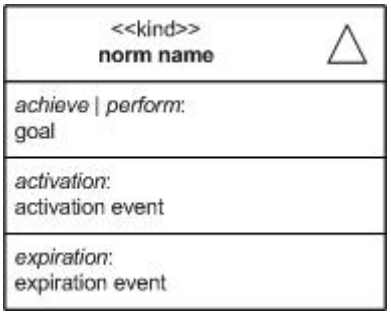

Abbildung 6.10: Notation einer Norm im Norm Model

Als Symbol für eine Norm wird ein stehendes Dreieck verwendet. Die Angabe, ob es sich um eine *Obligation*, *Permission* oder *Prohibition* handelt, erfolgt als Stereotyp über dem Namen der Norm. Innerhalb des Rahmens der Normdarstellung werden Ziel bzw. auszuführende Aktion, Aktivierungs- und Deaktivierungsereignisse angegeben.

Nachdem eine Norm für eine Rolle spezifiziert wird, ändert sich das Role Model entsprechend. Eine Rolle referenziert auf eine Norm, indem der Kopf der Norm innerhalb der Rolle eingefügt wird. Abbildung 6.11 zeigt dies für die Rolle *production order* und der Norm *offer acquisition*. Die Norm selber wird im Norm Model spezifiziert. Hier wird der Vorteil des Referenzierungskonzepts deutlich: Besitzt eine andere Rolle ebenfalls diese Norm, so reicht eine Referenz auf diese Norm aus, ohne die Norm noch einmal neu für die Rolle spezifizieren zu müssen.

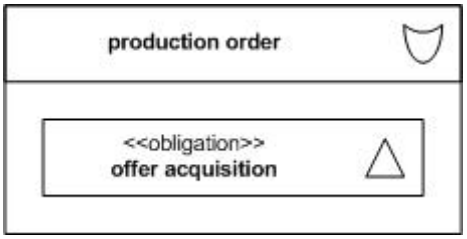

Abbildung 6.11: Rolle *production order* mit Referenz auf die Norm *offer acquisition*

Abbildung 6.12 zeigt die Modellierung der Norm *offer acquisition* der Rolle *production order* im Norm Model. Die Norm ist eine *Obligation* und hat das Ziel, Angebote zur Ausführung von Prozessschritten einzuholen (*offers acquired*). Die Norm wird gemäß dem Business Process Model und dem Use Case Model durch die Beendigung der Prozessschrittberechnung (*process step determined*) aktiviert und durch den Abschluss des Einholens von Angeboten (*offers acquired*) wieder deaktiviert. Das Ziel und die beiden Ereignisse signalisieren dabei Zustände der Rolle.

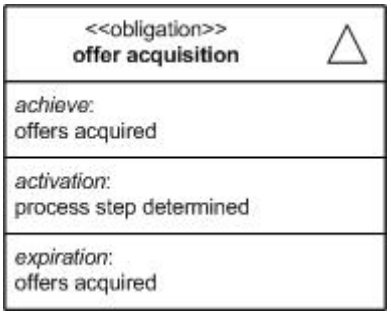

Abbildung 6.12: Norm *offer acquisition* im Norm Model

Abbildung B.2 im Anhang stellt das gesamte Norm Model für das PPCS dar. Zu erkennen ist, dass alle Normen zusammen den kompletten Use Case erfüllen. Die Aktivierungs- und Deaktivierungsereignisse sind so gewählt, dass sich für diese initialen Normen keine Überschneidungen ergeben und der Use Case planmäßig abgearbeitet werden kann. In Abbildung B.3 im Anhang ist das komplette Role Model nach der Spezifikation der Normen dargestellt.

6.3.8 Plan Model

Um die spezifizierten Normen erfüllen zu können, benötigen Rollen initiale Pläne, welche interne Aktionen und Interaktionen mit anderen Rollen enthalten. Rollen können sich zwar später zur Laufzeit neue Pläne selber generieren oder existierende Pläne modifizieren, jedoch müssen sie zur Entwicklungszeit mit initialen Plänen als Basis ausgestattet werden. Die Modellierung der Pläne geschieht im Plan Model, welches auch als Grundlage für die Ableitung des Interaction Models in 6.3.9 und des Service Models in 6.3.10 dient. Für die Entwicklung von Plänen werden wiederum verschiedene Informationen aus vorherigen Aktivitäten verwendet. Die Modellierung der Pläne erfolgt in Aktivitätsdiagrammen, welche so konstruiert werden müssen, dass sie sowohl ein Ziel einer Norm erfüllen, als auch am Ende ein oder mehrere Events senden, welches Aktivierungs- oder Deaktivierungsereignissen in Normen entsprechen. Nur aufgrund dieser Events kann eine aktive Norm deaktiviert bzw. weitere Normen aktiviert werden.

Abbildung 6.13 zeigt einen Ausschnitt aus dem Plan Model mit den Plänen *offer acquiring* bzw. *offer creating*, mit welchen die Norm *offer acquisition* der Rolle *production order* bzw. die Norm *offer creation* der Rolle *resource* erfüllt werden kann (siehe dazu Abbildung B.2 auf Seite 96 im Anhang). Die Norm *offer acquisition* der Rolle *production order* wird aktiviert, wenn der nächste auszuführende Prozessschritt berechnet ist, mit anderen Worten durch das Empfangen des Events *process step determined* (siehe Abbildung 6.12). Den Empfang eines Aktivierungsereignisses wird für einen Plan nicht modelliert, da dies bereits im Modell eines anderen Plans geschehen ist[4].

Ein Plan wird im Plan Model mit der Notation einer Aktion umrandet und kann Objekte als Eingangsparameter sowie Ereignisse als Ausgangsparameter enthalten. Der Plan *offer acquiring* enthält als Eingangsparameter den Prozessplan, generiert eine Angebotsanfrage (*generate offer request*) und schickt diese an alle bekannten Ressourcen. Bei einer Ressource, welche solch eine Angebotsanfrage empfängt, wird die Norm *offer creation* aktiviert. Der Plan *offer creating* besitzt als Eingangsparameter das Anfrageobjekt, überprüft die Fähigkeiten

[4]Ab diesem Modell werden in der Arbeit aus Platz- und Überblicksgründen nicht mehr alle Modelle der exemplarischen Entwicklung komplett modelliert, sondern nur noch wichtige Teilaspekte

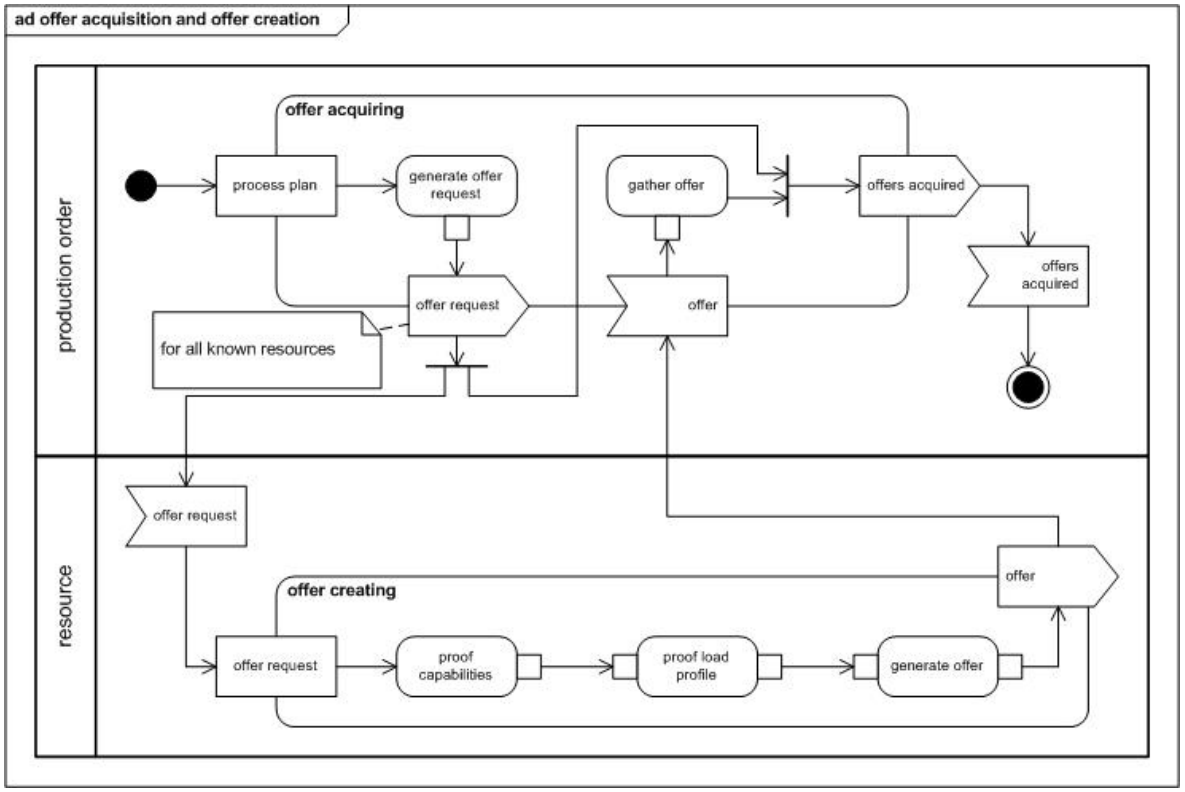

Abbildung 6.13: Pläne *offer acquiring* und *offer creation* im Plan Model

(*proof capabilities*) und das Auslastungsprofil (*proof load profile*) der Ressource, generiert ein Angebot für die Ausführung eines Prozessschrittes (*generate offer*) und sendet dieses an die Rolle *production order* zurück. Durch den Empfang wird der Plan der Rolle *production order* fortgesetzt. Dabei wird das Angebot gesammelt (*gather offer*) und sobald alle Angebote für ausgesandte Angebote eingegangen sind, das Event *offers acquired* ausgelöst. Durch den Empfang dieses Events wird die Norm *offer acquisition* deaktiviert und die Norm *offer analysis* aktiviert (siehe Abbildung B.2).

Die Entwicklung von Plänen beeinflusst auch das Role Model, da jeder Plan einer Rolle zugeordnet wird. Die Zuordnung von Plänen erfolgt wie bei den Normen durch eine Referenz auf den Plan innerhalb einer Rolle. Die Referenzierung erfolgt durch ein zusammengesetztes Viereck als Symbol für einen Plan (siehe Abbildung 6.14).

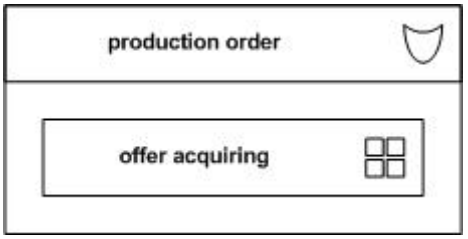

Abbildung 6.14: Rolle *production order* mit Referenz auf den Plan *order acquiring*

6.3.9 Interaction Model

Das Interaction Model modelliert Interaktionspattern zwischen zwei Rollen. Dabei wird noch nicht auf den genauen Nachrichtenaustausch für die Interaktionen nach einem definierten Protokoll eingegangen, sondern nur festgelegt, welche Nachrichten mit welchen Informationen zwischen zwei Rollen ausgetauscht werden. Die Ableitung der Interaktionspattern für ein Interaction Model erfolgt aus dem Plan Model. Dazu werden jeweils zwei Sende- und Empfangsereignisse zwischen zwei Partitionen im Plan Model als eine Interaktion betrachtet. Die Modellierung erfolgt in einem Sequenzdiagramm.

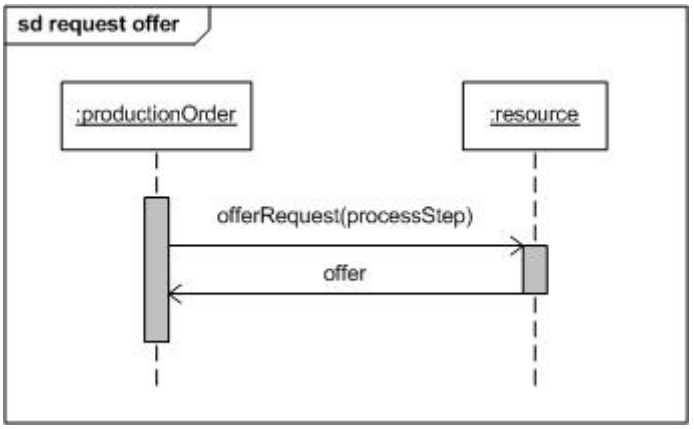

Abbildung 6.15: Interaktionspattern *requestOffer* zwischen den Rollen *production order* und *resource*

Das in Abbildung 6.15 dargestellte Interaktionspattern *requestOffer* hat die Rolle *production order* als Initiator und die Rolle *resource* als Beteiligten. Die ausgetauschten Nachrichten sind die Angebotsanfrage für die Ausführung eines Prozessschritts *offerRequest(processStep)* und das Angebot *offer* als Rückantwort. Dieses Model entspricht dem Nachrichtenaustausch aus dem Plan Model in Abbildung 6.13. Im Anhang sind in der Abbildungen B.4 alle weiteren Interaktionspattern dargestellt, welche sich für die Rollen ergeben.

Das Interaktionspattern wird von der initiierenden Rolle im Role Model durch das Symbol mehrerer konzentrischer Kreise referenziert. Abbildung 6.16 zeigt die Referenzierung der Rolle *production order* auf das Interaktionspattern *requestOffer*.

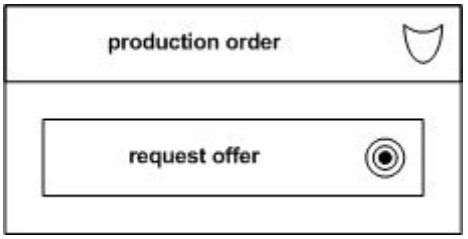

Abbildung 6.16: Rolle *production order* mit Referenz auf Interaktionspattern *request offer*

6.3.10 Service Model

Das Service Model spezifiziert die Dienste, welche eine Rolle sowohl nach außen, als auch nach innen anbieten muss. Diese werden aus dem Plan Model und dem Interaction Model ab-

geleitet. Dabei wird jedoch nicht die letztendliche Implementierung eines Dienstes modelliert, sondern lediglich dessen Signatur, sprich Deklaration, Name, Ein- und Ausgabeparameter.

Aktionen im Plan Model werden zu privaten Diensten – als *private* deklariert –, da sie vorerst nur für den internen Aufruf einer Rolle verwendet werden. Öffentliche Dienste – als *public* deklariert – werden aus dem Interaction Model abgeleitet. Dabei wird ein Dienst immer dem Beteiligten eines Nachrichtenaustausches zugeordnet, der ihn für den Aufruf durch einen Initiator bereitstellen muss.

Intuitiv würde sich für Dienste im Service Model die gewohnte Notation von Methoden in einem Klassendiagramm anbieten, was bedeuten würde, dass die Dienste direkt in den Rumpf der Rolle modelliert werden würden. Dies widerspräche jedoch dem Konzept der Referenzen und der Widerverwendbarkeit von Modellen, in diesem Fall von Diensten. Besitzen zwei Rollen denselben Dienst, müssten die Methoden bei jeder Rolle separat eingetragen werden. Durch den Eintrag einer Referenz könnte die weitere Verfeinerung eines Dienstes im Service Model stattfinden und muss nicht für jede Rolle separat durchgeführt werden. Zudem wäre es in Zukunft denkbar, Dienste mit gewissen Bedingungen (Aufrufbedingungen, Provisionen, ...) auszustatten, was bei Methoden in Klassendiagrammen nicht möglich wäre.

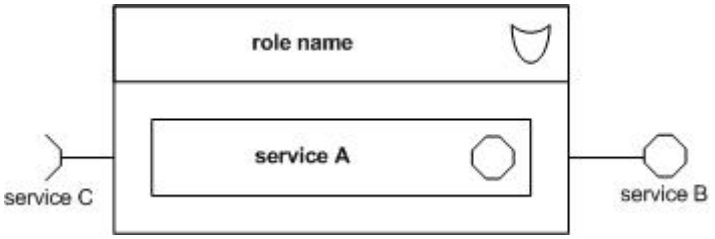

Abbildung 6.17: Notation von Services im Role Model

Aus diesem Grund wird sich bei der Notation eines Dienstes eher an der Agent Modeling Language (AML) [104] angelehnt und ein Sechseck als Symbol für einen Dienst verwendet. Interne Dienste werden innerhalb einer Rolle, extern angebotene Dienste außerhalb einer Rolle sowie extern benötigte Dienste außerhalb einer Rolle als halbes Sechseck referenziert (siehe Abbildung 6.17). Diese Notationsweise ähnelt der Notation von Schnittstellen in UML 2.0. Die Modellierung eines Dienstes im Service Model entspricht in etwa dem Aufbau einer Norm im Norm Model. Ein Service besitzt eine Sichtbarkeit (*visibility*), Eingabe- (*input*) und Ausgabeparameter (*output*).

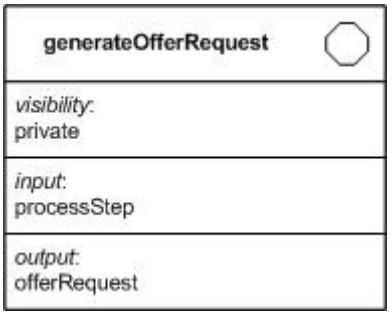

Abbildung 6.18: Service *generateOfferRequest* im Service Model

Abbildung 6.18 zeigt die Modellierung des Dienstes *generateOfferRequest* der Rolle *production order*, so wie er aus dem Plan Model in Abbildung 6.13 abgeleitet wird. Der Dienst soll eine Angebotsanfrage für die Ausführung eines Prozessschritts erzeugen. Er ist daher als *private* deklariert und besitzt den Prozessschritt als Eingabeparameter und die Angebotsanfrage als Ausgabeparameter.

6.3.11 Autonomic Manager Role Model

Das Autonomic Manager Role Model modelliert im Gegensatz zum Role Model diejenigen Rollen, welche später von Autonomic Managers übernommen werden und zur Erzeugung der Selbst-x Eigenschaften verwendet werden. Autonomic Manager Rollen können nicht wie Managed Resource Rollen aus anderen Modellen auf CIM-Ebene abgeleitet werden, da die Selbst-x Eigenschaften nicht softwareunabhängig sind und daher nicht auf CIM-Ebene definiert werden dürfen. Aus diesem Grund müssen die Normen für diese Rollen quasi parallel entwickelt werden, so dass sich Rollen identifizieren lassen. Das Notationssymbol einer Autonomic Manager Rolle entspricht von der Form her einer Managed Resource Rolle (siehe 6.3.6), ist jedoch ausgefüllt (siehe Abbildung 6.19).

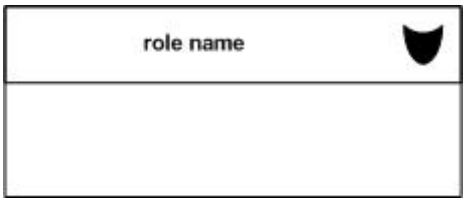

Abbildung 6.19: Notation einer Autonomic Manager Rolle im Autonomic Manager Role Model

6.3.12 Autonomic Manager Norm Model

Die Normen für Autonomic Manager Rollen besitzen dieselbe Funktionsweise und Notation wie Normen für Managed Resource Rollen (siehe 6.3.7). Hier erzeugen die Normen jedoch die Selbst-x Eigenschaften des Systems. Daher werden Ziele, Aktivierungs- und Deaktivierungsereignisse so formuliert, dass sie zur Erfüllung der Selbst-x Eigenschaften geeignet sind.

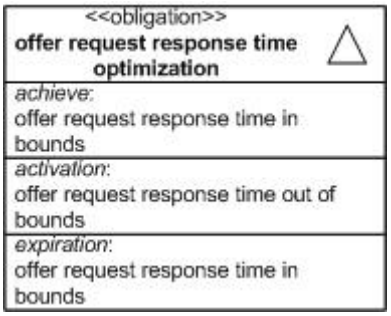

Abbildung 6.20: Norm *offer request response time optimization* im Autonomic Manager Norm Model

Abbildung 6.20 zeigt ein Beispiel für eine Norm der Autonomic Manager Rolle *resource*, welche dadurch identifiziert und ins Autonomic Manager Role Model eingetragen wird. Die Norm *offer request response time optimization* hat zum Ziel, dass die Antwortzeit auf eine

Angebotsanfrage innerhalb einer bestimmten Grenze bleibt (*offer request response time in bounds*) und wird aktiviert, sobald die Antwortzeit die Grenze überschreitet (*offer request response time out of bounds*). Wird die Grenze wieder unterschritten, so wird die Norm deaktiviert. Diese Norm unterstützt die Selbst-Optimierung des Systems und muss später von einem Autonomic Manager übernommen werden, da die Aktivierung der Norm von Messungen abhängt, welche eine Managed Resource nicht ausführen kann.

Auf diese Weise können beliebig viele weiteren Normen für Selbst-x Eigenschaften des gesamten OCS spezifiziert werden. Abbildung B.5 auf Seite 98 im Anhang zeigt die Modellierung der Norm (*order acquisition assuring*) der Autonomic Manager Rolle *production order* für die Selbst-Heilung. Im Fall, dass die Beschaffung von Aufträgen für die Ausführung eines Prozessschrittes aus beliebigen Gründen (bspw. durch den Erhalt eines Angebots für einen anderen Prozessschritt) fehlschlägt, wird eine *Internal Exception* als Event geworfen. Ergibt die Analyse, dass z. B. keine weiteren Angebote vorliegen, wird die Norm aktiviert (*offer acquisition failed*), um sicherzustellen, dass letztlich mindestens ein Angebot vorliegt und das System weiterlaufen kann. Das Werfen dieser Exception muss nachträglich in den entsprechenden Plan im Plan Model eingefügt werden (siehe Abbildung B.6). In dieser Form könnten auch noch Normen für weitere Selbst-x Eigenschaften formuliert werden, auf welche hier jedoch verzichtet werden soll.

Die beiden Autonomic Manager Rollen wurden immer nach der jeweiligen, Managed Resource Rolle benannt, auf welche die Norm angewandt wurde. Dies deutet auf eine Zuordnung der Autonomic Manager Rolle zu der gleichnamigen Managed Resource Rolle hin. Sind für die Erreichung des Ziels einer Norm mehrere Rollen bzw. andere Autonomic Manager Rollen betroffen, so müssen andere Rollennamen für Autonomic Manager Rollen definiert werden. Dies deutet auf orchestrierende Autonomic Manager hin. Die Referenzierung entspricht der bei Managed Resource Rollen.

6.3.13 Autonomic Manager Analyze Model

Das Autonomic Manager Analyze Model modelliert das spätere Monitoring und die Analyse innerhalb eines Autonomic Managers. Da die Analysefunktion entscheiden muss, welche Normen eines Autonomic Managers aktiviert werden, wird in diesem Modell ein zentraler Teil des Wissens eines Autonomic Managers modelliert. Die Modellierung geschieht in diesem Modell durch Aktivitätsdiagramme.

Das Autonomic Manager Analyze Model in Abbildung 6.21 zeigt die Modellierung der Metrik für die Bestimmung der Aktivierung der Norm aus Abbildung 6.20. Das Monitoring wird modelliert durch die beiden Empfangsereignisse *offer request* und *offer sent*. Dies legt fest, welche Ereignisse später vom Monitor eines Autonomic Manager beobachtet werden müssen um die Selbst-Optimierung zu ermöglichen. Ein empfangenes Event wird in diesem Fall vorerst nur gesammelt (*gather event*). Die Einordnung in Symptome ist hier nicht notwendig. Die Metrik zur Messung der Antwortzeit auf Angebotsanfragen soll eine Warnung (*offer request response time out of bounds*) senden, sobald die Antwortzeit 3 ms übersteigt. Dazu wird alle 0,5 ms eine Analyze gestartet, welche die zeitliche Entfernung der gesammelten, zusammengehörenden Ereignisse durchführt (*analyze events*). Es ist angedeutet, dass diese Aktivität noch komplexer ist, hier aber nicht weiter modelliert wird. Da selbst UML 2.0 noch keine Modellierung von zeitlich wiederkehrenden Ereignissen zulässt, wird die Sanduhr als Auslöseereignis mit der Bedingung $\forall$ *0,5 ms* versehen, um diese Funktionalität zu modellie-

ren. Im Falle einer ausgelösten Warnung wird die Norm für die Optimierung der Antwortzeit aktiviert. Andernfalls wird bzw. bleibt sie durch das Aussenden des Ereignisses (*offer request response time in bounds*) deaktiviert.

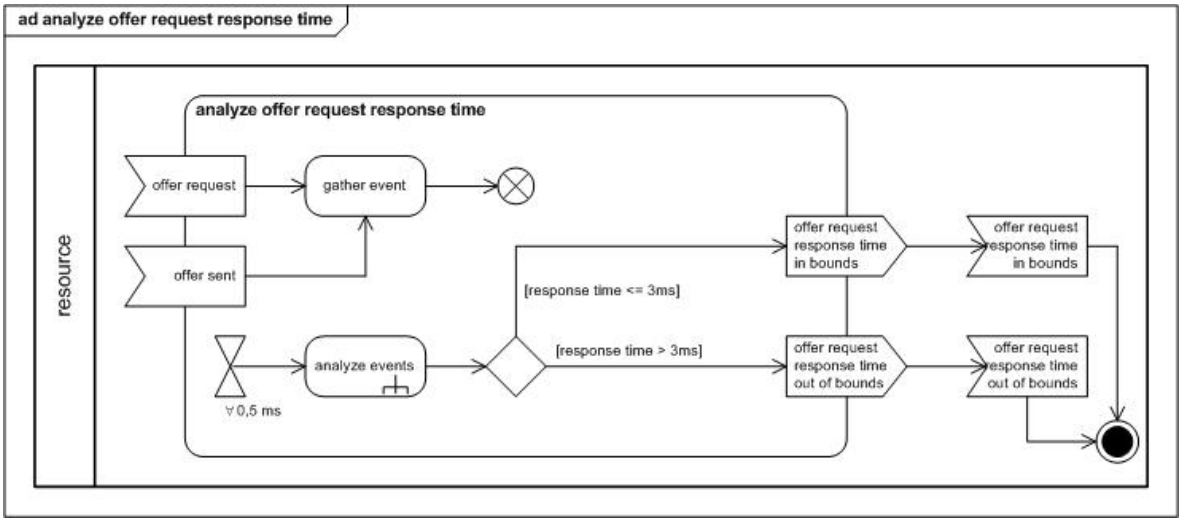

Abbildung 6.21: Autonomic Manager Analyze Model

Das Monitoring wird nicht von einer Autonomic Manager Rolle referenziert, da dies direkt in einen Autonomic Manager transformiert und über den Partitionsnamen des Aktivitätsdiagramms zugeordnet werden kann. Die Analyse hingegen wird referenziert, da diese für das gleiche Monitoring unterschiedlich sein kann. Als Notationssymbol für die Referenz einer Rolle auf eine Analyseregel wird ein sternförmiges Symbol verwendet. Abbildung 6.22 zeigt die Referenzierung der Rolle *resource* auf die Analyseregel *analyze offer request response time*.

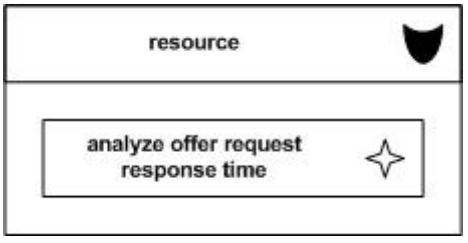

Abbildung 6.22: Autonomic Manager Rolle *resource* mit Referenz auf die Analyseregel *analyze offer request response time*

6.3.14 Autonomic Manager Plan Model

Das Autonomic Manager Plan Model ist ähnlich dem Plan Model und modelliert die Pläne für die Planungsfunktion von Autonomic Manager Rollen, um Ziele in Normen erfüllen zu können. Dabei müssen nicht nur Interaktionen zwischen zwei Autonomic Manager Rollen modelliert werden, sondern auch die Interaktionen zwischen Autonomic Manager Rollen und Managed Resource Rollen, um Aktionen und Interaktionen für die folgenden Modelle ableiten zu können. Die Modellierungsweise ist die gleiche wie im Plan Model, für Autonomic Manager Rollen wird noch zusätzlich zur Unterscheidung von Managed Resource Rollen ein *am* an den Partitionsnamen im Aktivitätsdiagramm angehängt.

Abbildung 6.23 zeigt ein Autonomic Manager Plan Model mit einem Plan zur Erfüllung der Norm *offer request response time optimization*. Unter der Annahme, dass die Planungsfunktion keinen Plan gefunden hat, die Antwortzeit durch Aktionen der eigenen Managed Resource zu verbessern bzw. dieser nicht zum gewünschten Ergebnis geführt hat, muss die Lösung für das Problem über andere Managed Resources geregelt werden. In diesem Fall generiert die Autonomic Manager Rolle *resource* einen Änderungswunsch für die Erstellung der Angebotsanfragen anderer *production orders* (*generate order request adjustment desire*) und schickt diesen an die Autonomic Manager Rolle der *prodcution orders*. Diese generieren einen Änderungsauftrag für die Erstellung der Anfragezeit von Angebotsanforderungen *production order*-Managed Resource Rollen (*generate offer request adjustment instruction*) und senden diesen an die Rollen. Eine *production order*-Rolle ändert bei Erhalt dieser Anforderung die Erstellungszeit für Angebotsanforderungen (*adjust offer request generation time*) entsprechend des Änderungsauftrages, in diesem Fall erhöht sie diese.

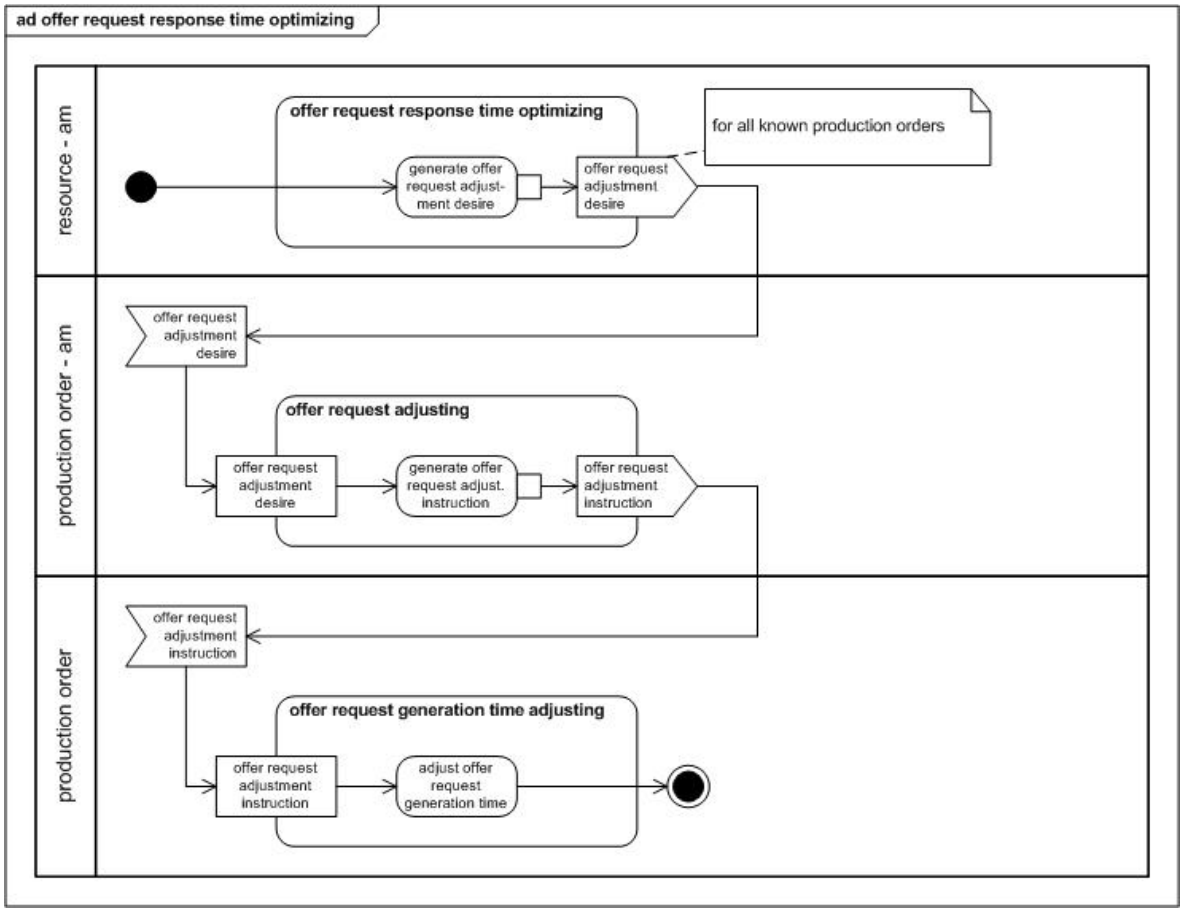

Abbildung 6.23: Autonomic Manager Plan Model

Dieser Plan fügt aber auch der Autonomic Manager Rolle *production order* einen Plan hinzu, sowie der Managed Resource Rolle *production order*. Daher wird implizit das Plan Model, Service Model und Interaction Model nachträglich geändert. Die Referenzierung entspricht der bei Managed Resource Rollen.

6.3.15 Autonomic Manager Interaction Model

Das Autonomic Interaction Model ist wie das Interaction Model aufgebaut und modelliert die Interaktionspattern und den notwendigen Nachrichtenaustausch für Autonomic Manager Rollen untereinander aber auch zwischen Autonomic Manager Rollen und Managed Resource Rollen.

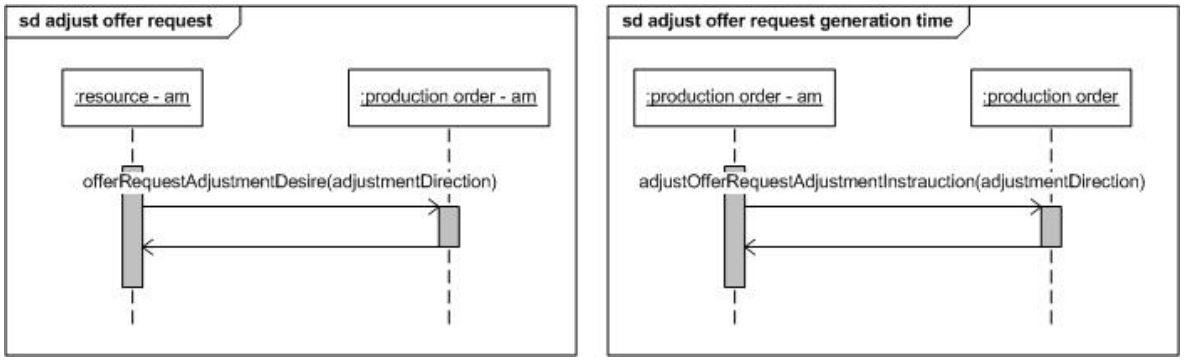

Abbildung 6.24: Autonomic Manager Interaction Model

Das Autonomic Manager Interaction Model in Abbildung 6.24 zeigt die Interaktionspattern, welche durch das Plan Model in Abbildung 6.23 entstanden sind. Zum einen das Interaktionspattern zwischen den Autonomic Manager Rollen *resource - am* und *production order - am* zum anderen zwischen *production order - am* und der Managed Resource Rolle *production order*. Die Referenzierung entspricht der bei Managed Resource Rollen.

6.3.16 Autonomic Manager Service Model

Das Autonomic Manager Service Model leitet sich aus dem Autonomic Manager Plan Model ab und verwendet dieselbe Notation wie das Service Model. Hier werden die Dienste definiert, welche eine Autonomic Manager Rolle ausführen muss, um die gewünschte Funktionalität für die Selbst-x Eigenschaften bereitzustellen.

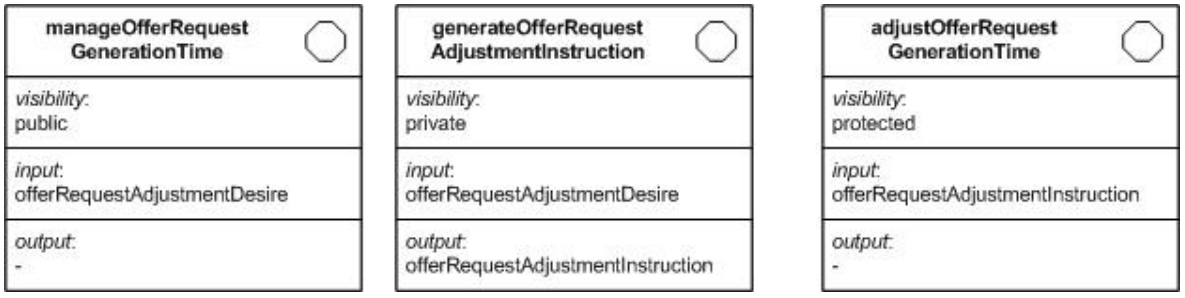

Abbildung 6.25: Services im Autonomic Manager Service Model und im Service Model

Abbildung 6.25 zeigt zwei Dienste des Autonomic Manager Service Models (*manageOfferRequestGenerationTime* und *generateOfferRequestAdjustmentInstruction*), sowie einen Dienst aus dem Service Model (*adjustOfferRequestGenerationTime*). Die ersten beiden Dienste gehören zu der Autonomic Manager Rolle *production order*, der dritte Dienst zu der Managed Resource Rolle *production order*. Die Dienste wurden aus dem vorangehenden Autonomic Manager Plan Model in Abbildung 6.23 sowie dem vorangehenden Autonomic Manager Interaction Model in Abbildung 6.24 abgeleitet. Zu beachten ist die unterschiedliche Sichtbarkeit

der Dienste. Die Deklaration *protected* wird für Dienste verwendet, welche Funktionen für Autonomic Manager Rollen anbieten, jedoch nicht von anderen Managed Resource Rollen aufgerufen werden können. Die Referenzierung der Dienste entspricht der Referenzierung im Service Model.

6.3.17 Interaction Protocol Model

Im Interaction Protocol Model werden konkrete Interaktionsprotokolle für alle Rollenarten definiert. Erst an diesem Punkt muss entschieden werden, ob die Kommunikation zwischen den Rollen direkt oder indirekt abläuft. Die Interaktionsprotokolle müssen dabei die Interaktionspattern aus dem Interaction Model und dem Autonomic Manager Interaction Model erfüllen. Dabei verhalten sich die Interaktionspattern zu den Interaktionsprotokollen wie Interfaces zu Implementierungen. Die Modellierung der Interaktionsprotokolle findet in Sequenzdiagrammen statt. Diese müssen ebenfalls von einer Rolle aus im Role Model referenziert werden. Für die Referenz im Role Model wird ein brücken-ähnliches Symbol verwendet (siehe Abbildung 6.26).

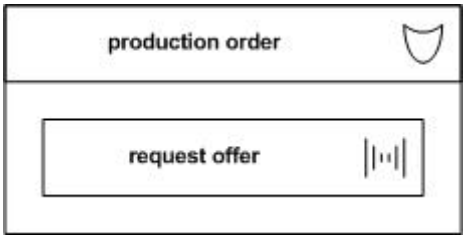

Abbildung 6.26: Referenz auf ein Interaktionsprotokoll im Role Model

Das zu entwickelnde PPCS soll wie in der Fallstudie I über indirekte Kommunikation ablaufen. Abbildung 6.27 zeigt ein Interaktionsprotokoll für das Interaktionspattern *request offer* aus Abbildung 6.15. Entsprechend der Ablaufbeschreibung in 4.1.2.2 erzeugt die Rolle *production order* in einer bestimmten Frequenz *exploring ants*, und geben ihnen den Auftrag mit auf den Weg, Angebote für eine attraktive Route einzuholen (*requestOffers(processStep)*). Die Ameisen bewegen sich entlang der Produktionsstraße und besorgen sich an jeder Kreuzung die Informationen über die *subnet capabilities*, welche dort zuvor von den *feasibility ants* (siehe 4.1.2.1) als Pheromone abgelegt wurden. Genau genommen interagieren die Ameisen dabei mit der Umgebung der Kreuzungen, jedoch wird diese Umgebung von der Kreuzung bereitgestellt, so dass die Interaktion im Sequenzdiagramm direkt mit den Kreuzungen geschieht. Dasselbe gilt später auch für die Interaktion zwischen Ameisen und *resources*. Treffen sie anstatt auf eine Kreuzung auf eine Ressource, so beschaffen sie sich von dieser ein Angebot und speichern es bei sich. Sobald die Ameisen am Ende der Produktionsstraße sind, kehren sie zum *production order* zurück und legen die gesammelten Angebote ihrer Route dort ab.

An diesem Beispiel ist zu erkennen, wie das Interaktionspattern *request offer* mittels indirekter Kommunikation umgesetzt wurde. Die Ameisen dienen im Interaktionsprotokoll als Interaktionsmedium, indem sie die Nachrichten von der initiierenden Rolle zur beteiligten Rolle transportieren und umgekehrt, ohne dass diese beiden Rollen direkt miteinander kommunizieren müssen. Die einzige kleine Abweichung ist, dass die initiierende Rolle im Gegensatz zum Interaktionspattern nicht nur ein Angebot zurückerhält, sondern eine Menge von Angeboten. Dies ist aber vernachlässigbar, da der Erhalt von nur einem Angebot ein Spezialfall dieses Interaktionsprotokolls ist.

Im Anhang befinden sich auf der Seite 99 die Interaktionsprotokolle für die Interaktionspattern *propagate capabilities* (Abbildung B.7) und *reserve resource* (Abbildung B.8), so dass alle drei Kontrollstufen des PPCS (siehe 4.1.2) über eine indirekte Kommunikation ablaufen.

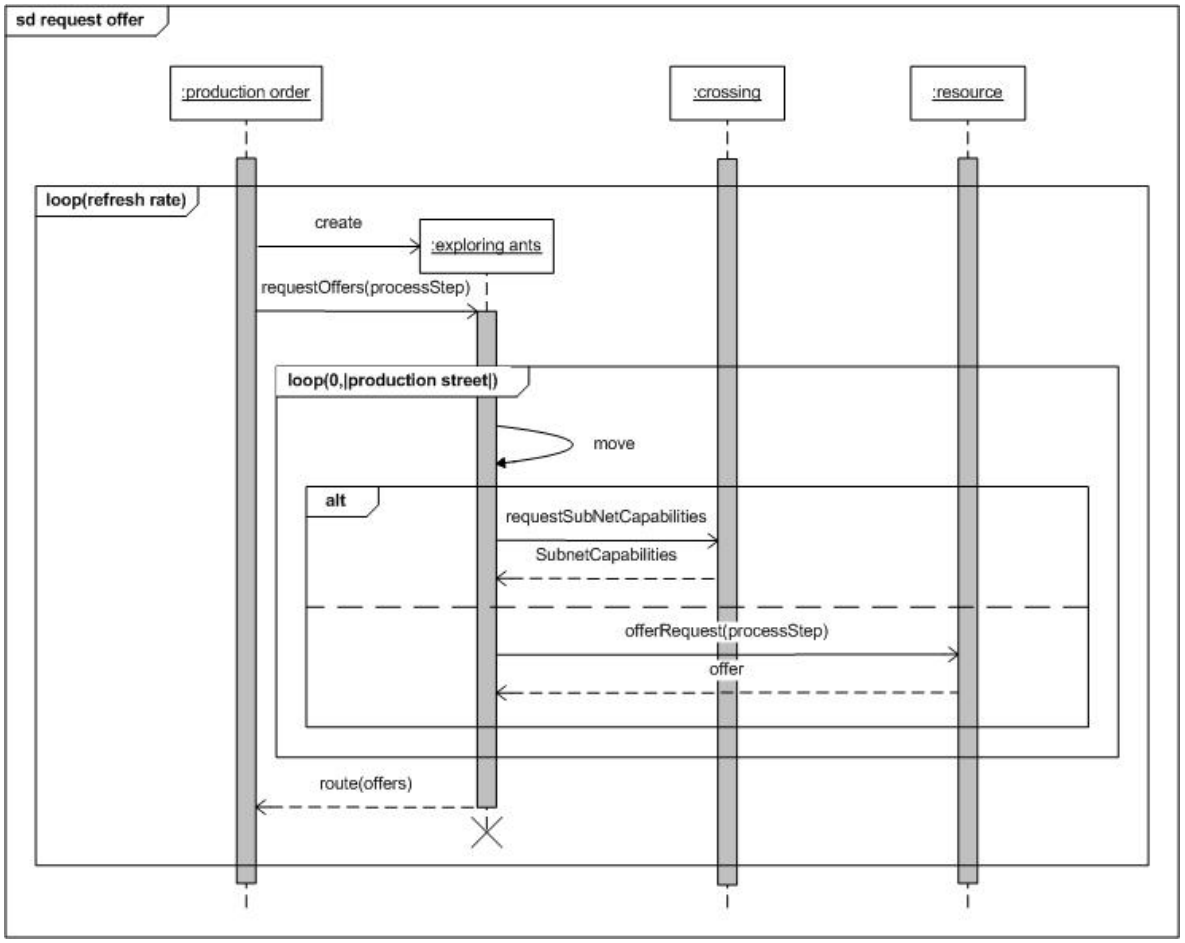

Abbildung 6.27: Interaktionsprotokoll für das Interaktionspattern *request offer*

6.3.18 Autonomic Element Model

Das Autonomic Element Model modelliert die zukünftigen Autonomic Elements, welche später zum Einsatz kommen sollen. Dabei wird festgelegt, welche Rollen von welchen Elementen übernommen werden und welche Autonomic Manager Rollen welche Managed Resource Rollen verwalten werden. Diese Aufteilung kann prinzipiell automatisiert werden und Regeln beachten, welche die Emergenz von Selbst-x Eigenschaften leichter ermöglichen. Da dies noch von vielen weiteren Forschungen abhängig ist, wird hier eine intuitive Regel nach Namen zur Aufteilung angewandt. In den Agentenmethodologien erfolgt die Zuteilung von Rollen auf Agenten durch Notation von Kanten zwischen Agenten und Rollen. Da dies kein Standard in UML 2.0 darstellt, erfolgt die Modellierung hier in einem Kompositionsstrukturdiagramm.

Abbildung 6.28 zeigt einen Ausschnitt aus dem Autonomic Element Modell des zu entwickelnden PPCS mit den zwei Autonomic Elements, nämlich ein Autonomic Manager *production order*, welcher die gleichnamige Autonomic Manager Rolle übernommen hat, und eine Managed Resource *production order*, welche ebenfalls die gleichnamige Managed Resource Rolle übernommen hat. Die Namensgebung ist beliebig, da ein Autonomic Element beliebig

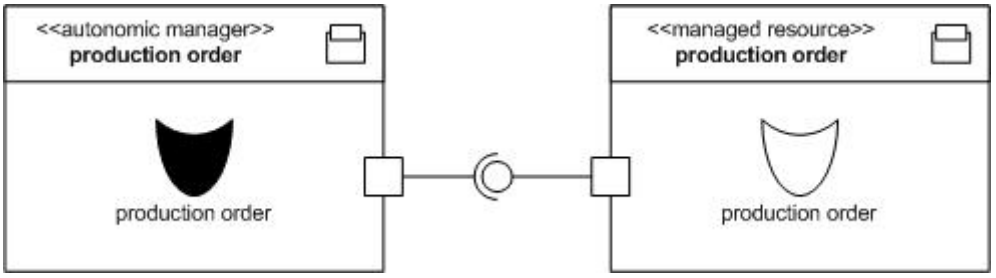

Abbildung 6.28: Autonomic Element Model

viele Rollen übernehmen kann. Das Notationssymbol für ein Autonomic Element ist dem des Autonomic Computing nachempfunden. Welche Autonomic Manager dabei welche Managed Resources verwalten, ist durch die Notation von Schnittstellen modelliert. Die Schnittstellen ergeben sich generell aus dem Autonomic Manager Plan Model bzw. den Service Models. Eine Schnittstelle fasst dabei alle nach außen angebotene Dienste eines Autonomic Elements für ein anderes Autonomic Element zusammen.

In diesem Modell werden alle Autonomic Elements mit ihren Abhängigkeiten untereinander modelliert, welche später zum Einsatz kommen werden. In manchen Agentenmethodologien wird an diesem Punkt ein Acquaintance Model verwendet, um den Kommunikationsfluss zwischen den Systemkomponenten zu bestimmen und, beispielsweise bei einem Flaschenhals bei einer Komponente, notwendige Änderungen zu vollziehen. Die Erstellung eines solchen Modells ist in diesem Ansatz nicht notwendig, da die Informationen über die Kommunikation bereits aus dem Autonomic Element Model abgelesen werden können. Auf die Meßverfahren zur Bestimmung von Änderungsanforderungen und eventuell notwendigen Änderungen wird in dieser Arbeit nicht weiter eingegangen.

6.3.19 Autonomic Element Instance Model

Das Autonomic Element Instance Model wird für die Modellierung der Verteilung der Autonomic Elements auf die Systemumgebung verwendet. Dabei werden die im Autonomic Element Model identifizierten Autonomic Elements auf verschiedene Knoten im System zugeordnet. Die verschiedenen Knotenarten des Systems sind bereits durch das Business Process Model bzw. das Environment Model bestimmt. Die realen Knoten des Systems werden erst hier festgelegt. Die Zuteilung kann wiederum nach Regeln erfolgen, welche in dieser Arbeit noch nicht von belang sind. Die Modellierung erfolgt in einem abgewandelten Verteilungsdiagramm.

Abbildung 6.29 zeigt das Autonomic Element Instance Model für das zu entwickelnde PPCS. Für das System wird eine Topologie vorausgesetzt, welche der aus Abbildung 4.1 entspricht. Die Kreuzungen und die Produktdatenbank besitzen dabei jeweils nur eine Managed Resource als Autonomic Element, wohingegen der Produktionsauftrag und die Ressourcen zusätzlich noch einen Autonomic Manager besitzen. Die Artefakte in einem Verteilungsdiagramm wurden durch die Autonomic Elements ersetzt, der Rest des Verteilungsdiagramms entspricht dem Standard.

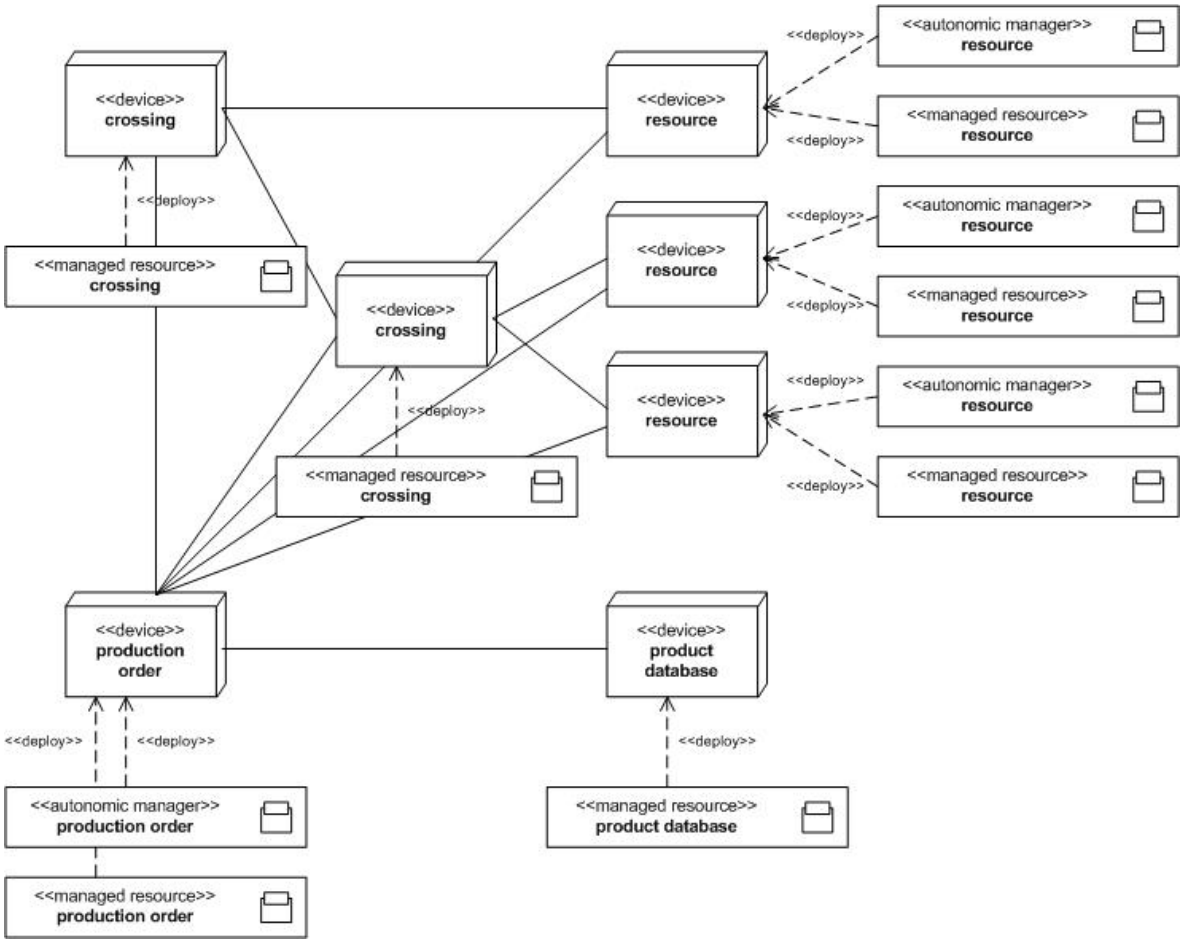

Abbildung 6.29: Autonomic Element Instance Model

6.4 Modelltransformationen

Der letzte Abschnitt beschrieb die Transformation von Modellelementen informell. Erst mit der Definition von formalen Modelltransformationen zwischen einem Quell- und einem Zielmodell wird jedoch der wirkliche Nutzen der MDA im Entwicklungsansatz deutlich. Indem eine Transformation zwischen zwei Modellen bzw. Modellebenen durch Regeln automatisiert wird, kann meist die gesamte Aktivität zur Erstellung des Zielmodells automatisiert und dem Entwickler abgenommen werden. Von der MDA werden grundsätzlich drei verschiedene Abbildungsarten für Modelltransformationen vorgeschlagen (vgl. [21]):

Typbasierte Abbildungen Dabei werden Transformationsregeln auf Basis der verwendeten Typen in einem PIM und einem PSM definiert, welche durch MOF-Metamodelle spezifiziert sind.

Instanzbasierte Abbildungen Dabei werden Modellelemente (sprich Instanzen) in einem PIM, welche in ein PSM transformiert werden sollen, identifiziert und plattformspezifisch markiert.

Musterbasierte Abbildungen Dabei werden die erwähnten Markierungen in Templates zusammengefasst, um auf unterschiedliche Modelle angewandt werden zu können.

Die in dieser Arbeit definierten Transformationsregeln erleichtern zwar die Entwicklungsarbeit, automatisieren jedoch nur weinige Aktivitäten vollständig. Die Regeln basieren auf den in 6.3 beschriebenen Informationsflüssen und Abhängigkeiten zwischen den Modellen und sind größtenteils musterbasiert. Sie umfassen Transformationen zwischen Modellen der CIM-Ebene, der PIM-Ebene und zwischen den beiden Ebenen. Die Notation der einzelnen Regeln erfolgt im *if-then*-Stil, d. h. wenn die Bedingung bei *if* wahr ist, wird die bei *then* angegebene Aktion ausgeführt. Die Regelbezeichnung erfolgt nach einer festen Syntax: *TR_XXX_YYY_ZZ*. Dabei steht

- *TR* für *transformation rule*, d. h. die Kennzeichnung einer Transformationsregel,
- *XXX* für eine dreistellige Kennzeichnung des Quellmodells der Transformation,
- *YYY* für eine dreistellige Kennzeichnung des Zielmodells der Transformation,
- *ZZ* für eine zweistellige Nummerierung der Transformationsregel zwischen dem Quell- und Zielmodell.

Die Zuordnung einer dreistelligen Kennzeichung zu ihrem Modell ist in Abbildung C.1 dargestellt. Die Beschreibung der Regeln erfolgt in diesem Abschnitt, die Regeln selber befinden sich im Anhang C und D ab Seite 101.

6.4.1 Transformationen im CIM

Für das CIM ergeben sich aufgrund der Abstraktionssicht nur wenig Transformationsregeln, da sich die Charakterisierung der Umgebung eines Systems und deren Nutzen nur begrenzt automatisieren lässt und in erster Linie vom Anwender bzw. von der Domäne abhängt.

6.4.1.1 Regel: TR_BCM_BPM_01

Diese Regel (siehe Abbildung C.1) bildet die Partition, welche einen Prozess im Business Context Model ausführt, der durch ein OCS unterstützt werden soll, in das Business Process Model ab. Der Name der Partition wird dabei beibehalten. Wie auch für die folgenden Transformationen wird hier vorausgesetzt, dass das Zielmodell bereits vorhanden ist, d. h. es existieren bereits alle Modellrahmen des Entwicklungsansatzes, welche anschließend automatisch durch Transformationen oder manuell durch den Entwickler gefüllt werden. In diesem Fall müssen nach Anwendung der Transformationsregel im Business Process Modell die Unterpartitionen und deren Geschäftsprozesse, welche das OCS unterstützen soll, nachgetragen werden.

6.4.1.2 Regel: TR_BPM_EVM_01

Diese Regel (siehe Abbildung C.2) trägt jede Unterpartition des Business Process Model als Klasse in das Environment Model ein. Der Name der Partition und der Klasse sind identisch. Weitere Klassen, Attribute und Assoziationen müssen im Environment Modell manuell nachgetragen werden.

6.4.1.3 Regel: TR_EVM_UCM_01 und TR_EVM_UCM_02

Diese Regeln (siehe Abbildung C.3 und C.4) fügen diejenigen Klassen, welche als Partition im Business Process Model existieren, in das Use Case Diagramm und in ein Sequenzdiagramm des Use Case Model ein. Die Klassen, welche dabei einen Startpunkt im Business Process Model besitzen, werden als primäre Akteure und somit als Initiatoren eines Anwendungsfalls in das Use Case Diagramm eingetragen, die anderen Klassen als sekundäre Akteure. Die Anwendungsfälle selber müssen manuell nachgetragen werden. Im Sequenzdiagramm werden die Klassen als Objekte inkl. Lebenslinie eingetragen. Klassen, welche lediglich für das Verständnis der Umgebung im Environment Model modelliert wurden, bleiben vorerst unberücksichtigt.

6.4.1.4 Regel: TR_UCM_UCM_01

Diese Regel (siehe Abbildung C.5) bildet das System aus dem Use Case Diagramm in ein Sequenzdiagramm des Use Case Models als Objekt inkl. Lebenslinie ab.

6.4.1.5 Regel: TR_UCM_UCM_02

Diese Regel (siehe Abbildung C.6) transformiert einen Anwendungsfall als initiierende Nachricht in ein Sequenzdiagramm. Die Nachricht läuft dabei von einem primären Akteur zu dem System hin.

6.4.1.6 Regel: TR_BPM_UCM_01

Diese Regel (siehe Abbildung C.7) legt den Rahmen der Nachrichten in einem Sequenzdiagramm des Use Case Model fest. Dazu wird eine Aktion im Business Process Model entweder in eine Nachricht von einem Objekt an sich selber (interner Aufruf) oder an ein anderes Objekt (externer Aufruf) transformiert. Diese Entscheidung muss der Entwickler treffen. Die Anwendung der Regel führt nur zu einer Referenz im Sequenzdiagramm, welche dann durch den Entwickler gefüllt werden muss. In Abbildung 6.7 ist bereits das endgültige Sequenzdiagramm der Fallstudie gezeigt.

6.4.1.7 Regel: TR_XXX_ONM_01

Diese Regel (siehe Abbildung C.8) steht für eine Wildcard-Regel und kann auf alle Modelle angewendet werden. Dabei wird jeder Begriff, welcher in einem beliebigen Modell definiert wurde, in das Ontology Model als Klasse eingetragen. Dort kann der Begriff manuell weiter verfeinert, d. h. die Klasse mit Attributen und Assoziationen ergänzt werden. Alle Klassen ergeben bilden zusammen den Grundwortschatz, so dass Anwender und Entwickler das gleiche Verständnis von Begriffen haben.

6.4.2 Transformationen vom CIM in das PIM

Für die Transformation zwischen CIM und PIM bestehen zum jetzigen Zeitpunkt nicht viele formale Regeln. Je mehr Ergebnisse die Erforschung der Selbstorganisation und ihrer Zusammenhänge liefert, desto mehr Regeln werden hier entstehen.

6.4.2.1 Regel: TR_UCM_MRM_01

Diese Regel (siehe Abbildung C.9) leitet die Rollen für das Role Model aus dem Use Case Model ab. Dazu wird jedes Objekt im Sequenzdiagramm in eine Rolle transformiert, mit Ausnahme des zu entwickelnden Systems an sich. Die Namen werden beibehalten.

6.4.2.2 Regel: TR_UCM_MRM_02

Diese Regel (siehe Abbildung C.10) transformiert eine initiierende Nachricht aus dem Sequenzdiagramm des Use Case Model in eine Rolle des Role Model. Die Regel ergänzt die Regel TR_UCM_MRM_01, da das System selber nur einen Rahmen vorgibt und dessen Repräsentation im Sequenzdiagramm nicht in eine Rolle transformiert werden darf.

6.4.2.3 Regel: TR_UCM_MNM_01

Diese Regel (siehe Abbildung C.11) identifiziert einen Teil der Normen für Rollen. Dazu wird jeder interne Nachrichtenaustausch im Sequenzdiagramm des Use Case Model zu einer Obligation für diejenige Rolle, dessen korrespondierendes Objekt im Sequenzdiagramm den Aufruf initiiert. Für Permissions und Prohibitions stehen noch keine Transformationsregeln zur Verfügung. Die Bezeichnung einer Norm entspricht der Bezeichnung der Nachricht. Zusätzlich wird eine Referenz auf den Kopf der Obligation in das Role Model bei der entsprechenden Rolle eingetragen.

6.4.2.4 Regel: TR_UCM_MNM_02

Diese Regel (siehe Abbildung C.12) identifiziert ebenfalls Obligationen als Normen für Rollen, allerdings wird durch diese Regel nur der Rahmen einer Norm im Norm Model erstellt. Dazu wird jeder externe Nachrichtenaustausch im Sequenzdiagramm des Use Case Model in solch einen Rahmen transformiert. Die Norm kann nicht automatisch benannt werden, da sie mehr als nur eine Nachricht umfasst. Dies muss vom Entwickler nachgetragen werden. Die Referenz auf den Kopf der Norm wird dennoch in das Role Model bei der entsprechenden Rolle eingetragen.

6.4.3 Transformationen im PIM

Aufgrund der Vielzahl der Modelle im PIM können hier viele Transformationsregeln definiert werden.

6.4.3.1 Regel: TR_MNM_MPM_01

Diese Regel (siehe Abbildung C.13) transformiert Normen in Bestandteile des Plan Model. Dazu wird die Rolle, welche eine Norm erfüllen muss, als Partition in das Plan Model eingetragen, das Aktivierungsereignis ignoriert, das Deaktivierungsereignis als Empfangsereignis vor den Endpunkt des Diagramms und als Sendeereignis am Ende eines Plans notiert. Zusätzlich wird eine Referenz von der entsprechenden Rolle im Role Model auf den Plan eingetragen.

6.4.3.2 Regel: TR_UCM_MPM_01

Diese Regel (siehe Abbildung C.14) verbindet Pläne im Plan Model miteinander. Dazu wird ein Nachrichtenaustausch im Sequenzdiagramm des Use Case Model als Sende- und Empfangsereignis an den entsprechenden Plänen derjenigen Partitionen notiert, deren korrespondierenden Objekte im Sequenzdiagramm an dem Nachrichtenaustausch beteiligt sind. Sowohl für diese Regel wie auch für die TR_MNM_MPM_01 könnten mehrere Transformationen definiert werden, welche noch genauer die Notation von Elementen im Plan Model beschreiben.

6.4.3.3 Regel: TR_MPM_MIM_01

Diese Regel (siehe Abbildung C.15) automatisiert Aktivität 9 des Entwicklungsprozesses vollständig. Sie transformiert einen Nachrichtenaustausch zwischen zwei Partitionen im Plan Model in ein Interaktionspattern im Interaction Model. Die initiierende Partition wird zur initiierenden Rolle, die empfangende Partition zur partizipierenden Rolle. Die Referenz auf das Interaktionspattern wird bei der initiierenden Rolle eingetragen.

6.4.3.4 Regel: TR_MPM_MSM_01

Diese Regel (siehe Abbildung C.16) automatisiert einen Teil der Aktivität 10 des Entwicklungsprozesses. Dazu wird eine interne Aktion im Plan einer Rolle in einen privaten Service transformiert und eine Referenz bei der Rolle im Role Model gesetzt.

6.4.3.5 Regel: TR_MIM_MSM_01

Diese Regel (siehe Abbildung C.17) automatisiert den restlichen Teil der Aktivität 10 des Entwicklungsprozesses, so dass auch dieser vollautomatisiert ablaufen kann. Dazu wird eine Nachricht in einem Interaktionspattern in einen öffentlichen Service der partizipierenden Rolle transformiert.

6.4.3.6 Regel: TR_ANM_ARM_01

Da Autonomic Manager Rollen noch nicht vom CIM abgeleitet werden können, müssen sie definiert werden, sobald eine geeignete Norm für sie spezifiziert wird. Diese Regel (siehe Abbildung D.1) erstellt lediglich eine Autonomic Manager Rolle und fügt ihr eine Referenz auf eine Norm hinzu.

6.4.3.7 Regel: TR_ANM_AAM_01

Diese Regel (siehe Abbildung D.2) transformiert das Aktivierungsereignis einer Norm einer Autonomic Manager Rolle in ein Empfangsereignis im Autonomic Manager Analyze Model und ein Sendeereignis in der Analyseregel. Die Analyseregel selber muss manuell erstellt werden, um zu entscheiden, wann eine Norm durch das Sendeereignis aktiviert wird.

6.4.3.8 Regel: TR_ANM_AAM_02

Diese Regel (siehe Abbildung D.3) transformiert das Deaktivierungsereignis einer Norm einer Autonomic Manager Rolle in ein Empfangsereignis im Autonomic Manager Analyze Model und ein Sendeereignis in der Analyseregel. Die Analyseregel selber muss manuell erstellt werden, um zu entscheiden, wann eine Norm durch das Sendeereignis deaktiviert wird.

6.4.3.9 Regel: TR_ANM_APM_01

Diese Regel (siehe Abbildung D.4) transformiert Normen von Autonomic Manager Rollen in Bestandteile des Autonomic Manager Plan Model. Dazu wird die Rolle einer Norm als Partition in das Plan Model eingetragen, das Aktivierungsereignis ignoriert, das Deaktivierungsereignis als Empfangsereignis vor den Endpunkt des Diagramms und als Sendeereignis am Ende eines Plans notiert. Zusätzlich wird eine Referenz von der entsprechenden Autonomic Manager Rolle auf den Plan im Role Model eingetragen. Der Rest des Plans muss manuell erstellt werden.

6.4.3.10 Regel: TR_APM_AIM_01

Diese Regel (siehe Abbildung D.5) automatisiert Aktivität 15 des Entwicklungsprozesses komplett. Sie transformiert einen Nachrichtenaustausch zwischen zwei Partitionen im Autonomic Manager Plan Model in ein Interaktionspattern im Autonomic Manager Interaction Model. Die initiierende Partition wird zur initiierenden Rolle, die empfangende Partition zur partizipierenden Rolle. Die Referenz auf das Interaktionspattern wird bei der initiierenden Rolle eingetragen.

6.4.3.11 Regel: TR_APM_ASM_01

Diese Regel (siehe Abbildung D.6) automatisiert einen Teil der Aktivität 16 des Entwicklungsprozesses. Dazu wird eine interne Aktion im Plan einer Autonomic Manager Rolle in einen privaten Service transformiert und eine Referenz bei dieser Rolle im Autonomic Manager Role Model gesetzt.

6.4.3.12 Regel: TR_APM_MSM_01

Diese Regel (siehe Abbildung D.7) transformiert eine interne Aktion im Plan einer Managed Resource Rolle in einem Autonomic Manager Plan Model in einen privaten Service der Managed Resource Rolle und setzt eine Referenz bei dieser Rolle im Role Model.

6.4.3.13 Regel: TR_AIM_ASM_01

Diese Regel (siehe Abbildung D.8) automatisiert einen weiteren Teil der Aktivität 16 des Entwicklungsprozesses. Dazu wird eine Nachricht an eine Autonomic Manager Rolle in einem Interaktionspattern in einen öffentlichen Service der Autonomic Manager Rolle transformiert.

6.4.3.14 Regel: TR_AIM_MSM_01

Diese Regel (siehe Abbildung D.9) automatisiert den restlichen Teil der Aktivität 16 des Entwicklungsprozesses, so dass auch dieser vollautomatisiert ablaufen kann. Dazu wird eine Nachricht an eine Managed Resource Rolle in einem Interaktionspattern in einen geschützten Service der Managed Resource Rolle transformiert.

Für das *Interaction Model*, das *Autonomic Element Model* und das *Autonomic Element Instance Model* können noch keine bzw. nur triviale Transformationsregeln angegeben werden. Dazu liegen entweder noch zu wenige Forschungsergebnisse aus dem Bereich der Selbstorganisation vor, oder eine Transformation kann nicht deterministisch durchgeführt werden. Aus diesem Grund wird in dieser Arbeit auf die Definition weiterer Regeln verzichtet.

7. Evaluierung des Entwicklungsansatzes

Der in Kapitel 6 vorgestellte Entwicklungsprozess diente dort der Entwicklung eines selbstorganisierenden, stigmergischen OCS, welches die Funktionsweise des Produktionsplanungs- und -kontrollsystems aus Kapitel 4, insbesondere die Prinzipien der Stigmergie, besaß. Zusätzlich konnten durch den Entwicklungsprozess gewisse Selbst-x Eigenschaften für das System konsistent modelliert, erzeugt und beherrscht werden. Um überprüfen zu können, ob der Ansatz auch für die Entwicklung nicht-stigmergischer OCS geeignet ist, wird der Prozess in diesem Kapitel für die Entwicklung eines anderen Produktionsplanungs- und -kontrollsystem verwendet. Als Vorlage dient dazu eine weitere Fallstudie aus der Domäne der Fertigungssteuerung, welche im Gegensatz zur Fallstudie I auf einer direkten Kommunikation mittels Auktionen basiert. Aufgrund der Ergebnisse dieser Entwicklung wird der Entwicklungsansatz nach verschiedenen Gesichtspunkten evaluiert, wodurch eine Identifizierung von weiteren, notwendigen Forschungsbereichen für die Verwendung des Entwicklungsprozesses in einer Softwareentwicklungsmethodologie möglich wird.

7.1 Fallstudie II: Rechnergestützte Fertigungssteuerung

Aufgrund des Wandels in der Produktionsindustrie haben Bussmann und Schild ähnlich wie Valckenaers ein agenten-basiertes Produktionsplanungs- und -kontrollsystem [27] enwickelt, welches bei der DaimlerChrysler AG [3] bereits prototypisch eingesetzt wurde. Im Gegensatz zum System von Valckenaers verwenden die Systemkomponenten jedoch kein Selbstorganisationsmechanismus, sondern kommunizieren über Auktionen direkt miteinander. Trotz des Fehlens eines derartigen Mechanismuses weist das System Charakteristika der Selbstorganisation auf, deren Entstehung im Folgenden untersucht wird.

7.1.1 Systemarchitektur

Wie bei den meisten robusten Produktionssystemen haben auch bei diesem System die Maschinen überlappende Produktionsfähigkeiten. Im Gegensatz zur Fallstudie I ist das gesamte Produktionssystem hier jedoch nicht wie ein Graph aufgebaut, sondern besteht aus einzelnen Modulen (siehe Abbildung 7.1). Jedes dieser Module besteht aus einer *Maschine*, drei Einbahn-*Transportbändern*, zwei *Switches* und einer *Shifting Table*. Die Anordnung dieser sieben Komponenten ist in Abbildung 7.2 dargestellt.

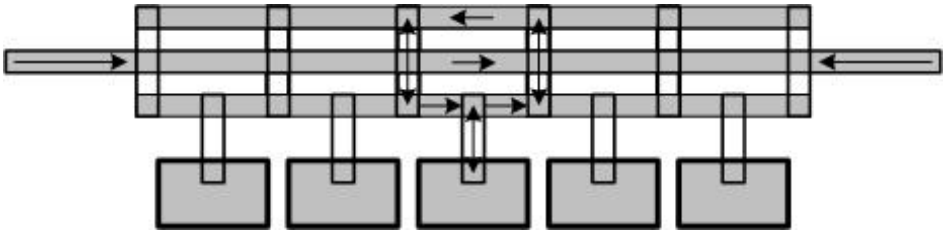

Abbildung 7.1: Architektur des Produktionsplanungs- und -kontrollsystems [27]

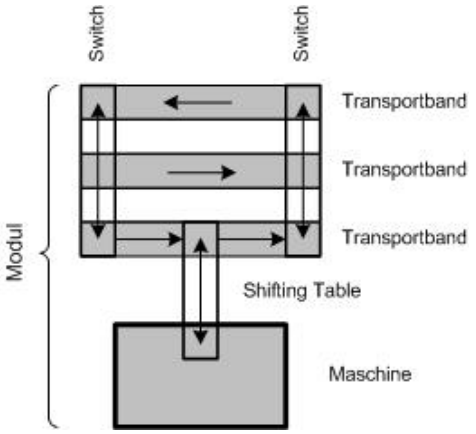

Abbildung 7.2: Struktur eines Moduls des Produktionsplanungs- und -kontrollsystems [27]

Jeder Switch kann eine Palette (mit einer Werkstück darauf) von einem beliebigen Eingang zu einem beliebigen Ausgang bewegen, wobei nur jeweils eine Palette zu einem bestimmten Zeitpunkt bewegt werden kann. In Abbildung 7.1 besitzt jeder innen liegende Switch zwei Eingänge und einen Ausgang auf seiner linken Seite und zwei Ausgänge und einen Eingang auf seiner rechten Seite. Eine derartige Anordnung der Module ermöglicht es einer Palette entweder über das unterste Transportband zu einer Maschine zu gelangen oder über das mittlere Transportband an ihr vorbei zu fahren. Fährt die Palette vorbei, stehen ihr zwei weitere Optionen offen: Die Palette kann entweder in die gleiche Richtung zu einer anderen Maschine weiterfahren oder über das oberste Transportband zurückfahren. Dadurch kann sich eine Palette solange im Kreis bewegen, bis das unterste Transportband, welches unter Umständen von einer anderen Palette belegt wurde, wieder frei ist und die Palette über die Shifting Table zu einer Maschine gelangen kann. Das gesamte Produktionssystem fungiert daher als eine Art flexibler Puffer.

7.1.2 Koordinationsmechanismus

Wie auch in der Fallstudie I ist die Kontrolle des Systems nicht zentral, sondern wird von den einzelnen Agenten übernommen. Dazu ist jeweils ein Agent jedem Werkstück, jeder Maschine und jedem Switch zugeordnet. Ein Werkstückagent kontrolliert dabei den Produktionszustand eines Werkstücks auf einer Palette, ein Maschinenagent den gesamten Materialfluss durch eine Maschine und ein Switchagent das Flussverhalten der Paletten, d. h. welcher Eingang zuerst beachtet wird und zu welchem Ausgang die Paletten bewegt werden. Für die Kontrolle des gesamten Materialflusses kontrolliert ein Maschinenagent nicht nur die aktuelle Prozessschrittausführung auf einer Maschine, sondern auch die ausgehenden Werkstücke, bis diese eine weitere Maschine für ihre Bearbeitung gefunden haben (virtueller Puffer).

Die Koordination all dieser simultanen, jedoch voneinander abhängigen Prozesse geschieht durch spezielle Verhandlungsprotokolle. Die Auktion zwischen den Werkstück- und Maschinenagenten besteht dabei aus einer einzelnen Runde und verläuft nach folgendem Auktionsprotokoll:

1. Die Auktion wird immer von einem Werkstückagenten angestoßen. Dies geschieht, wenn der Werkstückagent zum ersten Mal in das Produktionssystem eintritt bzw. unmittelbar nach dem Verlassen einer Maschine. In beiden Fällen bestimmt der Werkstückagent den nächsten, benötigten Prozessschritt für sein Werkstück nach einem vorgegebenen Produktionsplan und die Maschinen, auf denen dieser Prozessschritt möglicherweise ausgeführt werden kann. Anschließend sendet der Werkstückagent eine Aufforderung für Gebote an die ermittelten Maschinen. In dieser Aufforderung ist der benötigte Prozessschritt[1] enthalten.

2. Erhält ein Maschinenagent eine Gebotsaufforderung für einen Prozessschritt, überprüft er, ob seine Maschine den Prozessschritt erfüllen kann und der virtuelle Puffer eine weitere Palette aufnehmen kann. Ist dies der Fall, so erteilt der Maschinenagent ein Gebot; andernfalls ignoriert er die Aufforderung. Ein Gebot enthält immer (a) die aktuelle Auslastung des virtuellen Puffers und (b) den Prozessschritt[2], welchen die Maschine ausführen kann.

3. Der Werkstückagent sammelt alle Gebote für eine bestimmte Aufforderung auf. Sollte kein Gebot vorhanden sein, wird eine neue Aufforderung gesendet und von vorne begonnen; andernfalls erteilt er dem besten Gebot den Zuschlag. Dazu werden die Gebote anhand (a) und (b) verglichen, wobei (a) eine höhere Priorität hat. Je kleiner die aktuelle Auslastung eines virtuellen Puffers und je mehr Prozessschritte[3] in (b) enthalten sind desto besser.

Dieses Auktionsprotokoll entspricht dem Contract Net Protocol (CNP) [91]. Für gewöhnlich finden eine Vielzahl solcher Auktionen simultan statt und sind ineinander verschachtelt. Daher kann ein einzelner Maschinenagent in mehr als eine Auktion gleichzeitig verwickelt sein.

7.1.3 Produtkionsvorgang

Nach der Reservierung einer Maschine beginnt der Werkstückagent mit der Routenführung der Palette. Diese basiert auf einer Reihe von bilateralen Verhandlungen zwischen dem Werkstückagent und dem nächsten Switchagenten, welchen der Werkstückagent erreicht, solange bis der Werkstückagent die reservierte Maschine erreicht hat.

7.1.4 Selbstorganisierendes Verhalten

Das selbstorganisierende Verhalten des Systems zeigt sich an der Selbst-Limitierung. Das System passt sich immer dem momentanen Flaschenhals der Maschinen (die Maschine mit der aktuell niedrigsten Kapazität) an, um Staus im Transportsystem zu vermeiden. Der Flaschenhals des Systems wird automatisch in der entgegen gesetzten Materialflussrichtung propagiert,

[1] Der Einfachheit halber wird in dieser Fallstudie nur ein Prozessschritt auf einmal versteigert, in der Realität kann das System auch mehrere Prozessschritte auf einmal versteigern

[2] In der Realität sind auch hier mehrere Prozessschritte enthalten

[3] Hier nur einer

bis der Anfang der Produktionsstraße erreicht ist. Dort werden nur so viele Werkstücke in die Produktionsstraße gelassen, dass diese auch verarbeitet werden können, ohne dabei Staus zu erzeugen.

Der Grund dieser automatischen Propagation liegt in den virtuellen Puffern der einzelnen Maschinen. Wie bereits beschrieben, hält eine Maschine sowohl die gerade bearbeiteten Werkstücke als auch diejenigen, welche noch keine nächste Maschine ermittelt haben, in ihrem vrtuellen Puffer. Erreicht der Puffer einer Maschine seine Höchstgrenze, so werden keine weiteren Werkstücke mehr angenommen bzw. keine weiteren Gebote mehr erteilt. Da der Materialfluss in dem Produktionssystem vorwiegend linear ist[4], läuft der virtuelle Puffer einer oder mehrerer Maschinen, welche in Produktionsrichtung vor dem Flaschenhals des Systems liegen, ebenfalls voll. Dies setzt sich solange fort, bis der Anfang der Produktionsstraße erreicht ist. Indem dort nur noch begrenzt Werkstücke aufgenommen werden können, entstehen keine Staus im System, welche die Performance des Systems beeinträchtigen könnten. Dabei ist bemerkenswert, dass die Topologie des Systems und damit die Produktionsrichtung zufällig sein kann, da diese im Grunde genommen von den Bewegungen der Werkstücke aufgestellt wird.

7.2 Vergleich von Fallstudie I und II

Das in Fallstudie II (FS2) beschriebene Produktionsplanungs- und -kontrollsystem weist gegenüber dem System in Fallstudie I (FS1) in wesentlichen Punkten Unterschiede aber auch Gemeinsamkeiten auf, welche auf die anschließende Entwicklung einen Einfluss haben:

7.2.1 Systemarchitektur

Die Topologie der Produktionsstraße in FS2 ist im Gegensatz zu der in FS1 modular und nicht in einem Graphen angeordnet. Die Switches in FS2 können dabei mit den Kreuzungen aus FS1 gleichgesetzt werden. Auf die Entwicklung hat dies jedoch keinen nennenswerten Einfluss, da die Topologie keine Auswirkung auf die Funktionsweise eines der beiden Systeme hat und dort nicht festgelegt wird, wie viele Ein- und Ausgänge ein Switch bzw. eine Kreuzung besitzen muss. Bis auf die unterschiedlichen Namen sind auch die Agenten der beiden Fallstudien denselben Instanzen zugeordnet. Switch- und Maschinenagent in FS2 werden in FS1 als Ressourcenagent überschrieben. Lediglich der Produktagent aus FS1 tritt in FS2 nicht in Erscheinung. Dies hat auf die Entwicklung jedoch ebenfalls keinen gravierenden Einfluss, da die Rolle des Produktagenten aus FS1 in FS2 vom Werkstückagent mit übernommen werden kann.

7.2.2 Koordinationsmechanismus

Der deutlichste Unterschied liegt in der Art der Kommunikation. Während in FS1 für alle Interaktionen zwischen zwei Basisagenten Ameisenagenten als Übertragungsmedium verwendet werden, geschieht in FS2 jegliche Interaktion zwischen zwei Agenten direkt. Nichtsdestotrotz können die drei Kontrollstufen aus FS1 in FS2 wiedergefunden werden. Die Propagation der Produktionsfähigkeiten einer Maschine (*Feasibility Layer*) findet in FS2 nur implizit statt. Während in FS1 die *exploring ants* der Auftragsagenten die abgelegten Produktionsfähigkeiten der Maschinen an den Kreuzungen aufnehmen, wird in FS2 eine Maschine direkt im ersten

[4]Dies wird über einen hier nicht näher beschriebenen Mechanismus bei der Auswahl der Maschinen für eine Auktion erreicht

Auktionsschritt nach ihren Fähigkeiten gefragt. Die *exploring layer* und *intention layer* in FS1 sind zusammen im Auktionsprotokoll der FS2 enthalten. Das Einholen von Angeboten, das Bieten und das Reservieren ist in FS1 in mehrere Schichten aufgeteilt, in FS2 hingegen in einer Auktion zusammengefasst. Für die Entwicklung hat dieser Unterschied folglich Auswirkungen auf die Interaktionsprotokolle.

7.2.3 Selbstorganisation

In beiden Fallstudien werden neue oder ausgefallene Maschinen schnell bemerkt und erfordern keinen Eingriff von außen. In FS1 wird beispielsweise bei einem Maschinenausfall die Produktinstanz rechtzeitig an einer Kreuzung umgeleitet und an eine Maschine mit passenden Fähigkeiten für den nächsten Produktionssschritt weitergeleitet. In FS2 wird das Werkstück im Gegensatz dazu an einer ausgefallenen Maschine vorbeigeleitet um von einer anderen Maschine bearbeitet werden zu können. Da in FS2 bei jeder Auktion immer alle möglichen Maschinen betrachtet werden, stören hier ebenfalls keine neuen oder ausgefallenen Maschinen. Gewisse Eigenschaften hingegen hängen jedoch nur von der gewählten Kommunikationsart ab. So wird das System in FS2 kaum die Vorteile durch die Verwendung der Prinzipien der Stigmergie in FS1 aufweisen und umgekehrt. Für die Entwicklung haben diese Eigenschaften jedoch in dieser Arbeit keinen weiteren Einfluss, da sie von höheren Aspekten abhängen, welche hier nicht betrachtet werden können.

7.2.4 Sonstiges

Der virtuelle Puffer einer Maschine in FS2 entspricht der Auslastung einer Maschine in FS1. Die Verwaltung der von einer Maschine bearbeiteten Werkstücke in FS2 findet in FS1 in dieser Form nicht statt. Dies macht jedoch keinen wesentlichen Unterschied für die Entwicklung aus, da hier wiederum lediglich eine Rolle bzw. ein Teil einer Rolle einem anderen Agenten zugeteilt wird. Ein wesentlicher Unterschied ist jedoch, dass in FS1 durch die *exploring ants* immer eine komplette Produktionsroute gesucht wird, wohingegen in FS2 bei einer Auktion immer nur ein weiterer Schritt[5] der Produktion betrachtet wird. Dieser Unterschied ergibt sich allerdings aus der gewählten Kommunikationsart und ist als Vorteil der FS1 zu betrachten, hat aber keinen Einfluss auf die Entwicklung.

7.3 Anwendung des Entwicklungsprozesses

Im Folgenden wird der erstellte Entwicklungsprozess (siehe 6.2) für die Entwicklung eines Produktionsplanungs- und -kontrollsystem als Organic Computing System auf Vorlage des Systems aus Fallstudie II angewandt. Unter Berücksichtigung des Vergleichs in 7.2 werden dabei so weit wie möglich existierende Modelle aus der Entwicklung in 6.3 wiederverwendet.

Aufgrund der Tatsache, dass das zu entwickelnde System in derselben Domäne mit demselben Ziel in derselben angenommenen Umgebung eingesetzt werden soll, liefern die Aktivitäten 1–5 des Entwicklungsprozesses keine neuen Informationen gegenüber der vorangegangenen Entwicklung. Im CIM ergeben sich daher weder im *Business Context Model*, im *Business Process Model*, im *Environment Model*, im *Use Case Model* noch im *Ontology Model* Veränderungen, so dass alle Modelle bestehen bleiben können, von unterschiedlichen Bezeichnungen abgesehen.

[5] In der Realität zwar mehrere, jedoch selten alle

Die Aktivitäten 6–10 führen im PIM beim *Role Model*, *Norm Model*, *Plan Model*, *Interaction Model* und *Service Model* ebenfalls von unterschiedlichen Bezeichnungen abgesehen zu keinen Veränderungen. Da die unterschiedlichen Bezeichnungen weder für die Modelle, die Transformationen noch für die Funktionsweise des Systems eine Rolle spielt, besteht keine Veranlassung für eine neue Modellierung dieser Modelle. Die Aktivitäten 11-16 führen auf der Seite der applikationsspezifischen Selbst-x Entwicklung aufgrund der Fallstudie II logischerweise zu keinen Veränderungen, da das beschriebene System nicht über Selbst-x Eigenschaften verfügt. Daher können die Autonomic Manager Rollen *resource* und *production order* und die in 6.3 entwickelten Selbst-x Eigenschaften bestehen bleiben.

Der Vergleich der beiden Fallstudien hat erst für die Interaktionsprotokolle notwendige Änderungen festgestellt, so dass Aktivität 17 des Entwicklungsprozesses zu einem anderen Interaction Protocol Model führt. Dort müssen die bestehenden, stigmergischen Interaktionsprotokolle ausgetauscht und durch passende Auktionsprotokolle ersetzt werden, welche jedoch die bestehenden Interaktionspattern implementieren. Abbildung B.9 zeigt ein angepasstes Contract Net Protocol, welches die Interaktionspattern *request offer* (siehe Abbildung 6.15) und *reserve resource* (siehe Abbildung B.4) als Interaktionsprotokoll implementieren. Ähnlich wie in objektorientierten Programmiersprachen kann ein einziges Interaktionsprotokoll (Implementierung) mehrere Interaktionspattern (Schnittstellen) auf einmal implementieren. Aktivität 18 und 19 führen hingegen zu keiner Veränderung für das *Autonomic Element Model* und das *Autonomic Element Instance Model*, da auch diese unabhängig von der Art der Kommunikation sind.

7.4 Bewertung des Entwicklungsansatzes

Die Ergebnisse der Entwicklungen in 6.3 und 7.3 lassen bereits in einigen Punkten Rückschlüsse über die Qualität des Entwicklungsansatzes zu.

7.4.1 Unterstützung des MDA-Ansatzes

Der Entwicklungsansatz hält sich an die Vorgaben der Model Driven Architecture und ihre definierten Modellen. Vergleicht man die beiden Entwicklungen, so wurden keine Änderungen im CIM nötig. Dies ist insofern wichtig, da diese Ebene softwareunabhängig sein soll und es folglich keine Rolle spielt, ob ein OCS über direkte oder indirekte Kommunikation interagiert. Auch im PIM wurden bis auf ein Modell keine Veränderungen zwischen den beiden Entwicklungen festgestellt. Lediglich das Interaction Protocol Model musste für eine Änderung der Kommunikationsart ausgetauscht werden, was aber zulässig ist, da das PIM wie beschrieben nur plattformunabhängig sein soll.

Wie weiter zur erkennen ist, wurde bis zum Autonomic Element Instance Model auf der PIM-Ebene noch nicht festgelegt, welche Technologien das System – hier das PPCS – verwenden soll. Diese Entscheidung muss erst für die Transformation zwischen PIM und PSM getroffen werden. Im vorliegenden Fall könnten beispielsweise Plattform Models der Agententechnologie eingesetzt werden um eine entsprechende Realisierung des PPCS als Multi-Agenten-System zu erhalten. Dabei wäre denkbar, dass jedes Autonomic Element von einem Agenten implementiert werden würde, aber auch, dass jede Managed Resource und jeder Autonomic Manager durch einen eigenen Agenten realisiert werden könnten. Es können aber genauso gut andere Technologien eingesetzt werden, welche gewisse Anforderungen an verteilte Systeme erfüllen.

Für die Transformation des PIM in ein PSM bietet der Ansatz eine große Unterstützung, da jedes Modell der PIM-Ebene für sich transformiert werden kann. Durch die Einführung von Referenzen auf der PIM-Ebene zwischen den Role Models und den davon abhängigen Modellen könnte jedes Modell in ein eigenes PSM transformiert werden, ohne dass Zusammenhänge verloren gehen. Die Referenzen auf der PIM-Ebene können wo nötig zu Bridges zwischen den PSM und später zwischen den verschiedenen Codes transformiert werden. Mit der Definition von Modelltransformationen auf CIM- und PIM-Ebene sowie zwischen den beiden Ebenen werden bereits Hinweise auf die Art der Transformationsregeln gegeben. Erst durch die Definition der Regeln wird das Systemkonstruktionsparadigma der MDA erfüllt.

7.4.2 Verwendung und begrenzte Erweiterung des UML 2.0 Standards

Die Modellierung basiert bis auf wenige Notationssymbole ausschließlich auf UML 2.0 und die darin definierten Diagramme. Lediglich für Rollen, Normen und Services mussten neue Notationen definiert werden um den Standard begrenzt zu erweitern.

Ein neues Konzept, welches in UML 2.0 in dieser Form nicht enthalten ist, stellt die Referenzierung auf andere Modellelemente dar. Zwar können in einem Seuquenzdiagramm verschiedene Aspekte ausgelassen werden und mit einem *ref* auf eine andere Modellierung verweisen, jedoch ist dies die einzige Stelle in UML 2.0, wo dies bisher verwendet wird. Mit der Erweiterung auf eine allgemeine Referenzierung lassen sich Konzepte wie Modularisierung und Kapselung bereits auf Modellebene realisieren. Der Vorteil der Referenzierung zeigt sich an mehreren Punkten, welche in den beiden Entwicklungen deutlich zum Tragen kamen:

- Im Gegensatz zu vielen anderen Entwicklungsmethodologien wird durch die Referenzierung ein Zusammenhang zwischen den Modellen hergestellt, welcher bei großen Entwicklungen schnell verloren gehen kann. Die Ansammlung mehrerer Modelle auf den Ebenen der MDA würde zwar viele Teilaspekte des zu entwickelnden Systems preisgeben, es jedoch schwer machen, den Überblick zu bewahren und Zusammenhänge zu erkennen.

- Durch die Referenzierung wird eine wiederholte Modellierung von Teilaspekten vermieden, da von unterschiedlichen Modellen auf dieselbe Stelle referenziert werden kann. In vielen Systemen – insbesondere Organic Computing Systemen – müssen verschiedene Komponenten beispielsweise gleiche Basisdienste zur Verfügung stellen, welche dadurch nicht redundant modelliert werden müssen. Das Konzept der Modularisierung wird in Zukunft immer wichtiger werden, da so auf Modelle verwiesen werden kann, welche nicht Bestandteil der eigenen Entwicklung sind, sondern eventuell als Modellbibliothek extern verfügbar sind.

- Das Konzept der Kapselung spielt beim Zusammenspiel von Interaktionspattern und Interaktionsprotokollen eine Rolle, welches unten näher ausgeführt wird.

7.4.3 Umsetzung der Selbstorganisation

Der Entwicklungsansatz erfüllt zumindest für die beiden Entwicklungen die in 3.1 beschriebenen theoretischen Anforderungen und Charaktersitika selbstorganisierender Systeme. So ist es möglich, den Selbstorganisationsmechanismus der Stigmergie mittels Ameisenalgorithmen zu realisieren. Daran ist zu erkennen, wie beispielsweise die Kontrolle des Systems

verteilt ist, ein globales Verhalten aus den Interaktionen der Agenten entsteht oder wie robust das System gegenüber Ausfällen ist. Der Großteil der Prinzipien der Stigmergie wurde dabei von den Interaktionsprotokollen in der Entwicklung realisiert, welche die Interaktionspattern implementieren.

Durch den simplen Austausch dieser Interaktionsprotokolle können ohne großen Aufwand andere Prinzipien umgesetzt werden, wie an der Entwicklung des Systems aus Fallstudie II zu erkennen ist. Durch die Verwendung von Auktionen als direkte Kommunikation können andere Selbstorganisationseigenschaften realisiert werden, welche allein auf der Art der Kommunikation basieren. Genauso leicht können beispielsweise Auktionsprotokolle ausgetauscht werden oder die Rollen um weitere Protokolle ergänzt werden. Indem ein Interaktionspattern von mehreren, verschiedenen Interaktionsprotokollen implementiert wird, kann eine Rolle bzw. später ein Autonomic Element zwischen diesen wählen, um seine eigenen Interessen besser zu wahren oder mit anderen Autonomic Elements zu kommunizieren, welche eventuell nur wenige Protokolle unterstützen. Beispielsweise könnte das Contract Net Protocol aus der Fallstudie II auch durch ein Protokoll der englischen, holländischen oder Vickrey Auktion ersetzt oder ergänzt werden[6].

7.4.4 Verarbeitung unbekannter Situationen

Wie in 2.1.2 erwähnt, ist in der Forschungsgemeinschaft des Organic Computing noch nicht klar, wie die gegenläufigen Tendenzen zwischen Top-Down-Kontrolle und kreativem Bottom-Up-Verhalten in der Softwareentwicklung miteinander vereinbart werden können. Selbst wenn es auf den ersten Blick so aussieht, als würde der erstellte Entwicklungsansatz durch die Verwendung der MDA ein klarer Vertreter des Top-Down-Ansatzes sein, so trügt der Schein. Dieser Ansatz lässt auch Systemzustände zu, welche nicht bei der Entwicklung berücksichtigt werden können. Dazu wurde das Konzept der *Non-Cooperative Situations* mit aufgenommen, um unbekannte Situationen erkennen zu können. Der Umgang mit neuen Situationen wird durch Planungsalgorithmen geregelt, welche versuchen, die Situationen zu lösen, so dass existierende Normen entweder wieder eingehalten oder verändert bzw. abgeschafft werden müssen. Der Entwicklungsansatz stattet Autonomic Elements dabei nur mit initialen Plänen aus, welche zur Lösung von bekannten Situationen verwendet werden können. Neue Pläne müssen von den Autonomic Elements selber erstellt werden, wobei erstere auf bestehenden Plänen basieren können. Diese Fähigkeit des Lernens steht jedoch nicht im Fokus dieser Arbeit.

7.4.5 Ergänzung von Selbst-x Eigenschaften für bestehende Systeme

Bestehende Systeme, welche keine Selbst-x Eigenschaften besitzen, können mit diesem Entwicklungsansatz begrenzt mit solchen Eigenschaften ausgestattet werden. Falls es die Technologie des bestehenden Systems zulässt, können durch die Trennung der Entwicklung von Systemarchitektur und Selbst-x Eigenschaften unter Umständen diese Eigenschaften hinzugefügt werden. Dazu muss das bestehende System erweiterbar sein, gut dokumentiert sein und eine Kontrolle von Autonomic Managern unterstützen. Nichtsdestotrotz ist dieser Weg eher unorthodox und wohl nicht effizient.

[6] Da diese Protokolle im Gegensatz zum Contract Net Protocol aus mehreren Runden bestehen, würden sie jedoch für die Fallstudie II keinen Vorteil in der Performance liefern

8. Zusammenfassung und Ausblick

In dieser Arbeit wurde mit der Erstellung eines modellbasierten Entwicklungsprozesses für Organic Computing Systeme die Grundlage für eine konsistente und durchgängige Softwareentwicklungsmethodologie für diese Systeme geschaffen. Durch die Untersuchung von Selbstorganisationsmechanismen in der Natur – insbesondere der Stigmergie –, der Analyse eines selbstorganisierenden, stigmergischen IT-Systems und der technischen Erzeugung von Selbst-x Eigenschaften im Autonomic Computing konnten dazu wesentliche Anforderungen an die Architektur von Organic Computing Systemen identifiziert werden. Diese wurden in einem Metamodell für OCS umgesetzt, welches zur Erfüllung der identifizierten Anforderungen auch geeignete Konzepte aus Entwicklungsmethodologien für Multi-Agenten-Systeme verwendet und als Basis für den Entwicklungsprozess dient. Mit der Definition von Modelltransformationen wurde zudem dafür gesorgt, dass das Framework der Model Driven Architecture unterstützt wird. Weiter wurde exemplarisch vorgeführt, wie der Prozess für die Entwicklung eines stigmergischen OCS eingesetzt werden kann und wie Selbst-x Eigenschaften für dieses System modelliert werden können. Durch die Evaluierung des Entwicklungsansatzes an einer Fallstudie konnte gezeigt werden, dass der Prozess nicht nur für die Entwicklung von OCS verwendbar ist, welche indirekt miteinander kommunizieren, sondern auch für OCS, welche direkt miteinander kommunizieren. Weitere Ergebnisse der Evaluierung waren, dass der Ansatz auf dem Standard von UML 2.0 basiert, diesen wo nötig begrenzt erweitert und dass die zugrunde liegende Architektur mit unbekannten und nicht im Vornherein geplanten Situationen umgehen kann.

Der Entwicklungsansatz verwendet und erweitert zudem aktuelle Forschungsansätze im Organic Computing und kann dadurch auch mit anderen Forschungsprojekten kombiniert werden. So wurde durch die Verwendung des Autonomic Manager-Konzepts im OCS-Metamodell eine Basis für die Entwicklung einer mehrschichtigen Observer/Controller-Architektur geschaffen. Für die Realisierung und Implementierung können später Constraints auf der PSM-Ebene und Assertions auf der Code-Ebene des MDA-basierten Ansatzes genutzt werden. Letztere können für die Zusicherung von Zuständen bzw. das Melden von fehlerhaften Zuständen eingesetzt werden, so dass durch ausgegebene Warnungen (Exceptions) ein Autonomic Manager entsprechend reagieren kann, um so einen Teil für die Erhaltung der Selbst-Eigenschaften beizutragen.

Des Weiteren zeigt der Ansatz, wie die Selbst-x Eigenschaften eines OCS durch die Interaktionen zwischen einzelnen Systemkomponenten, vorwiegend zwischen Autonomic Managers und Managed Resources, entstehen und sich nicht allein durch die Gesamtheit aller Systemkomponenten erzeugen lassen. In diesem Sinne sind die Selbst-x Eigenschaften als emergent zu betrachten. Durch die Untersuchung natürlicher und künstlicher, selbstorganisierender Systeme wurde allerdings festgestellt, dass die Verwendung eines Selbstorganisationsmechanismuses für die Kommunikation der Systemkomponenten, wie z. B. der Stigmergie, nicht zur Entstehung der Selbst-x Eigenschaften beitragen kann. Da die Stigmergie nur die applikationsspezifische Art der Kommunikation festlegt, kann sie zwar ein selbstorganisierendes bzw. emergentes Verhalten des Systems hervorrufen, jedoch nicht dessen Selbst-x Eigenschaften explizit erzeugen. Für diese Erzeugung wurden andere Mechanismen gefunden, welche der Entwicklungsansatz mit der Verwendung von Normen berücksichtigt.

Durch die Spezifikation von (low-level) Normen kann das Verhalten des Systems bereits zur Entwicklungszeit genau festgelegt werden, so dass das Verhalten zur Laufzeit die gewünschten Ziele erreicht. Nichtsdestotrotz existiert gerade in diesem Bereich noch großer Forschungsbedarf. Da die Wissenschaft der Selbstorganisation im Vergleich zu anderen Wissenschaften noch recht jung ist, fehlen trotz bereits guter Ergebnisse – wie die Untersuchung der Nahrungssuche bei Ameisenkolonien zeigte – noch viele Informationen über verschiedene Zusammenhänge. Bleibt man bei dem Beispiel der Stigmergie, so ist es zwar möglich, einzelne Managed Resources mit spezifischen Normen auszustatten, um ein exploratives und emergentes Gesamtverhalten eines System zu erzeugen, jedoch ist noch nicht klar, wie die Rückrichtung funktioniert, d. h. wie man von dem gewünschten Gesamtverhalten eines Systems auf die spezifischen Normen einer einzelnen Komponente kommt. Solange diese Forschungsergebnisse fehlen, werden die low-level Normen weiterhin für jede einzelne Komponentenart nach bestehendem Wissen festgelegt. Gleiches gilt für Autonomic Managers: Durch die gezielte Spezifikation von low-level Normen für einzelne Autonomic Managers lassen sich gewisse Selbst-x Eigenschaften realisieren. Jedoch werden auch hier Forschungsergebnisse benötigt, welche das Zusammenspiel von Normen verschiedener Autonomic Managers erklären und so Rückschlüsse auf die Spezifikation von Normen einzelner Autonomic Managers zulassen.

Der Entwicklungsansatz berücksichtigt diese fehlenden Ergebnisse bereits und sieht eine zukünftige Erweiterung um (high-level) Normen vor. Sobald die Ergebnisse verfügbar sind, können high-level Normen auf der CIM-Ebene in einem weiteren Modell definiert werden und durch die dann möglich gewordene, automatische Transformation auf die PIM-Ebene abgebildet und so für Rollen als low-level Normen zugänglich gemacht werden. Die Ergebnisse werden auch für die automatische Ableitung von Rollen aus höheren Modellen dienlich sein. In gleicher Weise hängt die Adaptivität eines Systems von diesen Ergebnissen ab. Da erstere durch das komponenteneigene Aufstellen neuer Normen realisiert wird (Selbst-Optimierung), muss die Möglichkeit einer Kontrolle gefunden werden, welchen Einfluss die Adaption einer Norm auf das Gesamtverhalten eines verteilten Systems besitzt.

Es gibt allerdings einige Aspekte, welche nicht von diesen Forschungsergebnissen abhängen, jedoch vor dem Einsatz des Entwicklungsprozesses noch berücksichtigt werden müssen. So stammt ein Teil der identifizierten Anforderungen an die Architektur von OCS aus einem selbstorganisierenden, stigmergischen IT-System, welches mit Agenten realisiert wurde und der Domäne der Fertigungssteuerung angehört. Dies deckt zwei weitere Untersuchungsgebiete auf: Zum einen muss analysiert werden, welche Technologien sich neben den Agenten für eine

Realisierung von OCS einsetzen lassen, zum anderen, ob die identifizierten Anforderungen auch für weitere Domänen bzw. weitere Selbstorganisationsmechanismen gelten. Beide Untersuchungen haben Einfluss auf die weiteren Schritte im Entwicklungsprozess, besonders für die Modelle auf PSM- und Code-Ebene. Nach diesen Untersuchungen muss die praktische Umsetzbarkeit des Ansatzes gezeigt werden, indem ein OCS vollkommen neu entwickelt und nicht allein für das Reengineering eines bestehenden Systems verwendet wird. Dadurch kann geprüft werden, ob die in dem Metamodell verwendeten Konzepte den gewünschten Nutzen bringen oder welche Konzepte ausgetauscht bzw. ergänzt werden müssen.

Der erfolgreiche Einsatz des Entwicklungsprozesses hängt zudem stark vom Fortschritt bei der Definition von Modelltransformationen ab. Hier liegt ein großes Potential für die Zukunft, welches erst durch die Verwendung der MDA freigelegt wird. Je mehr Transformationen definiert werden können, desto mehr Aktivitäten können im Entwicklungsprozess von Werkzeugen automatisiert werden und desto weniger Aufwand entsteht für einen Entwickler. Dieses Prinzip ist vergleichbar mit der zeitlichen Entwicklung der Programmiersprachen. Zu anfangs existierten lediglich maschinennahe Sprachen, wie z. B. Assembler. Durch die Hinzunahme von immer abstrakteren Konstrukten entstanden höherwertige Sprachen, deren automatische Transformation auf eine maschinennahe Sprache ein Compiler übernahm und so einem Entwickler die Arbeit erleichterte. Je weniger Aufwand für die Entwicklung von OCS betrieben werden muss, desto höher wird die Akzeptanz des Entwicklungsprozesses sein.

Trotz der wichtigen Grundlagen, welcher dieser Ansatz für die Zukunft bereitstellt, darf eine ethische und moralische Betrachtung der Ergebnisse nicht fehlen. Die durch den Ansatz möglich gewordene, gezielte Entwicklung von Selbst-x Eigenschaften von Organic Computing Systemen und die dadurch entstehende Entlastung von Administratoren erfüllen zwar die gesetzten Vorgaben des Organic Computing, müssen jedoch auch kritisch gesehen werden. Können durch die Autonomität der OCS Arbeitsplätze gefährdet werden? Können mit diesem Ansatz OCS entwickelt werden, welche durch Lernen intelligenter werden als der Entwickler selber? Können OCS durch ihre Autonomität von alleine weitere OCS entwickeln und diese gegen den Entwickler oder gar gegen die Menschheit einsetzen? Viele Naturwissenschaftler und Philosophen haben sich mit Fragen dieser Art im Bereich der künstlichen Intelligenz bereits auseinander gesetzt und sind zu verschiedenen Ergebnissen gekommen (siehe [90]). Aus einem pessimistischen Blickwinkel heraus betrachtet kann meines Erachtens jede Erfindung negative Folgen haben. Weiß man hingegen um die Gefahren, so lassen sie sich bei der zukünftigen weiteren Entwicklung der positiven Folgen berücksichtigen und umgehen, so dass eine Erfindung – insbesondere dieser Ansatz – zum Wohle der Allgemeinheit genutzt werden kann.

Literaturverzeichnis

[1] *http://www.cs.rmit.edu.au/agents/SAC/methodology.shtml*

[2] *http://www.eurescom.de/public/projects/P900-series/P907*

[3] *http://www.daimlerchrysler.com*

[4] *http://www.ibm.com*

[5] *http://www-106.ibm.com/developerworks/autonomic/overview.html*

[6] *http://www-306.ibm.com/software/awdtools/rup/*

[7] *http://www.isscc.org/isscc/*

[8] *http://java.sun.com*

[9] *http://www.organic-computing.org*

[10] *http://www.omg.org/mda*

[11] *http://www.omg.org*

[12] *http://www.troposproject.org/*

[13] Banatre J.P., D. Le M'etayer: *The Gamma model and its discipline of programming*, Science of Computer Programming, Band 15, S. 55–77, 1990.

[14] Bauer B., J.P. Müller: *Methodologies and Modelling Languages*, in: Luck M., R. Ashri und M. d'Inverno (Hrsg.): Agent-Based Software Development, S. 77–131, Artech House, 2004.

[15] Bauer B., J.P. Müller und J. Odell: *Agent UML: A formalism for specifying multiagent software systems*, in: Ciancarini P., M. Wooldridge (Hrsg.): Agent-Oriented Software Engineering – Proceedings of the First International Workshop (AOSE 2000). Springer-Verlag, Berlin, Deutschland, 2000.

[16] Beck K.: *Extreme Programming Explained: Embrace Change*, Boston, Addison-Wesley, 2000.

[17] Bernon C., M. Cossentino, M. Gleizes, P. Turci und F. Zambonelli: *A study of some multi-agent meta-models*, in: Proceedings of AOSE 2004, New York, USA, Juli 2004.

[18] Bernon C., M. Gleizes, S. Peyruqueou und G. Picard: *ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering*, in: Proceedings of ESAW 2002, S. 156–169, Madrid, Spanien, September 2002.

[19] Bernon C., M. Gleizes, G. Picard und P. Glize: *The Adelfe Methodology For an Intranet System Design*, in: Proceedings of AOIS 2002, Toronto, Kanada, Mai 2002.

[20] Berry G., G. Boudol: *How to write Parallel Programs*, The Chemical Abstract Machine, Theoretical Computer Science, Band 96, S. 217–248, 1992.

[21] Born M., E. Holz und O. Kath: *Softwareentwicklung mit UML 2*, Addison-Wesley, München, 2004.

[22] Brinkschulte U., J. Becker und T. Ungerer: *CARUSO - An Approach Towards a Network of Low Power Autonomic Systems on Chips for Embedded Real-time Application*, IPDPS, 2004.

[23] Bullnheimer B., R.F. Hartl und C. Strauss: *An Improved Ant System Algorithm for the Vehicle Routing Problem*, in: Dawid, Feichtinger und Hartl (Hrsg.): Annals of Operations Research, Nonlinear Economic Dynamics and Control, 1999.

[24] Bullnheimer B., R.F. Hartl und C. Strauss: *Applying the Ant System to the Vehicle Routing Problem*, In: Voss S., Martello S., Osman I.H. und Roucairol C. (Hrsg.): Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer: Boston, 1999.

[25] Buschmann F., R. Meunier, H. Rohnert und M. Stal: *Pattern-orientierte Software-Architektur*, München, Addison-Wesley, 1998.

[26] Bussmann S., D. McFarlane: *Rationales for holonic manufacturing control*, in: Van Brussel H., P. Valckenaers (Hrsg.): Proceedings of the 2nd Int. Workshop on Intelligent Manufacturing Systems, S. 177–184, 1999.

[27] Bussmann S., K. Schild: *Self-Organizing Manufacturing Control: An Industrial Application of Agent Technology*, in: Proceedings of the Fourth International Conference on Multi-Agent Systems, S. 87-94, Boston, MA, USA, 2000.

[28] Caire G., F. Leal, P. Chainho, R. Evans, F. Garijo, J. Gomez, J. Pavon, P. Kearney, J. Stark und P. Massonet: *Agent Oriented Analysis using MESSAGE/UML*, in: Proceedings AOSE 2001, S. 101–108, April 2001.

[29] Carriero N., D. Gelernter: *How to write Parallel Programs*, MIT Press, 1990.

[30] Cockburn A.: *Agile Software Development*, Boston, Addison-Wesley, 2002.

[31] Colorni A., M. Dorigo, V. Maniezzo und M. Trubian: *Ant System for Job-shop Scheduling*, JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science, band 34(1), S. 39–53, 1994.

[32] Cossentino M., C. Potts: *A CASE tool supported methodology for the design of multi-agent system*, in: Proceedings of SERP 2002, Las Vegas, USA, 2002.

[33] Costa D., A. Hertz: *Ants Can Colour Graphs*, Journal of the Operational Research Society, Band 48, S. 295–305, 1997.

[34] Crutchfield J.P.: *Is Anything Ever New? Considering Emergence*, SFI Series in the Science of Complexity XIX, Addison-Wesley, 1994.

[35] DeLoach S.A., Wood M.F.: *Multiagent Systems Engineering: The Analysis Phase*, Technical Report, Air Force Institute of Technology, AFIT/EN-TR-00-02, Juni 2000.

[36] Deneubourg J.-L., S. Aron, S. Goss, J.-M. Pasteels: *The self-organizing exploratory pattern of the argentine ant*, in: Journal of Insect Behaviour, 3. Jg., S. 159–168, 1990.

[37] Department of Energy, Office of Defense Energy Projects and Special Applications: *Strategic Defense Initiative Multimegawatt Space Nuclear Power Program – Summary*, April 1986.

[38] Di Caro G., M. Dorigo: *AntNet: A Mobile Agents Approach to Adaptive Routing*, Tech. Rep. IRIDIA/97-12, Université Libre de Bruxelles, Belgien, 1997.

[39] Di Caro G., M. Dorigo: *AntNet: Distributed Stigmergetic Control for Communications Networks*, Journal of Artificial Intelligence Research (JAIR), Band 9, S. 317–365, 1998.

[40] Di Marzo Serugendo G., N. Foukia, S. Hassas, A. Karageorgos, S.K. Mostéfaoui, O.F. Rana, M. Ulieru, P. Valckenaers und C. Van Aart (Hrsg.): *Self-Organisation: Paradigms and Applications*, Engineering Self-Organising Systems 2003, S. 1–19, 2004.

[41] Dorigo M., L.M. Gambardella: *Ant Colonies for the Traveling Salesman Problem*, in: BioSystems, Band 43, S. 73–81, 1997. (Auch als Technical Report TR/IRIDIA/1996-3, IRIDIA, Université Libre de Bruxelles.)

[42] Dorigo M., V. Maniezzo und A. Colorni: *Ant System: An autocatalytic optimizing process*, Working Paper No. 91-016 Revised, Politecnico di Milano, Italien 1991.

[43] Eigen M., P. Schuster: *The Hypercycle: a principle of natural self-organization*, Berlin, Heidelberg, New York, Springer, 1979.

[44] Gambardella L.M, M. Dorigo: *An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem*, INFORMS Journal on Computing, Band 12(3), S. 237–255, 2000.

[45] Gambardella L.M., M. Dorigo: *Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem*, in: Prieditis A., S. Russell und M. Kaufmann (Hrsg.): Proceedings of ML-95, Twelfth International Conference on Machine Learning, Tahoe City, CA, S. 252–260, 1995.

[46] Gambardella L.M, M. Dorigo: *Solving Symmetric and Asymmetric TSPs by Ant Colonies*, ICEC96, Proceedings of the IEEE Conference on Evolutionary Computation, Nagoya, Japan, 20.-22. Mai, 1996.

[47] Gesellschaft für Informatik: *VDE/ITG/GI-Positionspapier zum Organic Computing*, 2003 (http://www.gi-ev.de/download/VDE-ITG-GI-Positionspapier%20Organic%20Computing.pdf).

[48] Glansdorff P., I. Prigogine: *Thermodynamic study of structure, stability and fluctuations*, Wiley, New York, 1978.

[49] Goss S., S. Aron, J.-L. Deneubourg, J.-M. Pasteels: *Self-organized shortcuts in the argentine ant*, in: Naturwissenschaften, 76. Jg., S. 579–581, 1989.

[50] Glaser N.: *Contribution to Knowledge Modelling in a Multi-Agent-Framework (the CoMoMAS approach)*, PhD thesis, L'Universtit' e Henri Poincar'e, Nancy I, Frankreich, November 1996.

[51] Grasse P.P.: *La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp.*, La theorie de la stigmergie: essai d' interpretation du comportement des termites constructeurs, Insectes Sociaux 6, S. 41–81, 1959.

[52] Giunchiglia F., J. Mylopoulos und A. Perini: *The Tropos Software Development Methodology: Processes, Models and Diagrams*, in: Proceedings AOSE 2002, S. 162–173, 2001.

[53] Hadeli K., P. Valckenaers, C. Zamfirescu, H. Van Brussel, B. Saint Germain, T. Holvoet und E. Steegmans: *Self-Organising in Multi-agent Coordination and Control Using Stigmergy*, Engineering Self-Organising Systems 2003, S. 105–123, 2003.

[54] Heylighen F.: *The Science of Self-organization and Adaptivity*, in: The Encyclopedia of Life Support Systems, (EOLSS Publishers Co. Ltd), 2001.

[55] Horn P.: *Autonomic computing: IBM perspective on the state of information technology*, IBM T.J. Watson Labs, NY, 15th October 2001. Presented at AGENDA 2001, Scotsdale, AR (http://www.research.ibm.com/autonomic/).

[56] IBM: *An architectural blueprint for autonomic computing*, 2004 (http://www-306.ibm.com/autonomic/pdfs/ACBP2_2004-10-04.pdf).

[57] IEEE: *Standard Glossary of Software Engineering Terminology*, 610.12-1990, 2002.

[58] Iglesias C.A., M. Garijo, J.C. González und J.R. Velasco: *A Methodological Proposal for Multiagent Systems Development extending CommonKADS*, in: Proceedings of 10th KAW, Banoe, Canada, 1996.

[59] Jeckle M., C. Rupp, J. Hahn, B. Zengler und S. Queins: *UML 2 glasklar*, Hanser Fachbuchverlag, November 2003.

[60] Juan T., A. Pearce und L. Sterling: *ROADMAP: Extending the Gaia Methodology for Complex Open Systems*, in: Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), S. 310 ff., Bologna, Italien, Juli 2002.

[61] Kephart J.O., D.M. Chess: *The Vision of Autonomic Computing*, IEEE Computer, S. 41-50, Januar 2003.

[62] Kohonen T.: *Self-Organizing Maps*, Springer Series in Information Sciences, Band 30, Springer, 3. Auflage, 2001.

[63] Kollingbaum M., T. Heikkilä, P. Peeters, J. Matson, P. Valckenaers, D. McFarlane und G.-J. Bluemink: *Rationales for holonic manufacturing control*, Emergent Flow Shop Control based on MASCADA Agents, MIM 2000, Patras, Griechenland, 2000.

[64] Kollingbaum, M.J., T.J. Norman: *NoA - A Normative Agent Architecture*, IJCAI 2003.

[65] Kreuzinger J., A. Schulz, M. Pfeffer, T. Ungerer, U. Brinkschulte und C. Krakowski: *Real-time Scheduling on Multithreaded Processors*, Real-Time Computing Systems and Applications (RTCSA), Cheju Island, South Korea, S. 155–159, Dezember 2000.

[66] Lawler E.L., J.K. Lenstra, A.H.G. RinnooyKan, D.B. Shmoys (Hrsg.): *The Travelling Salesman Problem*, New York: Wiley, 1985.

[67] Maniezzo V., A. Colorni und M. Dorigo: *The Ant System Applied to the Quadratic Assignment Problem*, Tech. Rep. IRIDIA/94-28, Université Libre de Bruxelles, Belgien, 1994.

[68] Meyer B.: *Object-oriented software construction*, Prentice Hall, 2. Auflage, 2000.

[69] Moore G.E.: *Cramming More Components onto Integrated Circuits*, Electronics Magazine, Band 38, S. 114–117, April 1965.

[70] Müller-Schloer C., C. von der Malsburg und R.P. Würtz: *Organic Computing*, in: Informatik Spektrum, Band 27, Heft 4, S. 332–336, 2004.

[71] Mylopoulos J., M. Kolp und J. Castro: *UML for Agent-Oriented Software Development: The Tropos Proposal*, in: Proceedings of UML 2001, Toronto, Canada, S. 422–441, Oktober 2001.

[72] Newton I.: *Philosophiae Naturalis Principia Mathematica*, I.B. Cohen, A. Koyre (Hrsg.), Harvard University Press, 3. Auflage, 1990.

[73] Object Management Group: *Common Warehouse Metamodel (CWM) Version 1.1*, OMG Document: formal/03-03-02, März 2003.

[74] Object Management Group: *Meta Object Facility (MOF) Version 1.4*, OMG Document: formal/02-04-03, April 2002.

[75] Object Management Group: *MDA Guide Version 1.0.1*, OMG Document omg/03-06-01, Juni 2003.

[76] Object Management Group: *Model Driven Architecture: A Technical Perspective*, ormsc/01-07-01, Juli 2001.

[77] Object Management Group: *Unified Modeling Language (UML) Version 1.5*, OMG Document: formal/03-03-01, März 2003.

[78] Object Management Group: *UML Profile for Schedulability, Performance, and Time Version 1.0*, OMG Document formal/03-09-01, September 2003.

[79] Object Management Group: *UML Profile for enterprise distributed Object Computing (EDOC) Version 1.0*, http://www.omg.org/technology/documents/formal/edoc.htm.

[80] Object Management Group: *XML Metadata Interchange (XMI) Version 2.0*, OMG Document: formal/03-05-02, Mai 2003.

[81] Omicini A.: *Societies and Infrastructures in the Analysis and Design of Agent-based Systems*, in: Proceeding of AOSE 2000, Springer, 2000.

[82] Oodes T., C. Müller-Schloer: *A New Approach to the Design of Safety-Critical Systems based on Virtual Prototyping, Assertions and Simulation*, VR@P Leira/Portugal, 1.-4. Oktober 2003, ISBN 972-99023-05, S. 321–328, 2003.

[83] Oodes T., C. Müller-Schloer: *Entwurf sicherheitskritischer, eingebetteter Systeme - Übersicht und neue Ansätze*, in: Embedded Intelligence 2002, Band 2, S. 523–532, Nürnberg, 2002.

[84] Oodes T., H. Krisp H. und C. Müller-Schloer: *On the combination of assertions and virtual prototyping for the design of safety-critical systems*, ARCS 2002/Trends in Network and Pervasive Computing, Karlsruhe. Berlin, Heidelberg, New York, Tokio, Springer, 2002.

[85] Oodes T., C. Müller-Schloer: *UML-basierter Systementwurf sicherheitskritischer, heterogener Systeme*, in: ASIM 2002, Rostock, 2002.

[86] Padgham L., M. Winikoff: *Prometheus: a methodology for developing intelligent agents*, in: Proceeding of AOSE 2002, S. 174–185, November 2002.

[87] Pasteels J.-M., J.-L. Deneubourg, S. Goss: *Self-organisation mechanisms in ant societies: Trail recruitment to newly discovered food sources*, in: Experimentica Supplmentum, 54. Jg., S. 155–175, 1987.

[88] Prigogine I., D. Kondepudi: *Modern thermodynamics: From heat engines to dissipative structures*, Chichester: John Wiley & Sons, 1998

[89] Roman G.C., H.C. Cunningham: *Mixed Programming Metaphors in a Shared Dataspace Model of Concurrency*, IEEE Transactions on Software Engineering, Band 16, Nr. 12, S. 1361–1373, 1990.

[90] Russel S., P. Norvig: *Künstliche Intelligenz*, Pearson Studium, August 2004.

[91] Smith R.G.: *he contract net protocol: High-level communication and control in distributed problem solving*, in: IEEE Transactions on Computers, Band C-29, Nr. 12, S. 1104–1113, 1980.

[92] Schreiber A.Th., B.J. Wielinga, J.M. Akkermans und W. Van de Velde: *CommonKADS: A comprehensive methdodology for KBS development*, Deliverable DM1.2a KADS-II/M1/RR/UvA/70/1.1, Universität Amsterdam, Niederlande, Energy Research Foundation ECN and Free University of Brussels, 1994.

[93] Stützle T., M. Dorigo: *ACO Algorithms for the Quadratic Assignment Problem*, in: Corne D., M. Dorigo und F. Glover (Hrsg.): New Ideas in Optimization, McGraw-Hill, 1999. (Auch Tech.Rep.IRIDIA/99-2, Université Libre de Bruxelles, Belgien)

[94] Ungerer T., J. Silc und B. Robic: *A Survey of Processors with Explicit Multithreading*, IEEE Computing Surveys, Band 35, Heft 1, S. 29-63, März 2003.

[95] Valckenaers P., H. Van Brussel, M. Kollingbaum und O. Bochmann: *Multi-agent coordination and control using stigmergy applied to manufacturing control*, In: Lecture Notes in Artivicial Intelligence, Nr. 2086, S. 317–334, Springer-Verlag, 2001.

[96] Valckenaers P., M. Kollingbaum, H. Van Brussel und O. Bochmann: *Short-term forecasting based on intentions in Multi-agent Control*, Proceedings of the 2001 IEEE Systems, Man and Cybernetics Conference, 2001.

[97] Valckenaers P., M. Kollingbaum, H. van Brussel, O. Bochmann und C. Zamfirescu: *The Design of Multi-Agent Coordination and Control Systems Using Stigmergy*, Proceedings of the IWES'01 Conference, 12.-13. März, Bled, Slowenien, 2001.

[98] Van Brussel H., J. Wyns, P. Valckenaers, L. Bongaerts und P. Peeters: *Reference architecture for holonic manufacturing systems: PROSA*, in: Computers In Industry 37, 1998.

[99] Voigt B.Fr.: *Der Handlungsreisende, wie er sein soll und was er zu thun hat, um Aufträge zu erhalten um eines glücklichen Erfolgs in seinen Geschäften gewiss zu sein*, Ilmenau, 1832.

[100] Waldo J.: *The Jini Architecture for Networkcentric Computing*, Communications of the ACM, S. 76–82, Juli 1999.

[101] Warmer J., A. Kleppe und W. Bast: *MDA explained. The Model Driven Architecture: Practice and promise*, Addison-Wesley, 2003.

[102] Warmer J., A. Kleppe: *The Object Constraint Language: Precise modeling with UML*, Addison-Wesley, 1999.

[103] Watkins C.J.C.H.: *Learning with delayed rewards*, Ph. D. dissertation, Psychology Department, University of Cambridge, England, 1989.

[104] Whitestein Technologies: *Agent Modeling Specification*, Version 0.9, 2004, http://www.whitestein.com/resources/aml/wt_AMLSpecification_v0.9.pdf.

[105] Wood M.F.: *Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems*, MS thesis, AFIT/GCS/ENG/00M-26.

[106] Wood M.F., S.A. DeLoach: *An Overview of the Multiagent Systems Engineering Methodology*, in: Proceedings of AOSE 2000, S. 207–222, 2000.

[107] Wooldridge M., P. Ciancarini: *Agent-Oriented Software Engineering: The State of the Art*, in: Ciancarini, P., M. Wooldridge (Hrsg.): Agent-Oriented Software Engineering, Lecture Notes in AI, Nr. 1957, Springer-Verlag, 2001.

[108] Wooldridge M., N. Jennings und D. Kinny: *The Gaia Methodology for Agent-Oriented Analysis and Design*, Journal of Autonomous Agents and Multi-Agent Systems, Band 3, Heft 3, S. 285–312, 2000.

Abbildungsverzeichnis

Tabellenverzeichnis

A. ASC-Algorithmus

1. */* Initialization phase */*
   For each pair (r, s) $\tau(r, s) := \tau_0$ End-for
   For k:=1 to m do
   Let r_{k1} be the starting city for an agent k
   $J_k(r_{k1}) := \{1, \ldots, n\} - r_{k1}$
   / $J_k(r_{k1})$ is the set of yet to be visited cities for agent k in city r_{k1}rk1 */*
   $r_k := r_{k1}$ */* r_k is the city where agent k is located */*
   End-for
2. */* This is the phase in which agents build their tours.*
   *The tour of agent k is stored in $Tour_k$. */*
   For $i := 1$ to n do
   if $i < n$
   Then
   For k:=1 to m do
   Choose the next city s_k according to formula 3.1 and formula 3.2
   If $i < n - 1$ Then $J_k(s_k) := J_k(r_k) - s_k$
   If $i = n - 1$ Then $J_k(s_k) := J_k(r_k) - s_k + r_{k1}$
   $Tour_k(i) := (r_k, s_k)$
   End-for
   Else
   For k:=1 to m do */* In this cycle all the agents go back to the initial city r_{k1} */*
   $s_k := r_{k1}$
   $Tour_k(i) := (r_k, s_k)$
   End-for
   / In this phase local updating is computed and τ-values are updated using formula 3.3 */*
   For k:=1 to m do
   $\tau(r, s) := (1 - \rho) \cdot \tau(r, s) + \rho \cdot \tau_0$
   $r_k := s_k$
   End-for
   End-for
3. */* In this phase delayed reinforcement is computed and τ-values are updated */*
   For k:=1 to m do
   Compute L_k */* L_k is the length of the tour done by agent k */*
   End-for
   Compute $L_{best-iter}$
   */*Update edges belonging to $L_{best-iter}$ using formula 3.4 */*
   For each edge (r, s)
   $\tau(r, s) := (1 - \alpha) \cdot \tau(r, s) + \alpha \cdot (L_{best-iter})^{-1}$
   End-for
4. If $(End\_condition = True)$
   Then
   Print shortest of L_k
   Else
   goto phase 2

Abbildung A.1: Der ACS-Algorithmus von Dorigo und Gambardella [46]

B. Diagramme und Tabellen

Solution Topology Knowlegde	Dies umfasst Wissen über die Komponenten und deren Konstruktion bzw. Konfiguration für eine Lösung oder ein Business System. Installations- und Konfigurationswissen sind in einem allgemeinen Installationseinheitsformat gekapselt, um Komplexität zu reduzieren. Die Planungsfunktion eines Autonomic Managers kann dieses Wissen für die Installations- und Konfigurationsplanung verwenden.
Policy Knowledge	Eine Policy umfasst eine Menge von Verhaltensbedingungen oder Präferenzen, welche die Entscheidungen einer Autonomic Managers beeinflussen. Policy Knowledge wird zu Rate gezogen, um zu bestimmen, ob Änderungen am System notwendig sind oder nicht. Ein Autonomic Computing System benötigt ein einheitliches Verfahren für die Definition der Policies, welche die Entscheidungsfindung von Autonomic Managers beeinflussen. Durch eine standardisierte Definition können Policies von mehreren Autonomic Managers genutzt werden, so dass das gesamte System von einer gemeinsamen Menge von Policies verwaltet werden kann.
Problem Determination Knowledge	Problem Determination Knowledge schließt beobachtete Daten, Symptome und Entscheidungsbäume mit ein. Der Problem Determination Prozess kann ebenfalls Wissen erstellen. Durch die Beobachtung, wie das System auf korrigierende Aktionen reagiert, kann das gelernte Wissen im Autonomic Manager gespeichert werden. Ein Autonomic Computing System benötigt auch hier ein einheitliches Verfahren für die Repräsentation von Problem Determination Wissen, wie beobachtete Daten, Symptome und Entscheidungsbäume.

Tabelle B.1: Wissensarten im Autonomic Manager [56]

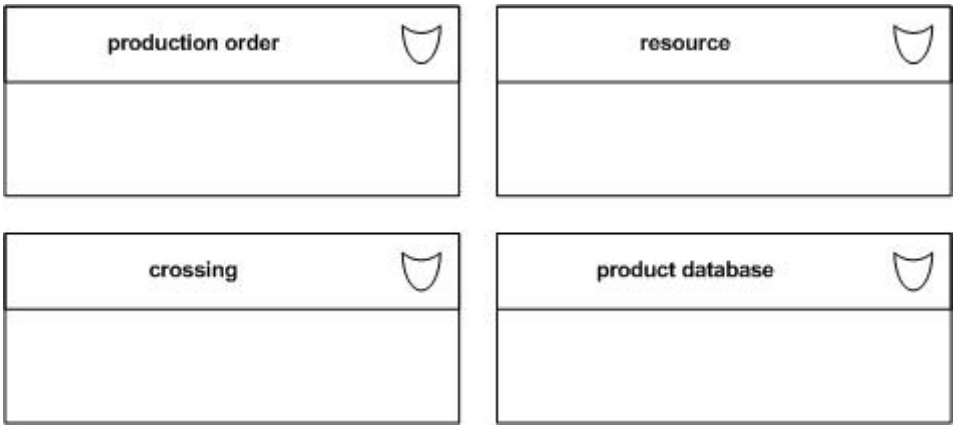

Abbildung B.1: Role Model nach der Identifikation der Rollen

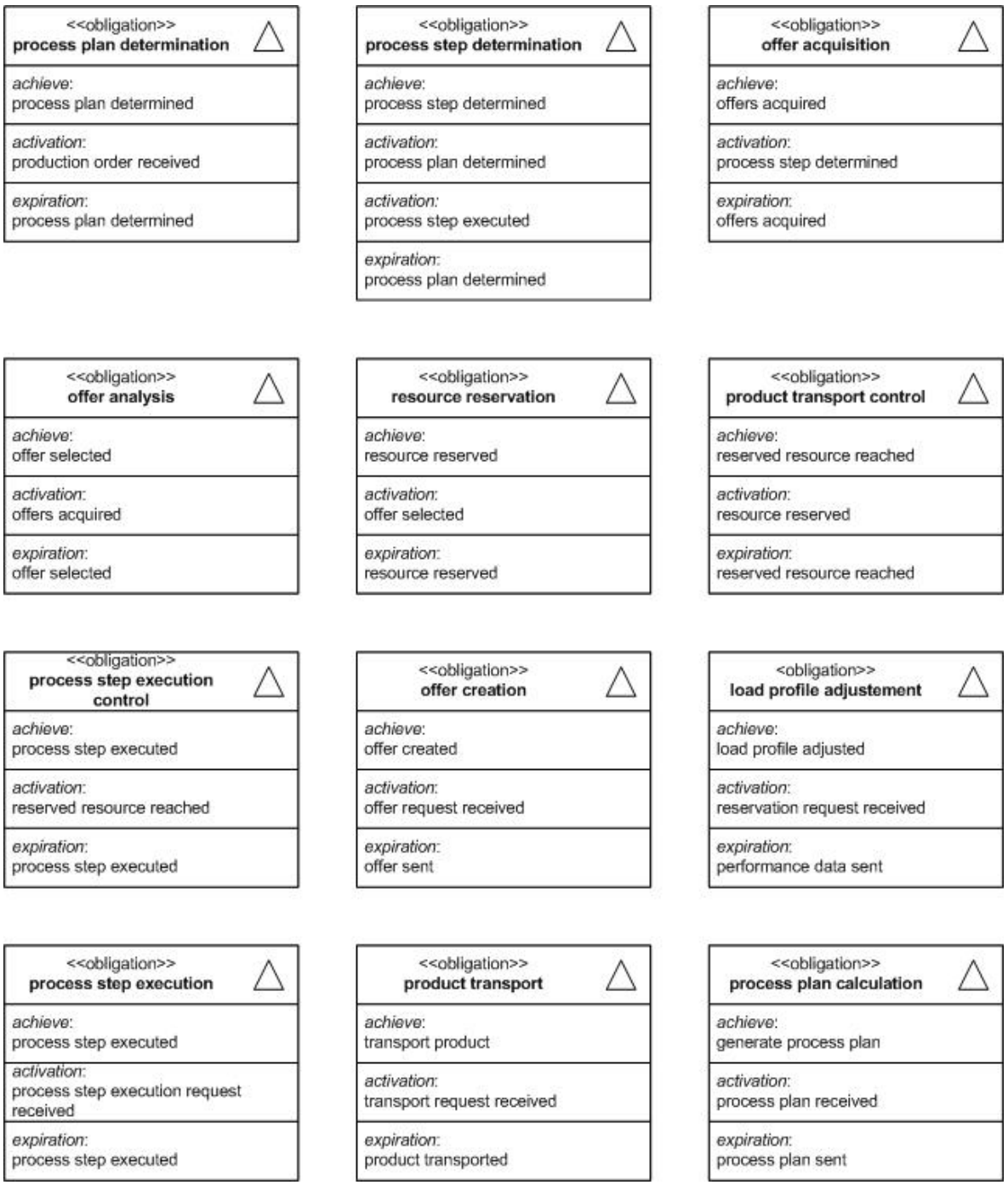

Abbildung B.2: Norm Model

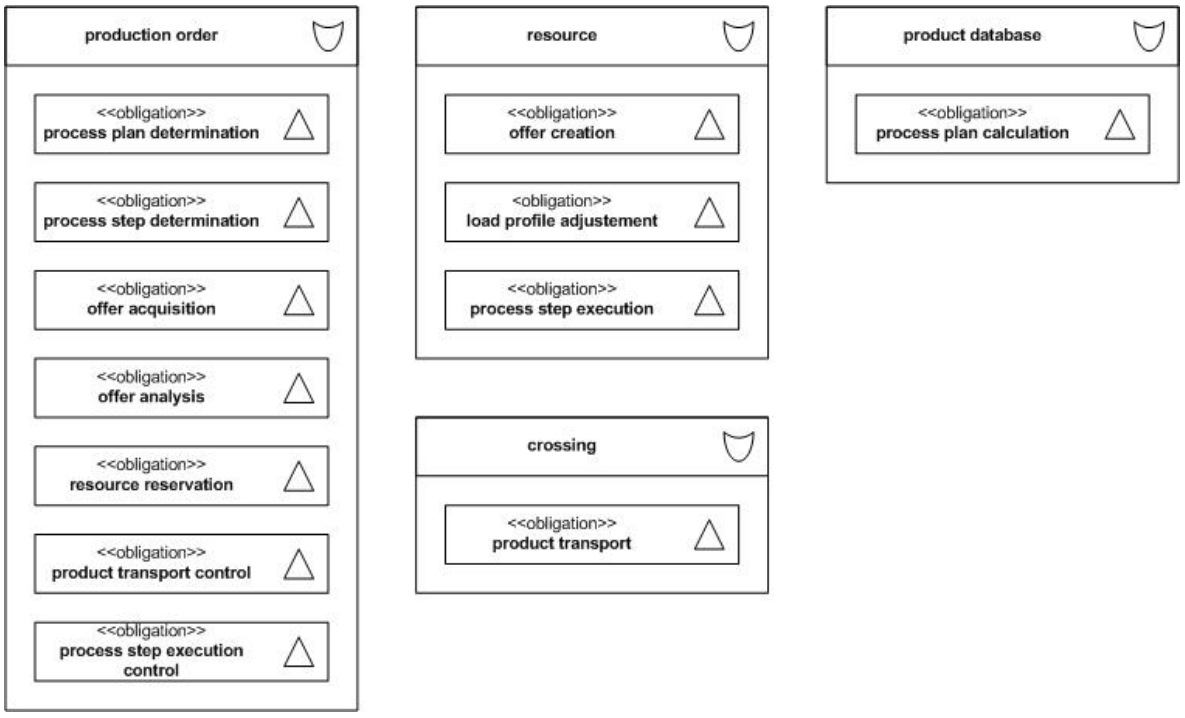

Abbildung B.3: Role Model nach der Spezifikation der Normen

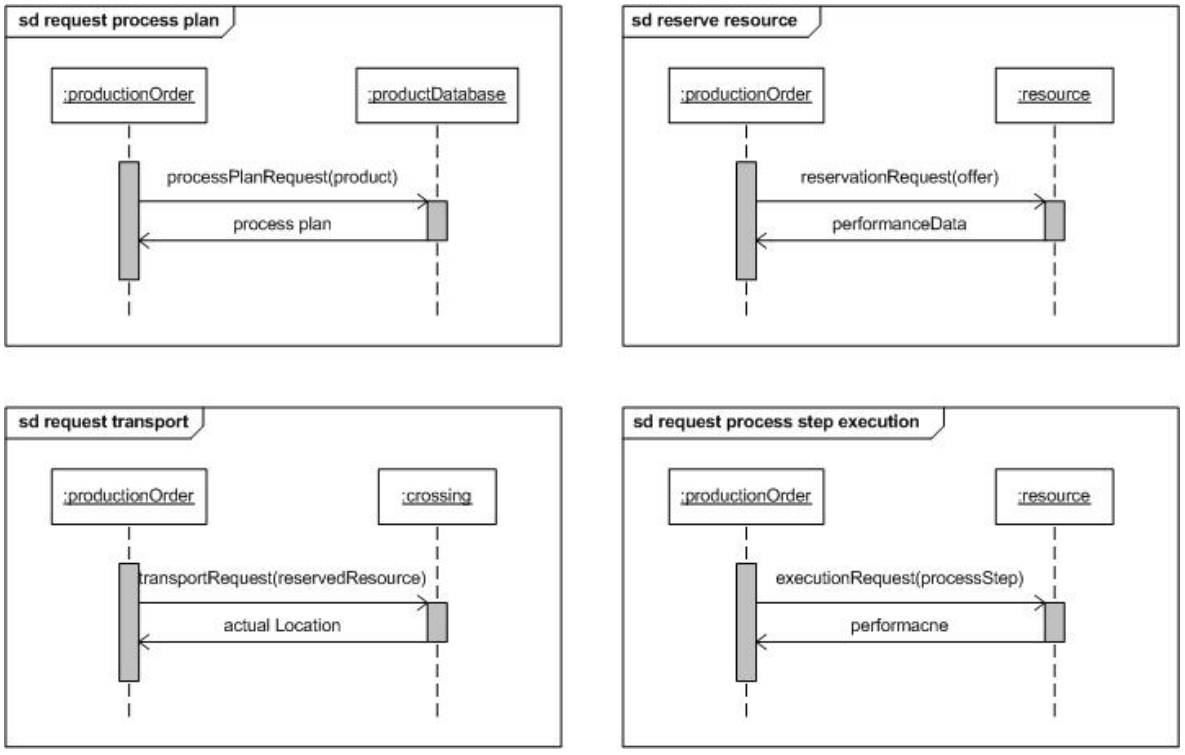

Abbildung B.4: Interaktionspattern *request process plan*, *reserve resource*, *request transport* und *request process step execution*

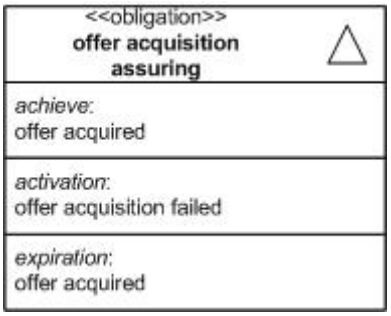

Abbildung B.5: Autonomic Manager Norm für die Selbst-Heilung des Systems

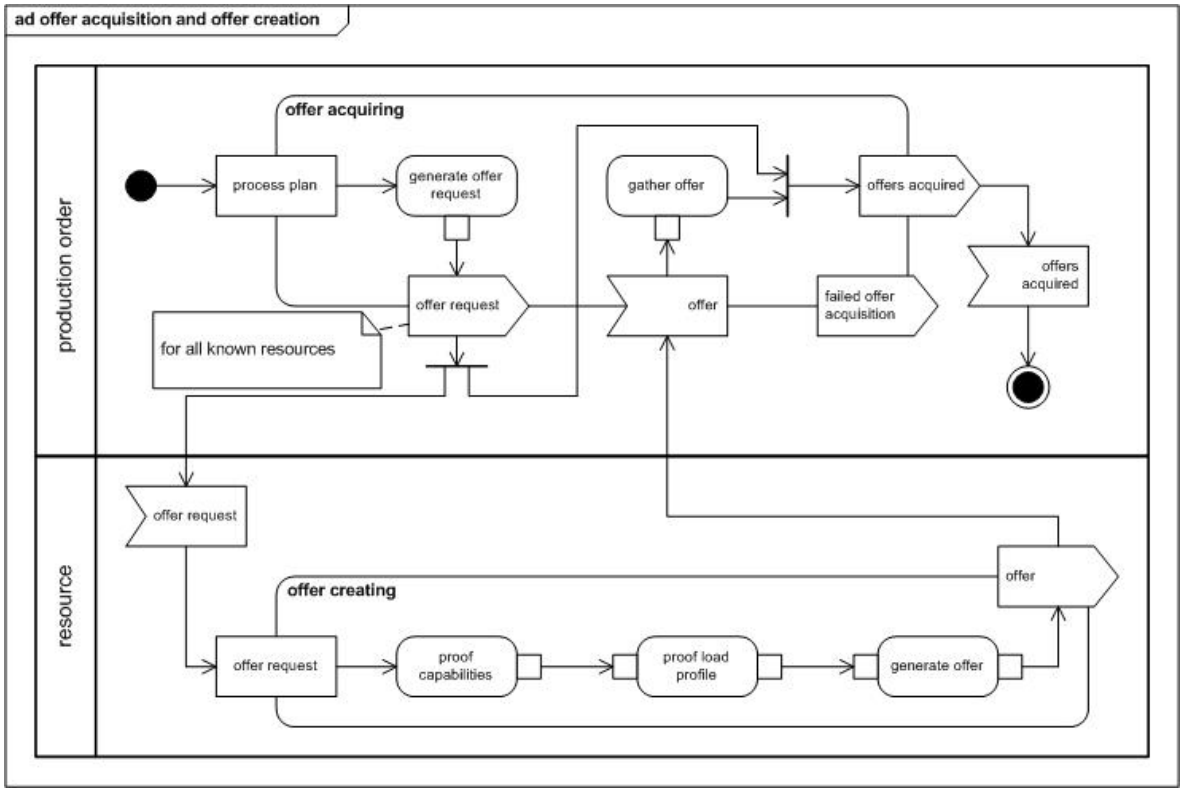

Abbildung B.6: Modifizierter Plan *offer acquiring* im Plan Model

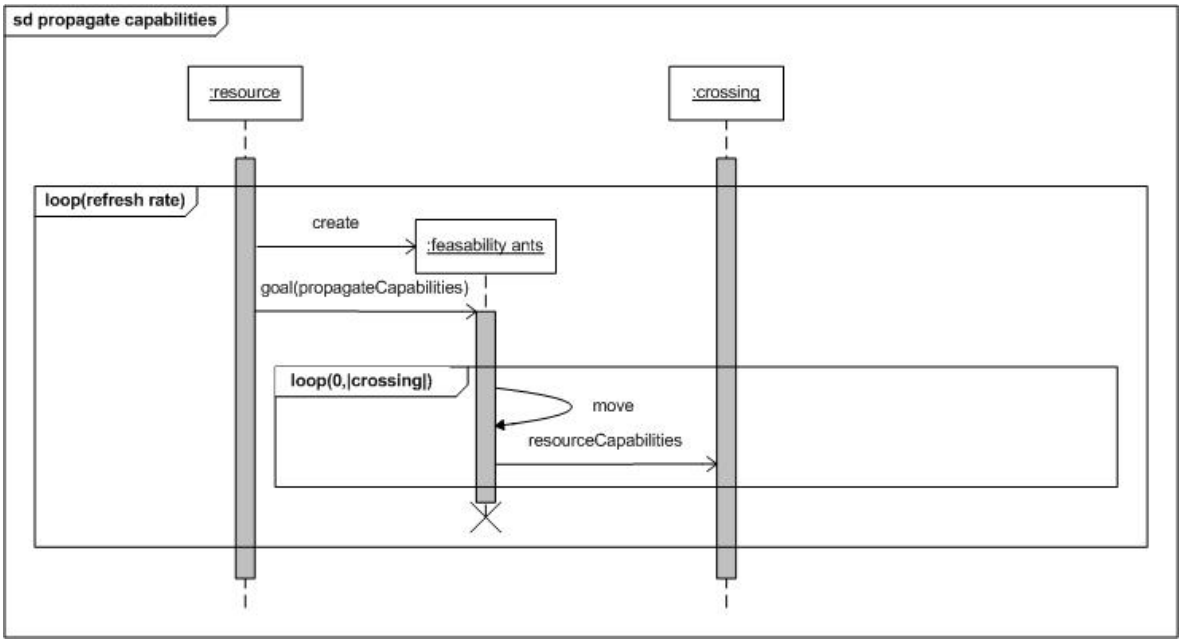

Abbildung B.7: Interaktionsprotokoll *propagate capabilities*

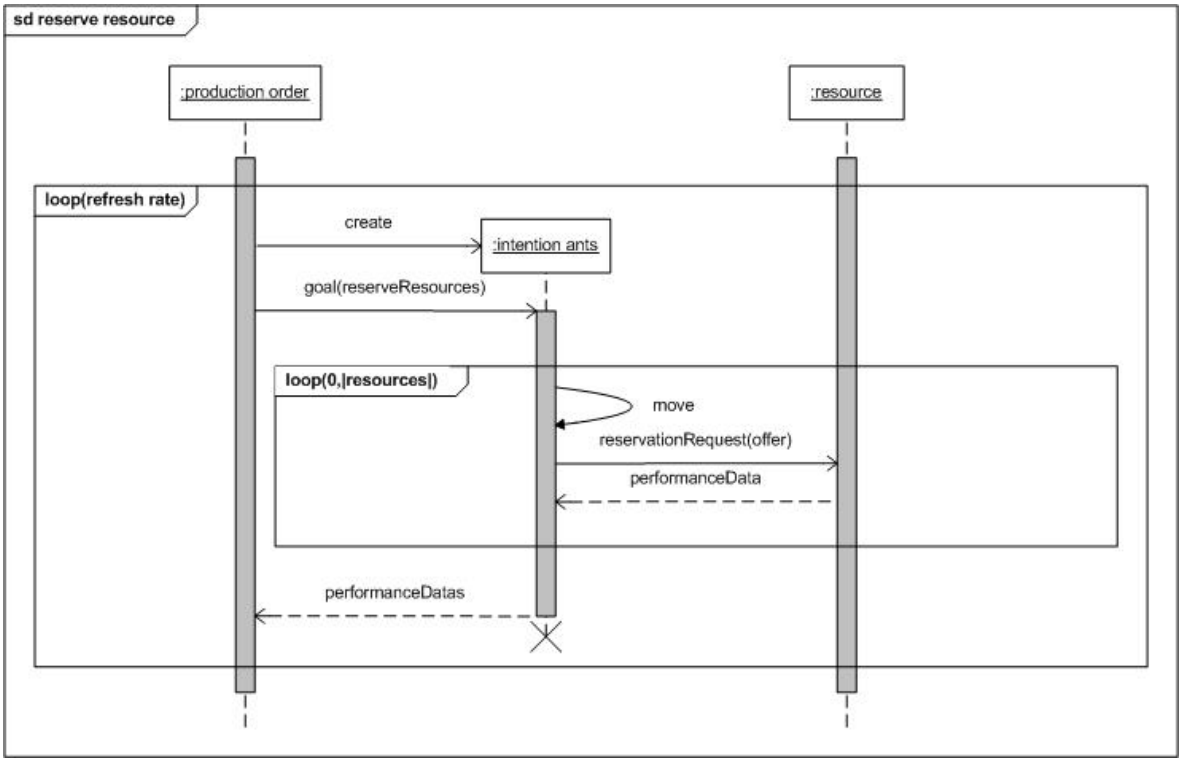

Abbildung B.8: Interaktionsprotokoll *reserve resource*

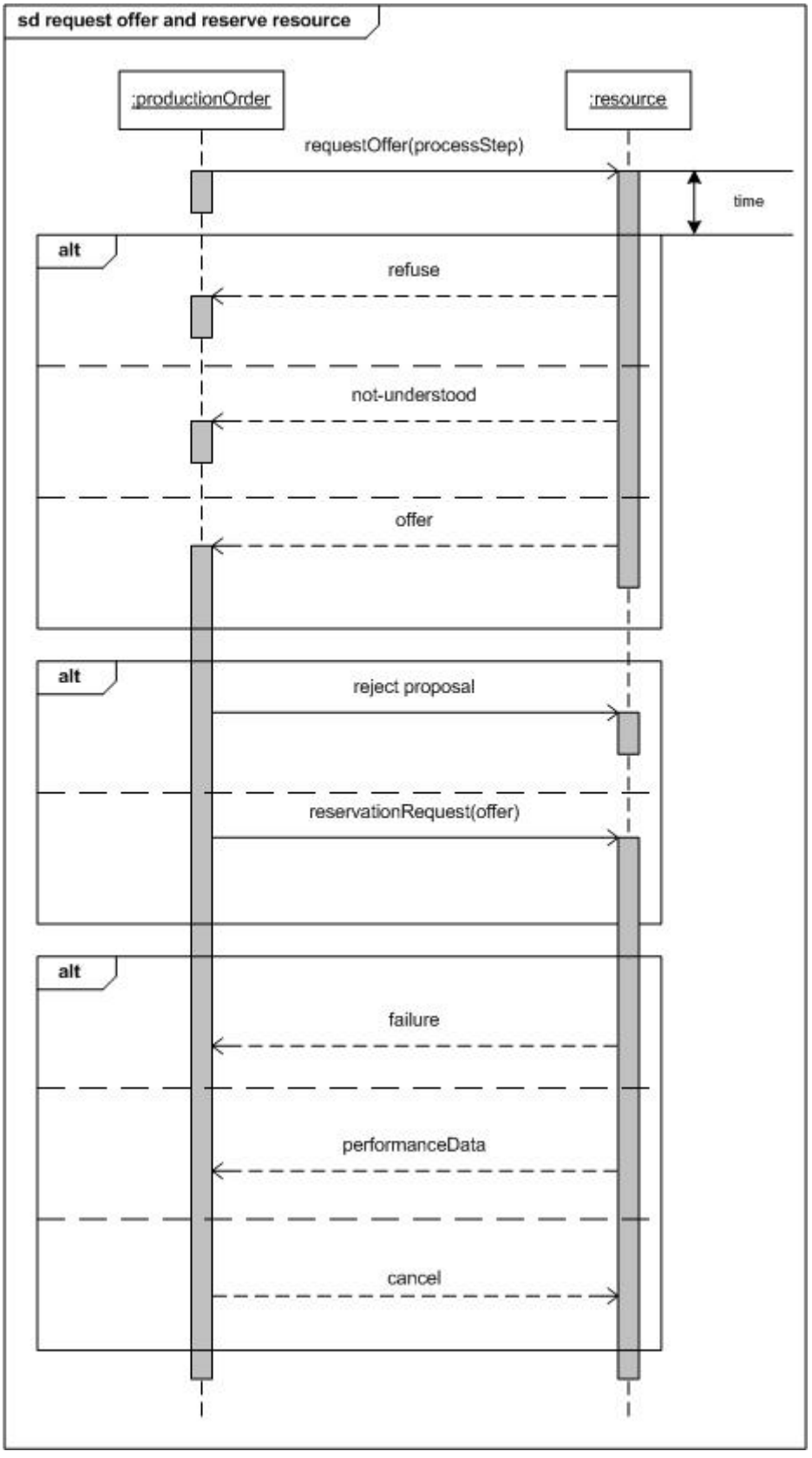

Abbildung B.9: Interaction Protocol Model

C. Modelltransformationsregeln - Teil 1

Entwicklungsmodell	**Abkürzung**
Business Context Model	BCM
Business Process Model	BPM
Environment Model	EVM
Use Case Model	UCM
Ontology Model	ONM
Role Model	MRM
Norm Model	MNM
Plan Model	MPM
Interaction Model	MIM
Service Model	MSM
Autonomic Manager Role Model	ARM
Autonomic Manager Norm Model	ANM
Autonomic Manager Analyze Model	AAM
Autonomic Manager Plan Model	APM
Autonomic Manager Interaction Model	AIM
Autonomic Manager Service Model	ASM
Interaction Protocol Model	IPM
Autonomic Element Model	AEM
Autonomic Element Instance Model	EIM

Tabelle C.1: Entwicklungsmodelle und ihre Abkürzungen

Die folgenden Modelltransformationsregeln werden für Transformationen zwischen Modellen auf der CIM-Ebene, der PIM-Ebene und zwischen den beiden Ebenen eingesetzt bis Aktivität 10. Die Notation der Regeln erfolgt im *if-then*-Stil, d. h. wenn die Bedingung bei *if* wahr ist, wird die bei *then* angegebene Aktion ausgeführt. Die Erklärung der einzelnen Regeln befindet sich in 6.4.

TR_BCM_BPM_01

if	*Prozess <p> im BCM soll durch ein OCS unterstützt werden*
then	*Trage Partition <name>, welche <p> ausführt, aus dem BCM in das BPM als Partition <name> ein*

Abbildung C.1: Transformationsregel *TR_BCM_BPM_01*

TR_BPM_EVM_01

if	*true*
then	*Trage jede Unterpartition <p> aus dem BPM als Klasse <p> in das EVM ein*

Abbildung C.2: Transformationsregel *TR_BPM_EVM_01*

TR_EVM_UCM_01

if	*Klasse <k> im EVM existiert als Partition im BPM*
and	*<k> enthält als Partition einen Startpunkt im BPM*
then	*Trage <k> als primären Akteur in das Use Case Diagramm des UCM ein*
and	*Trage <k> als Objekt mit Lebenslinie in ein Sequenzdiagramm des UCM ein*

Abbildung C.3: Transformationsregel *TR_EVM_UCM_01*

TR_EVM_UCM_02

if	*Klasse <k> im EVM existiert als Partition im BPM*
and	*<k> enthält als Partition keinen Startpunkt im BPM*
then	*Trage <k> als sekundären Akteur in das Use Case Diagramm des UCM ein*
and	*Trage <k> als Objekt mit Lebenslinie in ein Sequenzdiagramm des UCM ein*

Abbildung C.4: Transformationsregel *TR_EVM_UCM_02*

TR_UCM_UCM_01

if	*true*
then	*Trage System <s> aus dem Use Case Diagramm als Objekt <s> mit Lebenslinie in das Sequenzdiagramm ein UCM ein*

Abbildung C.5: Transformationsregel *TR_UCM_UCM_01*

TR_UCM_UCM_02

if	*true*
then	*Trage Anwendungsfall <bezeichung> aus dem Use Case Diagramm als aufrufende Nachricht zwischen einem primären Akteur und dem Sytem <s> in ein Sequenzdiagramm ein*

Abbildung C.6: Transformationsregel *TR_UCM_UCM_02*

TR_BPM_UCM_01

if	*true*
then	*Trage Aktion <name> aus dem BPM als Referenz <name> in das Sequenzdiagramm des UCM ein*

Abbildung C.7: Transformationsregel *TR_BPM_UCM_01*

TR_XXX_ONM_01
if *true*
then *Trage jeden Begriff aus dem Modell XXX in das ONM als Klasse ein*

Abbildung C.8: Transformationsregel *TR_XXX_ONM_01*

TR_UCM_MRM_01
if *Objekt <name> im Sequenzdiagramm des UCM ist nicht das System selber*
then *Trage Rolle <name> in das MRM ein*

Abbildung C.9: Transformationsregel *TR_UCM_MRM_01*

TR_UCM_MRM_02
if *Nachricht <name> ist initiierende Nachricht im Sequenzdiagramm des UCM*
then *Trage Rolle <name> in das MRM ein*

Abbildung C.10: Transformationsregel *TR_UCM_MRM_02*

TR_UCM_MNM_01
if *Nachricht <name> ist ein interner Aufruf eines Objekts <o>*
then *Trage Norm <name> als Obligation in das MNM ein*
and *Trage Referenz auf <name> in das MRM bei Rolle <o> ein*

Abbildung C.11: Transformationsregel *TR_UCM_MNM_01*

TR_UCM_MNM_02
if *Nachricht <name> ist ein externer Aufruf eines Objekts <o>*
then *Trage eine Norm als Obligation in das MNM ein*
and *Trage eine Referenz auf die Norm in das MRM bei Rolle <o> ein*

Abbildung C.12: Transformationsregel *TR_UCM_MNM_02*

TR_MNM_MPM_01
if *true*
then *Trage die Rolle <r> einer Norm <n> als Partition <r> in das MPM ein*
and *Trage das Deaktivierungsereignis <d> der Norm <n> als Empfangsereignis <d> in die Partition <r> im MPM ein*
and *Trage das Deaktivierungsereignis <d> der Norm <n> als Sendeereignis <d> in die Partition <r> des Plans <p> im MPM ein*
and *Trage eine Referenz auf den Plan <p> in das MRM bei Rolle <r> ein*

Abbildung C.13: Transformationsregel *TR_MNM_MPM_01*

TR_UCM_MPM_01

if *Nachricht <name> ist ein externer Aufruf eines Objekts <o>*
and *<name> führte zur Norm <n> im MNM*
and *<p> ist ein Plan für die Erfüllung von <n> im MPM*
then *Trage Parameter <x> von <name> als Sendeereignis bei <p> ein*
and *Trage Parameter <x> von <name> als Empfangsereignis bei derjenigen Partition im MPM ein, welche zu dem Empfangsobjekt <q> der Nachricht <name> korrespondiert*
and *Trage einen Pfeil zwischen Sende- und Empfangsereignis ein*

Abbildung C.14: Transformationsregel *TR_UCM_MPM_01*

TR_MPM_MIM_01

if *Ereignis <e> ist Sendeereignis im Plan einer Partition <m> im MPM*
and *Ereignis <f> ist Empfangsereignis einer Partition <n> im MPM*
and *<e> und <f> sind miteinander über einen Pfeil verbunden*
then *Trage <m> als Objekt mit Lebenslinie in ein Interaktionspattern <i> im MIM ein*
and *Trage <n> als Objekt mit Lebenslinie in <i> im MIM ein*
and *Trage <e> als Nachricht zwischen <m> und <n> im MIM ein*
and *Trage eine Referenz auf <i> bei der Rolle <m> im MRM ein*

Abbildung C.15: Transformationsregel *TR_MPM_MIM_01*

TR_MPM_MSM_01

if *Aktion <a> liegt innerhalb eines Plans einer Partition <m> im MPM*
then *Trage <a> als privaten Service <s> im MSM ein*
and *Trage den Eingangspin <a1> als Eingangsparameter von <s> ein*
and *Trage den Ausgangspin <a2> als Ausgabeparameter von <s> ein*
and *Trage eine Referenz auf <s> bei der Rolle <m> im MRM ein*

Abbildung C.16: Transformationsregel *TR_MPM_MSM_01*

TR_MIM_MSM_01

if *Rolle <r> ist Partizipant in einem Interaktionspattern <i> im MIM*
then *Trage den Namen der erhaltenen Nachricht in <i> als öffentlichen Service <s> im MSM ein*
and *Trage den Parameter <n1> von <name> als Eingangsparameter von <s> ein*
and *Trage den Rücknachricht <n2> in <i> als Ausgabeparameter von <s> ein*
and *Trage eine Referenz auf <s> bei der Rolle <r> im MRM ein*

Abbildung C.17: Transformationsregel *TR_MIM_MSM_01*

D. Modelltransformationsregeln - Teil 2

Die in diesem Teil aufgeführten Transformationsregeln befassen sich mit Regeln ab Aktivität 11 des Entwicklungsprozesses.

TR_ANM_ARM_01

if		*Norm <n> im ANM wird noch nicht referenziert*
then		*Erstelle eine Rolle <r> im ARM*
	and	*Trage eine Referenz auf <n> bei <r> im ARM ein*

Abbildung D.1: Transformationsregel *TR_ANM_ARM_01*

TR_ANM_AAM_01

if		*Ereignis <e> ist Aktivierungsereignis einer Norm <n> der AM-Rolle <a>*
then		*Trage Partition <a> in das AAM ein*
	and	*Trage <e> als Empfangsereignis bei Partition <a> im AAM ein*
	and	*Trage <e> als Sendeereignis bei in der Analyzeregel der Partition <a> im AAM ein*

Abbildung D.2: Transformationsregel *TR_ANM_AAM_01*

TR_ANM_AAM_02

if		*Ereignis <f> ist Deaktivierungsereignis einer Norm <n> der AM-Rolle <a>*
then		*Trage Partition <a> in das AAM ein*
	and	*Trage <e> als Empfangsereignis bei Partition <a> im AAM ein*
	and	*Trage <e> als Sendeereignis bei in der Analyzeregel der Partition <a> im AAM ein*

Abbildung D.3: Transformationsregel *TR_ANM_AAM_02*

TR_ANM_APM_01

if *true*
then *Trage die Rolle <r> einer Norm <n> im ANM als Partition <r-am> in das APM ein*
and *Trage das Deaktivierungsereignis <d> der Norm <n> in die Partition <r> als Empfangsereignis <d> im APM ein*
and *Trage das Deaktivierungsereignis <d> der Norm <n> in die Partition <r> als Sendeereignis <d> des Plans <p> im APM ein*
and *Trage eine Referenz auf den Plan <p> in das ARM bei Rolle <r> ein*

Abbildung D.4: Transformationsregel *TR_ANM_APM_01*

TR_APM_AIM_01

if *Ereignis <e> ist Sendeereignis im Plan einer AM-Partition <m> im APM*
and *Ereignis <f> ist Empfangsereignis einer Partition <n> im APM*
and *<e> und <f> sind miteinander über einen Pfeil verbunden*
then *Trage <m> als Objekt mit Lebenslinie in ein Interaktionspattern <i> im AIM ein*
and *Trage <n> als Objekt mit Lebenslinie in <i> im AIM ein*
and *Trage <e> als Nachricht zwischen <m> und <n> im AIM ein*
and *Trage eine Referenz auf <i> bei der Rolle <m> im ARM ein*

Abbildung D.5: Transformationsregel *TR_APM_AIM_01*

TR_APM_ASM_01

if *Aktion <a> liegt innerhalb eines Plans einer AM-Partition <p> im APM*
then *Trage <a> als privaten Service <s> im ASM ein*
and *Trage den Eingangspin <a1> als Eingangsparameter von <s> ein*
and *Trage den Ausgangspin <a2> als Ausgabeparameter von <s> ein*
and *Trage eine Referenz auf <s> bei der Rolle <p> im ARM ein*

Abbildung D.6: Transformationsregel *TR_APM_ASM_01*

TR_APM_MSM_01

if *Aktion <a> liegt innerhalb eines Plans einer Partition <p> im APM*
then *Trage <a> als privaten Service <s> im MSM ein*
and *Trage den Eingangspin <a1> als Eingangsparameter von <s> ein*
and *Trage den Ausgangspin <a2> als Ausgabeparameter von <s> ein*
and *Trage eine Referenz auf <s> bei der Rolle <p> im MRM ein*

Abbildung D.7: Transformationsregel *TR_APM_MSM_01*

TR_AIM_ASM_01

if *AM-Rolle <r> ist Partizipant in einem Interaktionspattern <i> im AIM*
then *Trage den Namen der erhaltenen Nachricht in <i> als öffentlichen Service <s> im ASM ein*
and *Trage den Parameter <n1> von <name> als Eingangsparameter von <s> ein*
and *Trage den Rücknachricht <n2> in <i> als Ausgabeparameter von <s> ein*
and *Trage eine Referenz auf <s> bei der AM-Rolle <r> im ARM ein*

Abbildung D.8: Transformationsregel *TR_AIM_ASM_01*

TR_AIM_MSM_01

if		*MR-Rolle <r> ist Partizipant in einem Interaktionspattern <i> im AIM*
then		*Trage den Namen der erhaltenen Nachricht in <i>*
		als geschützten Service <s> im MSM ein
	and	*Trage den Parameter <n1> von <name> als Eingangsparameter von <s> ein*
	and	*Trage den Rücknachricht <n2> in <i> als Ausgabeparameter von <s> ein*
	and	*Trage eine Referenz auf <s> bei der MR-Rolle <r> im MRM ein*

Abbildung D.9: Transformationsregel *TR_AIM_MSM_01*