# Interaktion und Koordination in Multiagentensystemen

Von der Fakultät Informatik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

**Klaus Matthias Muscholl**

aus München

| | |
|---|---|
| Hauptberichter: | Prof. Dr. rer. nat. habil. P. Levi |
| Mitberichter: | Prof. Dr.-Ing. E. Lehmann |
| Tag der mündlichen Prüfung: | 17. Juli 2000 |

Institut für Parallele und Verteilte Höchstleistungsrechner
der Universität Stuttgart

2001

# Danksagung

Die vorliegende Arbeit entstand in den Jahren 1992 bis 1999 während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Parallele und Verteilte Höchstleistungsrechner der Universität Stuttgart.

Mein besonderer Dank gilt Herrn Prof. Dr. Paul Levi, der mit seinem Enthusiasmus und seinen Anregungen wesentlich zur thematischen Ausrichtung meiner Forschungsarbeiten beigetragen hat. Er interessierte sich stets für die Zusammenhänge und systemdynamischen Interpretationen, welche aus der Arbeit abgeleitet werden konnten und Gegenstand zahlreicher anregender Diskussionen waren.

Herrn Prof. Dr.-Ing. Egbert Lehmann möchte ich sehr herzlich für sein Interesse an meiner Arbeit und für die Übernahme des Mitberichts danken.

Mit Michael Becht habe ich nicht nur unser Büro geteilt, er hat auch meine wissenschaftliche Arbeit zuerst als Student und dann als Mitarbeiter aktiv begleitet. Die Zusammenarbeit mit ihm und sein kritischer Geist waren im großen Maße motivierende Faktoren. Jürgen Klarmann verdanke ich die Einblicke in den Bereich des *computer supported cooperative work* (CSCW).

Den Mitgliedern der Abteilung Bildverstehen möchte ich für den Gruppengeist und die gegenseitige Unterstützung danken, die sich nicht zuletzt aus der Zusammenarbeit im Comros-Projekt entwickelt haben.

Anca und Nora verdanke ich mein privates Glück.

Paris, im Juni 2001 Matthias Muscholl

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Definitionsverzeichnis

# Abstract

Ensuring coherence in Multi-Agent Systems is still an open question due to the problem of distribution and independent development in open systems. This thesis proposes an approach at design time, which uses so-called *interaction protocols* to design coherence. *Interaction protocols* are described by a formal *interaction model*.

The basic idea is that by participating in an interaction, agents transfer parts of their autonomy to the interaction and to its mechanisms of decision-making. They subordinate themselves to the interaction, which can be specified by a protocol. The protocol is responsible for the coordination and supervision of the transfered autonomy. *Interaction protocols* are authorized to give directives.

To be able to participate in an *interaction protocol,* an agent has to support an interface, which permits the protocol to access the competences it receives from the agent and to carry out the decisions that have been taken. For each application domain services have to be defined and have to be supported by the related agents.

An *interaction protocol* defines the process scheme of an interaction. The scheme abstracts from agents by roles. The scheme is structured by phases and defines the interaction between roles. Roles are the smallest active unit of the *interaction model* and are valid only for one phase. Agents participating in an *interaction protocol* are supervised by roles that are assigned to them in each phase.

This work is part of the architectural concepts for robot systems in the Comros project. Agents are interconnected by *dataflow networks*. They coordinate themselves by *decision networks* that are responsible to keep the consistency of their world models. Finally the *decision networks* are modelled by *interaction protocols*.

# Zusammenfassung

Das Zusichern von kohärentem Verhalten in Multiagentensystemen ist durch die inhärente Verteiltheit des Systems, als auch durch den unabhängigen Entwurf der Agenten bei offenen Systemen, ein weithin ungelöstes Problem. In der vorliegenden Dissertation wird ein entwurfstechnischer Ansatz vorgestellt, welcher mit Hilfe von *Interaktionsverfahren* Kohärenz sicherstellt. Interaktionsverfahren werden dabei durch das *Interaktionsmodell* beschrieben.

Die Grundidee besteht darin, daß Agenten durch die Teilnahme an einer Interaktion einen Teil ihrer Autonomie an das die Interaktion beschreibende Verfahren und seine Entscheidungsmechanismen abtreten und sich ihm unterordnen. Dies wird dadurch erzielt, daß ein Interaktionsverfahren die Koordination der an ihn übertragenen Kompetenzen übernimmt. Ein Interaktionsverfahren ist somit gegenüber den teilnehmenden Agenten weisungsbefugt.

Um an Interaktionsverfahren teilnehmen zu können, muß ein Agent eine Schnittstelle unterstützen, welche es dem Interaktionsverfahren ermöglicht, auf die an ihn übertragen Kompetenz zuzugreifen und die kollektiv getroffenen Entscheidungen im einzelnen durchzusetzen. Hierzu sind unabhängig von Interaktionsverfahren für einen Anwendungsbereich Dienstklassen definiert, welche Schnittstellen zu Fähigkeiten eines Agenten bilden.

Ein Interaktionsverfahren definiert das Ablaufschema einer Interaktion. Das Ablaufschema abstrahiert von Agenten in Form von Rollen. Das Schema ist in einzelne Phasen strukturiert und definiert, wie die Rollen untereinander interagieren. Rollen sind die kleinsten aktiven Einheiten des Interaktionsmodells und nur innerhalb einer Phase gültig. Agenten, welche an einem Interaktionsverfahren teilnehmen, werden durch Rollen gesteuert, die ihnen in den jeweiligen Phasen zugewiesen werden.

Die vorliegende Arbeit ist Bestandteil des Architekturkonzepts von Robotersystemen im Comros-Projekt. Agenten werden durch *Datenflußnetzwerke* verbunden. Sie koordinieren sich über sogenannte *Entscheidungsnetzwerke*, welche ihre Weltmodelle konsistent halten. Die Entscheidungsnetzwerke werden als Interaktionsverfahren definiert.

# Kapitel 1

# Einführung

In den letzten Jahren hat sich das Interesse an der Agententechnologie sowohl in der Forschung als auch im industriellen Einsatz enorm gesteigert. Man hat die Hoffnung, daß Agenten die geeignete Abstraktion von Prozessen darstellen, welche über weltweite Netzwerke interagieren und kooperativ Problemlösungen erarbeiten.

Das Gebiet der Agententechnologie läßt sich aus heutiger Sicht in vier Bereiche gliedern, welche sich unterschiedlich scharf gegeneinander abgrenzen lassen:

1. Der Bereich der Verteilten Künstlichen Intelligenz (VKI) entwickelte sich aus der klassischen Künstlichen Intelligenz heraus. Anfang der 80iger Jahre wurde begonnen, einzelne Planer (Agenten) zu koppeln ([Mül 93] S. 105). Ende der 80iger Jahre waren die wesentlichen Fragestellungen der VKI bekannt [BoGa 88] und die ersten spezialisierten Konferenzen zur Modellierung von Multiagentensystemen wurden ausgerichtet [DeMü 89, Les 95, WoJe 94]. In der Folge wuchsen das Interesse und die Anzahl der Arbeiten in dem Bereich. Heute kann man von einer Festigung des Wissens über Multiagentensysteme ausgehen [Wei 99]. Im Bereich der VKI hat sich der Begriff der intelligenten Agenten durchgesetzt, welche sich durch ihre kognitiven Fähigkeiten auszeichnen.

2. Im Bereich der Robotik [Lev 96] ist die Forschung über Softwarearchitekturen weiterhin aktuell [Par 99], welche den Anforderungen an Realzeitfähigkeit einerseits und den Anforderungen an eine geeignete Infrastruktur für komplexe Algorithmen andererseits erfüllen. Aufgabe des Gesamtsystems ist es die hohe Dynamik der unstrukturierten Umgebung zu beherrschen, in welcher sich autonome Roboter einzeln oder in Teams zurechtfinden müssen. Der Bereich der Robotik prägte hierfür den Begriff der Roboteragenten.

3. Aus dem Bereich der verteilten Systeme mit ihrem Schwerpunkt der sicheren, zuverlässigen und leistungsstarken Systemarchitekturen sind Konzepte zum Zwecke der Lastbalancierung entwickelt worden, wie Prozesse während ihres Ablauf auf andere Rechner umverlagern werden können. Verallgemeinert versucht man Systeme zu erforschen, in denen Prozesse oder sogenannte mobile Agenten [Roth 97, Roth 98] in

der Lage sind, durch ein Netzwerk zu wandern und Dienstleistungen einzelner Rechnerknoten in Anspruch zu nehmen. Die Agenten sind meist durch den Auftrag eines Benutzers entstanden, stehen jedoch nicht unter seiner Kontrolle, sondern versuchen in Erfüllung einer Aufgabe das Gesamtsystem bestmöglich zu nutzen.

Die Architektur mobiler Agenten konzentriert sich auf die Fähigkeiten, zwischen Rechnersystemen zu migrieren, und im Gegensatz zu der Architektur von intelligenten Agenten nicht auf ihre kognitiven Fähigkeiten. Hier wird Fragestellungen hinsichtlich der zugrundeliegenden Systemmechanismen nachgegangen (Dienstschnittstellen zur Migration, Nachrichtenübermittlung, entfernter Prozeduraufruf). Es werden Systemplattformen bereitgestellt, welche die Sicherheit des Systems gegenüber fremden Agenten, die Sicherheit von Agenten gegenüber fremden Systemen und die Sicherheit zwischen einander fremden Agenten gewährleistet. Grundsätzlich finden Interaktionen über die Systemplattform als Mediator statt, wobei das Forschungsinteresse im Bereich der Basisinfrastruktur und nicht im Anwendungsbereich liegt.

4. In Abgrenzung zu dem Bereich der Robotik mit seinen meist mit intelligenten Verfahren ausgestatteten Roboteragenten hat sich der Bereich der Softwareagenten gebildet, in dem die Forschung die Technologie ohne den Anwendungsbezug zur Robotik vorantreibt. Aus dem Bereich der Computeranimation heraus wird der Begriff der Softwareagenten gerne für Agenten verwendet, welche ein visuelles Erscheinungsbild haben und sich z. B. durch virtuelle Welten bewegen. [Sen 99] argumentiert, daß ein Agent durch die Steuerung seines Erscheinungsbildes das Verständnis eines Benutzers fördern kann, weshalb er sich auf eine bestimmte Art und Weise verhält. Allgemein wird jedoch der Begriff der Softwareagenten unschärfer und allgemeiner gefaßt.

Der Begriff des Agenten ist bisher nicht allgemein akzeptiert definiert. Dies liegt darin begründet, daß verschiedene Anwendungsbereiche unterschiedliche Schwerpunkte auf einzelne Eigenschaften des Agenten setzen, wobei der Begriff der Autonomie in allen Bereichen eine zentrale Bedeutung für das Agentenkonzept besitzt (Abschnitt 3.1). Jedoch ist auch dieser nicht einheitlich definiert. [Woo 99] definiert den Begriff eines Agenten wie folgt:

> An *agent* is a computer system that is *situated* in some *environment,* and that is capable of *autonomous action* in this environment in order to meet its design objectives. (S. 29, Zeile 6-8)

## 1.1 Motivation

Obwohl die Agententechnologie noch weit davon entfernt ist standardisiert[1] zu werden, läßt sich schon heute abschätzen, welche Bedeutung ihr in Zukunft in der Anwendungsentwicklung zukommen wird.

Grundsätzlich stellt sich bei jeder neuen Technologie die Frage, ob sie nicht allgemein bekannte Konzepte mit einem neuen Namen versieht und ob sie wirklich in der Praxis einen qualitativen Fortschritt beim Entwurf von Problemlösungen darstellt. Eine neue Technologie wird sich in Softwareprojekten nur dann durchsetzen können, wenn sie bei der Kosten-Nutzen-Abschätzung andere Technologien ausstechen kann und insbesondere heutige iterative Softwareentwicklungszyklen, welche in wenigen Monaten ein Produkt in seiner Grundfunktionalität auf den Markt bringen läßt, besser unterstützt.

[Par 99] charakterisiert Probleme, welche sich mit der Agententechnologie erfolgreich lösen lassen, durch eine möglichst große Kombination aus folgenden Kategorien:

- *inhärent modular*: Das Anwendungsfeld zerfällt auf natürlich Art und Weise in Module mit hohem Zusammenhalt und geringen Beziehungen nach außen.
- *dezentral*: Die Anwendung kann in einzelne Prozesse zerlegt werden, welche ohne Anleitung eines anderen Prozesses nützliche Aufgaben erledigen können.
- *veränderlich* oder *schlecht strukturiert*: Das Anwendungsfeld ist dynamisch und verändert seine Struktur. Teilprobleme werden dynamisch in Beziehung zueinander gesetzt und können diese Beziehung wieder verlieren. Oder zwei Anwendungen des selben Anwendungsfeldes unterscheiden sich in ihrer Struktur, wobei das Anwendungssystem das gesamte Feld abdecken soll.
- *komplex*: Die Anzahl der Fälle ist aufgrund der Kombination von Zuständen und Verhaltensweisen in den Modulen des Systems hoch. Ein Programmcode, der diese Fälle explizit unterscheidet, wäre nicht mehr wartbar.

Von einem Anwendungssystem wird dann gefordert, daß es selbst *wandelbar* [SFB 99] ist: Über die Laufzeit des Systems soll die Architektur des Anwendungssystems in der Lage sein, sich auf ändernde Anforderungen einzustellen und sich an neu hinzukommende Teilprobleme anzupassen. Das Anwendungssystem soll in der Lage sein, neue Module aufzunehmen, ohne daß es für eine Wartung angehalten werden muß. Ebenso sollen Module während des Betriebs außer Dienst gestellt werden können. Auf veränderte Strukturen soll das System selbständig durch eine neue Kombination von Problemlösungen reagieren.

Anwendungsfelder, welche die genannten Charakteristika besitzen, sind beispielsweise die Koordination der Verkehrsströme des Individualverkehrs, die Steuerung innerhalb von Kraft- oder Luftfahrzeugen, die Produktion von variantenreichen Serienprodukten oder die Telekommunikation. Als sehr aussagekräftiges Experimentierfeld hat sich im universitären Bereich der Roboterfußball (RoboCup) herausgestellt, bei dem zwei Teams von Robotern in einem abgesteckten Spielfeld nach fest definierten Regeln gegeneinander Fußball spielen.

1. Ein Gremium, welches die Standardisierung im Bereich der Produktion vorantreibt ist das *National Industrial Information Infrastructure Protocols Program* [NIIP 98]. Die Foundation for Intelligent Physical Agents (FIPA) ist ein weltweites Konsortium, welches einen allgemeinen Agentenstandard erarbeitet [Chi 98].

Die praktische Bedeutung der Agententechnologie liegt darin begründet, daß durch sie die Beherrschbarkeit von Softwaresystemen, auf deren Anwendungsbereich die oben genannten Kriterien zutreffen, qualitativ verbessert wird. Obwohl diese Technologie auf bestehenden Technologien (Objektorientierung, Client-Server-Interaktion, Kommunikationsprotokolle, Trader und Brokermechanismen, KI-Techniken und viele andere) aufbaut, erhält sie ihre Mächtigkeit durch die Kombination der Techniken unter einem neuen Prinzip:

Das Operationsprinzip in Multiagentensystemen ist definiert durch die *Autonomie* der Agenten. Dabei definiert [Lev 96] Autonomie als ein dem Agenten übertragenes *Recht*, wobei entsprechende *Selbständigkeit* in seinen Aufgabenbereichen von ihm gefordert wird. Selbständigkeit bedeutet nach [Lev 96]:

- eigenständige *Entscheidungsfähigkeit* durch *Wahrnehmen*, *Überwachen*, *Planen*, *Schlußfolgern* und *Abschätzen der Konsequenzen* seines Handelns;
- aufgabenbezogenes Handeln entsprechend den *Zielvorgaben*, welche durch eine *Kombination aus Planungs- und Überwachungsschritten* gewährleistet werden;
- *Lernen* und Beseitigen von Störungen;
- *Fähigkeit zur Kooperation* als eine Fähigkeit *Zielvorgaben abzustimmen* und eigene Pläne mit denen anderer zu *koordinieren*, um *gemeinsame Ziele zu erreichen.*

Zum Begriff der Selbständigkeit kommt wesentlich hinzu, daß der Agent das *Recht* hat, selbständig zu agieren. Dabei wird sein Recht und das Recht der Anderen durch *Vorgaben*, *Regelungen* und *Normen* geschützt, welche ihre *Unabhängigkeit* gewährleisten. Erst wenn dies erfüllt ist, kann man von der *Autonomie* eines Agenten sprechen.

Nach diesem Prinzip ist es jedem Modul des Anwendungssystems gestattet, abweichend von der von ihm geforderten Leistung innerhalb des Anwendungssystems ein Leistungsspektrum entsprechend seinen aktuellen Fähigkeiten zu erbringen, welches von einer vollständigen Erfüllung einer Aufgabe über ihre partielle Erfüllung bis hin zu einem Ausfall reicht. Bei konventionellen Techniken gibt es nur eine korrekte Abarbeitung der Aufgabe oder eine Ausnahmebehandlung. Schränkt man die Autonomie in verschiedenen Graden ein, so erhält man ein Instrument, mit dem man je nach der Stellung eines Moduls im Gesamtsystem dessen Korrektheit enger oder weiter definieren kann.

Durch das Recht eines jeden Agenten, die von ihm zu erbringende Leistung abweichend von der Spezifikation mit einer niedrigeren Qualität zu erbringen, erhält der Agent die Möglichkeit, Belastungsspitzen auszugleichen und Ressourcenengpässe abzupuffern. Dies müssen die Interaktionspartner wahrnehmen können und ihrerseits durch Anpassung ihrer Leistung *reagieren.* Alternativ dazu kann ein Interaktionspartner einen anderen Agenten suchen, welcher in der Lage ist, die gewünschte Leistung in der geforderten Qualität zu erbringen. Hierzu beendet er die Interaktionsbeziehung zu dem Agenten, bei dem die Belastungsspitze aufgetreten ist, und baut eine neue Beziehung zu dem Ausweichagenten auf.

Hierfür kommen einerseits bekannte Broker und Trader-Techniken zum Einsatz. Weiterhin werden diese Techniken um Verhandlungsmechanismen erweitert, welche im Bereich der Rechnernetze bei der Verhandlung der Qualität des Dienstes auch bekannt sind. Diese Verhandlungstechniken sind im allgemeinen jedoch bilateral und nicht für eine Menge von Prozessen definiert, welche über eine Leistungskette in Beziehung zueinander stehen.

Ein Multiagentensystem ist also in der Lage, Strukturveränderungen vorzunehmen, welche durch eine multilaterale Entscheidungsfindung unter Verwendung von Verhandlungen erfolgt. Je nach den Strukturen, in welche die Agenten eingebunden sind, ist es möglich, Optimierungen für das Gesamtsystem, eine Leistungskette oder einen Aufgabenbereich zu finden. Dabei werden die aktuellen Fähigkeiten des Systems durch die Suche und die Verknüpfung der einzelnen Fähigkeiten der Agenten unter den gegebenen Verhältnissen ausgenutzt. Eine wesentliche Bedeutung kommt dabei den Entscheidungsstrategien der einzelnen Agenten zu, welche das Erreichen von Resultaten erst ermöglichen. Die für eine Entscheidungsfindung benötigten Ressourcen wie Zeit und Rechenleistung gehen dabei in die Strategien mit ein, so daß über sie die Echtzeitfähigkeit garantiert werden kann. Sind die Ressourcen beschränkt, so wird zumindest eine suboptimale Variante realisiert, welche die geforderte Leistung in einer noch zulässigen Qualität erbringt.

Der Fortschritt, den die Agententechnologie der Anwendungsentwicklung bereitstellt, ist es, Systeme zu entwerfen, welche in der Lage sind, sich selbst zu organisieren. Dies ist für Anwendungsbereiche hilfreich, welche fortlaufend die Grenzen ihrer Gleichgewichtszustände überschreiten. Nach [Hak 83a, Hak 83b] treten Strukturveränderungen an Bifurkationspunkten von Systemen auf. Die Erkenntnisse der Synergetik und der Chaostheorie zeigen, daß trotz des Verlassens von Gleichgewichtsbereichen stabile Systeme konstruiert werden können. Insbesondere durch die Nutzung von Entscheidungsstrategien, die chaotische Bereiche vermeiden, unterstützt die Agententechnologie den Entwurf solcher Anwendungssysteme [LSA 98, LSAL 99].

Die Kosten und die Risiken, welche mit der Einführung der Agententechnologie in Softwareprojekte verbunden sind, lassen sich einerseits in Risiken aufteilen, welche bei der Einführung jeder neuen Technologie auftreten und hier nicht behandelt werden sollen. Andererseits bestehen Kosten und Risiken, die durch die Technologie selbst begründet sind.

Da die Agententechnologie auf eine Vielzahl von weiteren Technologien aufbaut, ist mit ihrer Einführung immer auch eine Vielzahl von anderen Methoden und Techniken zu installieren. Damit verbunden sind häufig Architekturgerüste (engl. Frameworks) für verschiedene Bereiche, die unterschiedliche Architekturkonzepte und Entwurfsmethoden besitzen. Ein Entwicklungsteam, das diese Techniken nicht beherrscht, ist schnell überfordert. Im praktischen Einsatz ist die Verfügbarkeit von qualitativ hochwertigen Architekturgerüste für Multiagentensysteme von entscheidender Bedeutung, da im Unterschied zum universitären Bereich ein eigenes Erstellen von Werkzeugen oder Architekturgerüsten aus Zeit- und Kostengründen nicht in Betracht kommt. Die Qualität eines Architekturgerüstes ist daran zu messen, inwieweit der Entwickler Mechanismen in

die Hand gelegt bekommt, mit denen er in die Lage versetzt wird, ein kohärentes Verhalten der Agenten zum Entwurfszeitpunkt konzipieren und zusichern zu können.

## 1.2 Zielsetzung

Ziel der vorliegenden Dissertation war es, aufbauend auf den Erfahrungen, die beim Entwurf der Agentenarchitektur im Projekt Comros (Cooperative Mobile Robots Stuttgart) hinsichtlich der Koordination von Agenten gesammelt wurden, ein Interaktionsmodell für Multiagentensysteme zu entwerfen. Dieses Modell stellt die theoretische Grundlage für die im Teilprojekt C4 “Informationstechnische Kommunikation und Kooperation” im Sonderforschungsbereich 467 an der Universität Stuttgart über “Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion” entworfene Entwicklungsumgebung Rope (Role Oriented Programming Environment for Multiagent Systems) [BGKM 99, BKM 99, BML 97, SFB 99] dar.

Durch die Verwendung des Modells bei der Modellierung eines Multiagentensystems besitzt der Entwickler ein Konzept, mit welchem er in der Lage ist, das Systemverhalten zum Entwurfzeitpunkt durch die Definition von Interaktions- und Koordinationsmechanismen so zu bestimmen, daß zur Laufzeit ein kohärentes Verhalten der dezentral ablaufenden, autonomen Agenten erzielt wird.

Änderungen an der Definition der Koordinationsmechanismen sollten keine Reimplementierung von Teilen der Agenten erfordern, die den Mechanismus anwenden sollen. Dies ist Voraussetzung für die geforderte Wandelbarkeit der Anwendungssysteme. Weiterhin soll es konzeptionell möglich sein, während der Laufzeit neue Interaktionsmechanismen dem System zur Verfügung zu stellen, welche dazu genutzt werden, die Fähigkeiten der Agenten auf eine neu Art organisieren.

## 1.3 Aufbau der Arbeit

Kapitel 2 gibt einen kurzen Literaturüberblick, über Gebiete und Bereiche, die mit dieser Dissertation in engem Zusammenhang stehen. Dabei ist es nicht das Ziel, die Agententechnologie lehrbuchartig aufzuarbeiten, sondern sie entsprechend der Bedeutung und des Bezugs zu dieser Arbeit aufzuführen.

In Kapitel 3 werden ausgehend von den Anforderungen, die an eine Entwurfssprache für Multiagentensysteme zu stellen sind, eine Einführung in die in dieser Arbeit benutzten Konzepte gegeben, um anschließend das Interaktionsmodell in drei Iterationsstufen vorzustellen und es anhand eines einheitlichen Beispieles einer Auktion zu veranschaulichen.

Die *erste Iterationsstufe* beschränkt das Modell auf Rollen, die von Agenten bei der Teilnahme an einem Interaktionsmechanismus angenommen werden. Hierzu werden Phasen eingeführt, wobei eine Phase die miteinander interagierenden Rollen gruppiert. Transitionen spezifizieren das zu erreichende Ziel, deren Umsetzung durch an den Rollen definierte Aktionen vorgenommen werden. Die Rollen sind als Handlungstypen zu verstehen, welche mit der Übernahme durch einen Agenten mit Leben gefüllt werden. Dabei kann eine Rolle mehrfach instanziiert (oder Rollenmarken) sein, welche gleichzeitig miteinander oder mit anderen interagieren. In der Diskussion am Ende der ersten Iterationsstufe wird deutlich, daß aufgrund des Mangels an Kommunikationskanälen und Interaktionsobjekten die Spezifikation von Interaktionsmechanismen unvollständig ist.

Daraufhin werden in der *zweiten Iterationsstufe* Ressourcen eingeführt, über welche die Agenten interagieren und von denen im Interaktionsmodell in Form von Objekten abstrahiert wird. Ähnlich den Agenten, die beim Schalten einer Transition Rollenwechsel vollziehen, führten Ressourcen beim Schalten von Transitionen Objektwechsel aus. Objekte definieren die Schnittstellen (als Sichten auf die Ressourcen), die in einer Phase durch dort agierende Rollenmarken genutzt werden. Durch Ressourcen lassen sich beispielsweise Kanäle definieren, über welche die Rollen oder wie in Kapitel 5 ausgeführt wird, die Agenten effizient kommunizieren können. Im Beispiel der Auktion wird der Versteigerungskatalog durch eine Ressource definiert, in dem die Rollenmarken individuell und nebenläufig navigieren können.

In der *dritten Iterationsstufe* wird das Modell um das Modulkonzept erweitert. Das der Realisierung des Modulkonzepts zugrundeliegende Verständnis besteht darin, daß jede Ressource beispielsweise durch interne Sperrmechanismen die zulässigen Interaktionsabläufe selbst einschränkt. Da ein Interaktionsmechanismus selbst eine genaue Spezifikation der zulässigen Interaktionen darstellt, ist es naheliegend, einen als Modul verwendeten Interaktionsmechanismus als Ressource zu begreifen und die Aufrufschnittstelle in Form eines Objekts zu beschreiben. Fragestellungen nach dem Eintritt in einen Interaktionsmechanismus, welcher insbesondere in offenen Systemen wie es die Multiagentensysteme darstellen, auch nach dem Start des Interaktionsmechanismus möglich ist, lassen sich auf elegante Weise lösen.

Die Schnittstelle zwischen dem Interaktionsmodell und den Agenten oder Ressourcen wird in Form von Diensten beschrieben, welche die Fähigkeiten der Agenten und die Funktionalität der Ressourcen entsprechend den im Multiagentensystem definierten Handlungen spezifiziert. Die möglichen Handlungen werden aus dem Anwendungsbereich abgeleitet und sind unabhängig von konkreten Agenten oder Ressourcen definiert. Ein Agent oder eine Ressource realisiert im Allgemeinen nur einen Ausschnitt der Handlungen. Durch die Menge der Rollen (oder Objekte) die ein Teilnehmer im Laufe einer Interaktion durchläuft, wird vorgegeben, welche Handlungen der Teilnehmer unterstützen muß, um sich an dem Interaktionsmechanismus beteiligen zu können. Besitzt er die geforderten Fähigkeiten nicht, kann er erst gar nicht dem Interaktionsmechanismus beitreten.

Kapitel 4 stellt die Roboterarchitektur in Comros als konkretes Multiagentensystem vor. In ihr werden einzelne Sensoren mit den zugehörigen Aktoren gruppiert und durch einen sogenannten Elementaragenten gesteuert. Der Elementaragent definiert selbst wieder ein

virtuelles Sensor-Aktor-Paar. Elementaragenten werden durch Datenflußnetzwerke verbunden, die aus einzelnen überlagerten Pipelines bestehen, welche das arbeitsteilige Zusammenwirken hinsichtlich einer Aufgabe beschreiben. Die Elementaragenten koordinieren sich über sogenannte Entscheidungsnetzwerke, welche die Weltmodelle der einzelnen Agenten durch Einsatz der Restriktionstechnik konsistent halten. Die Entscheidungsnetzwerke sind als Interaktionsmechanismus entsprechend Kapitel 3 definiert. Am Beispiel eines Gedankenexperiments wird die Umsetzung der Architektur dargestellt.

In Kapitel 5 wird das Interaktionsmodell am Beispiel eines Puffers der Datenflußnetze angewendet. Die Anwendung war Teil des Teams der Universität Stuttgart bei der Fußballweltmeisterschaft 1999 für Roboter in der Middle Size League. Das in diesem Kapitel beschriebene Interaktionsverfahren ist für die Administration der Kommunikation zuständig, über welche die einzelnen Roboter ihre Eigenposition und die Position des von ihnen erkannten Balles austauschten. Die Kommunikation wurde durch eine Ressource realisiert, die verteilt auf jedem Roboter vorliegt und die Funktionalität des Datenpuffers bereitstellt. Das Interaktionsverfahren ist für die Steuerung des Spielerwechsels (Ersatzspieler kommt auf Spielfeld und ersetzt einen Spieler), für die Initialisierung und für das Abbauen der Kommunikation verantwortlich.

Im Schlußkapitel werden die durchgeführten Arbeiten nochmals zusammengefaßt und eine Einschätzung der erzielten Ergebnisse gegeben.

# Kapitel 2

# Stand der Forschung

## 2.1 Agentenorientierte Programmierung

Der Begriff der “Agentenorientierten Programmierung (AOP)” wurde von Y. Shoham 1989 geprägt [Sho 89], [Sho 93], [Sho 97]. Shoham faßt die Metapher der Objektorientierten Programmierung als das Modell eines Information verarbeitenden Systems auf, das aus Modulen besteht, welche miteinander kommunizieren und welche auf individuelle Art und Weise eintreffende Nachrichten bearbeiten. Shoham sieht die Wurzel der Idee der Objektorientierten Programmierung in dem *Actor* Formalismus von Hewitt [Hew 77].

### 2.1.1 Agent-0

Das von Shoham vorgeschlagene Programmierkonzept ist nach seinem Verständnis eine Spezialisierung des Zustandes von Modulen (nun als *Mentale Zustände* von *Agenten* bezeichnet). Mentale Zustände bestehen seiner Meinung nach

- aus Meinungen (*Belief*) über die Umwelt, über den Agenten selbst und über andere Agenten,
- aus Pflichten (*Obligation*) oder Verpflichtungen des Agenten,
- aus Entscheidungen (*Choice*), definiert als Selbstverpflichtungen des Agenten, und
- aus Fähigkeiten (*Capability*).

Für jeden Mentalen Zustand wird in einer modalen Logik ein entsprechender Operator definiert, der zeitbehaftet ist. Zwischen den mentalen Zuständen existieren Restriktionen, welche sicherstellen, daß sie dem “gesunden Menschenverstand” entsprechen:

- Interne Konsistenz (*Internal consistency*): Die Meinungen des Agenten sowie die Pflichten des Agenten sind in sich konsistent;
- Gewissenhaftigkeit (*Good faith*): Agenten verpflichten sich nur zu Aufgaben, zu deren Lösung sie sich für fähig halten.
- Selbstbeobachtung (*Introspection*): Agenten sind sich über die Verpflichtungen, die sie eingegangen sind, bewußt, wissen aber nicht notwendigerweise über Verpflichtungen anderer ihnen gegenüber Bescheid.

Die Kommunikation zwischen den Agenten erfolgt in Anlehnung an die Sprechakt-Theorie, wobei sich Shoham auf drei Typen beschränkt: *Inform*, *Request*, und *Unrequest*. Die Kommunikation ist der Ausführung einer *privaten Aktion* gleichgestellt. Aufbauend auf dieser Sprache definiert er Regeln, welche er *mentale Muster* nennt.

Für Shoham besteht ein AOP System aus drei Komponenten:

- Eine formale Sprache mit einer klaren Syntax und Semantik zur Beschreibung von mentalen Zuständen
- Eine interpretierte Programmiersprache, mit der Agenten definiert und programmiert werden können, und welche Kommunikationsprimitive wie *Request* und *Inform* bestitzt. Die Semantik der Programmiersprache entspricht der Semantik der formalen Sprache.
- Ein “Agentifizierer”, der neutrale Geräte in programmierbare Agenten konvertiert.

Shoham stellt als mögliche Programmiersprache *Agent-0* vor. Die Integration von KQML ([FLM 97]) in die Sprache wurde mit *Agent-K* [DaEd 94] vorgeschlagen.

### 2.1.2 Mentale Zustände nach Cohen und Levesque

Im Unterschied zu Shoham führen Cohen und Levesque ([CoLe 90]) Kategorien für Absichten (*Intention*) und Ziele (*Goal*) ein, wohingegen Pflichten und Fähigkeiten nicht vorkommen. Im Einzelnen umfassen die mentalen Zustände folgende Kategorien:

- Meinungen (*Belief*) über die Umwelt, über andere Agenten und über sich selbst., wobei im Vergleich zu Shoham der zeitliche Bezug fehlt.
- Ziele (*Goals*) eines Agenten sind eine konsistente Teilmenge der Wünsche eines Agenten. Es wird zwischen zu erreichenden Zielen (*Achievement goals*) und Zielen, welche durch die Beibehaltung einer Situation charakterisiert sind

(*Maintenance goal*), unterschieden. Eine Untermenge wird von den dauerhaften Zielen (*Persistent goal*) gebildet, welche der Agent nie aufgibt, außer sie sind erreicht oder nie mehr erreichbar.

Die Ziele eines Agenten dürfen sich nicht widersprechen und Konsequenzen, welche aus zwei Zielen eines Agenten folgen, sind ebenfalls Ziele des Agenten.

- Wünsche (*Desires*) unterscheiden sich von den Zielen (*Goals*) durch ihre fehlende Konsistenz. Wünsche konkurrieren solange untereinander – verstärkt durch entsprechende Meinungen (*Belief*) des Agenten – bis der Agent die Wahl (*choose*) trifft und aus einem Wunsch ein Ziel macht.
- Absichten (*Intention*) sind dauerhafte Ziele (*Persistent goal*) des Agenten, von denen der Agent annimmt (*believe*), daß sie durch Ausführen von Aktionen oder Plänen erreicht werden können.

Leider geben Cohen und Levesque keine Programmiersprache an, in der man Agenten nach ihrem Modell entwerfen kann.

### 2.1.3 BDI-Architekturen

BDI-Architekturen stehen für eine Klasse von Architekturen, denen die mentalen Zustände (*Belief, Desires, Intention*) gemeinsam sind. Im Unterschied zu Cohen und Levesque verschmelzen sie Wünche und Ziele. Ihnen fehlt damit das Konzept des dauerhaften Ziels, das der Agent zu einer Absicht (*Intention*) überführt, sobald er meint das Ziel erreichen zu können und die Ausführung eines entsprechenden Plans anstößt.

Nach [Woo 99] besteht ein BDI-Agent aus folgenden Komponenten:

- Einer Menge von aktuellen Meinungen, welche die Information des Agenten über seine Umwelt darstellen;
- eine Funktion zur Überprüfung der Meinungen des Agenten, welche aufbauend auf den Wahrnehmungen und auf Basis der Meinungen des Agenten eine neue Menge von Meinungen erzeugt;
- eine Funktion zum Generieren von Optionen oder Wünschen, welche dem Agenten auf der Basis seiner aktuellen Meinungen über die Umwelt und seinen aktuellen Intentionen zur Verfügung stehen;
- eine Menge von Optionen, welche die möglichen Handlungsrichtungen repräsentieren, welche dem Agenten offen stehen;
- Eine Filterfunktion, welche den Planungsprozeß des Agenten darstellt. Sie bestimmt die Absichten des Agenten unter Berücksichtigung seiner aktuellen Meinung, seinen Wünsche und den bisherigen Absichten.
- Eine Menge von Absichten.

- Eine Funktion zur Auswahl von Aktionen: Sie bestimmt, welche Aktion, ausgehend von der Menge der Absichten, als nächstes ausgeführt werden soll.

Die Interaktion mit anderen Agenten erfolgt durch Aktionen und Wahrnehmungen über einen sogenannten Kommunikationskanal. Interaktionsabläufe zerlegen sich also in einzelne Nachrichten, welche zwischen den Agenten ausgetauscht werden.

### 2.1.4 KIF, Ontolingua und KQML

KQML (Knowledge Query and Manipulation Language) hat das Ziel, eine von allen Agenten unterstützte Kommunikationssprache zu werden. Das Format des ausgetauschten Wissens wird durch KIF (Knowledge Interchange Format) spezifiziert. Während KIF die syntaktischen Aspekte spezifiziert, definiert Ontolingua die semantischen Aspekte. Jede Ontologie wird durch eine Menge von Klassen, Funktionen und Konstanten definiert und enthält Axiome, welche Restriktionen hinsichtlich der zulässigen Interpretationen definieren [FLM 97, Ont 99].

Die Sprache KQML besteht aus drei Schichten, der Inhaltsschicht, der Nachrichtenschicht und der Kommunikationsschicht. Die Inhaltsschicht deckt den eigentlichen Inhalt einer Nachricht in der dem Programm eigenen Sprache ab. Die Kommunikationsschicht kodiert Nachrichtenattribute, wie Sender und Empfänger, sowie einen Identifikator für die Sitzung. Die Nachrichtenschicht enthält den Inhalt der Nachricht in einer anwendungsunabhängigen Sprache. Sie enthält den Sprechakt (Tabelle 2.1), mit den der Absender den Inhalt der Nachricht attributiert hat. Optional kann in der Nachrichtenschicht die eine Beschreibung übermittelt werden, welche Ontologie in der Inhaltsschicht zu verwenden ist.

| **Kategorie** | **Name** |
|---|---|
| Basic Query | evaluate, ask-if, ask-about, ask-one, ask-all |
| Multi-response (query) | stream-about, stream-all, eos |
| Response | reply, sorry |
| Generic informational | tell, achieve, cancel, untell, unachieve |
| Generator | standby, ready, next, rest, discard, generator |
| Capability-definition | advertise, subscribe, monitor, import, export |
| Networking | register, unregister, forward, broadcast, route |

Tabelle 2.1: Sprechakte in KQML nach [FLM 97]

KQML eröglicht es also mehreren gegebenenfalls sich fremden Agenten, untereinander zu kommunizieren und Informationen auszutauschen. Dabei besitzen die Kommunikati-

onspartner nicht notwendigerweise das gleiche Verständnis von der Interaktion, an der sie teilnehmen.

### 2.1.5 Rollen

Bei den Modellierungsmethoden für Multiagentensysteme wird in der Analysephase ähnlich zu Use Cases häufig die Rollenabstraktion zur Identifizierung verschiedener Aufgaben und Verantwortlichkeiten von Agenten eingesetzt [WJK 99, Ken 98]. In der Entwurfsphase werden diese Rollen jedoch durch Agentenklassen realisiert. Es wird also im Entwurf festgelegt, welche Agentenklasse welche Rollen übernimmt. Dadurch werden jedoch Aspekte der Ablauf- und der Aufbauorganisation in einem einzigen Systemelement integriert, was sich auf die Wandelbarkeit des Gesamtsystems negativ auswirkt.

In [HSK 99] wird in Anlehnung an die objektorientierten Entwurfsmuster ein Katalog agentenorientierter Koordinationsmuster vorgestellt. Es wird jedoch keine Beschreibungssprache oder Architektur entwickelt; das Ziel der Arbeiten ist vielmehr, einen Katalog allgemein verwendbarer Entwurfsmuster für die Interaktion autonomer Agenten zu erstellen. Die Arbeiten scheinen noch in einem frühen Stadium zu sein.

In [StVe 98] wird ein rollenorientierter Ansatz in Umgebungen vorgestellt, welche nur eine periodische Synchronisation im Multiagentensystem erlauben. Die Rollen sind dabei fester Bestandteil des Agenten und steuern dessen internes und externes Verhalten. Der Agent wechselt seine Rolle aufgrund vordefinierter Ereignisse. Die möglichen Interaktionen zwischen Agenten sind durch die Beschreibung des externen Verhaltens der Rollen gegeben und werden nicht explizit modelliert.

## 2.2 Verhandlungen

Interagieren Agenten unter Verwendung einer gemeinsamen Sprache in einem System, so wird dies in der Literatur häufig als Verhandeln bezeichnet. [JSW 98] faßt die wesentlichen Charakteristika einer Verhandlung wie folgt zusammen:

- Es existiert eine Form des Konflikts, der auf dezentrale Weise gelöst werden muß
- Das System besteht aus eigennützigen Agenten, mit
  - begrenzter Rationalität
  - und unvollständigem Wissen.
- Die Agenten kommunizieren und tauschen iterativ Vorschläge und Gegenvorschläge aus.

### 2.2.1 Verhandlungsprotokolle

Einen spieltheoretischen Ansatz beim Entwurf von Verhandlungsprotokollen verfolgen [RoZl 94] aufbauend auf den Arbeiten von [Ros 85]. Agenten besitzen eine Utility-Funktion, die sie optimieren, wobei sie als allwissend angenommen werden. Der Nutzen der Agenten wird in der Spieltheorie [FuTi 91] üblich, durch sogenannten Payoff-Matrizen ermittelt und ist den Agenten bei der Verhandlung bekannt. Die Autoren charakterisieren unterschiedliche Bereiche, in denen sie nachweisen, daß eine Übervorteilung des Verhandlungspartners durch Lügen oder Verschweigen von Informationen nicht von Vorteil ist, wenn das Verhandlungsergebnis nach gewissen Regeln vereinbart wird.

Hahndel ([Han 95]) stellt mit seinem System NEDIP ein agentenbasiertes Planungssystem für flexible Fertigungsumgebungen vor. Erstmals werden in dieser Arbeit Produktionsaufträge auf flexiblen Fertigungsmaschinen unter Verwendung von Verhandlungsprotokollen eingeplant, ohne daß eine zentrale Komponente koordinierend eingreift. Der verwendete adaptive Ansatz führt die Planung jedoch nicht anhand von Endterminen durch, sondern verhandelt ausgehend von einem Anfangstermin in fortlaufenden Wellen die aktuell beste Ressourcenzuordnung für einen gewissen Planungshorizont. Das von ihm verwendete Verhandlungsprotokoll, basiert auf dem Contractnet-Protokoll.

Das Contractnet-Protokoll [Smi 80] wurde für die Zuordnung von Aufgaben zu Ressourcen entworfen. Es gibt zwei unterschiedliche Teilnehmerarten an dem Verfahren: den *Manager* und die *Contarctors*. Der Manager schreibt seine Aufgabe aus und erhält von den Contarctors Angebote erstellt. Unter diesen Angeboten wählt er dasjenige, welches seinen Bedürfnissen am besten entspricht.

### 2.2.2 Koordination

Während der Begriff der Verhandlung den Mechanismus beschreibt, mit welchem die Agenten einen Konflikt lösen, bezieht sich der Begriff der *Koordination* auf das zeitliche, räumlich oder allgemein an Restriktionen gebundene Abstimmen von Aktivitäten einzelner Agenten. In der Literatur hat sich der Begriff der Koordination als Unterbegriff der Interaktion durchgesetzt. Entsprechend wird auch von Koordinationsprotokollen gesprochen.

[HuSt 99] charakterisieren das Vorgehen, mit welchem Koordination erreicht wird, wie folgt:

1. Definieren des Zielgraphen verbunden mit der Identifikation und Klassifikation der Abhängigkeiten
2. Zuweisen von Ausschnitten des Graphen zu entsprechenden Agenten
3. Steuerung der Entscheidungen, welcher Bereich des Graphen exploriert wird
4. Durchlaufen des Graphen
5. Erfolgreiches Durchlaufen bekanntgeben

Einige dieser Phasen sind von Agenten gemeinschaftlich zu erbringen, wohingegen andere durch die Agenten einzeln zu erledigen sind. Welcher der beiden Ansätze in den einzelnen Phasen gewählt wird ist ein Aufgabe des Systementwurfs.

Das Schlüsselkonzept sind Selbstverpflichtungen (*commitments*) und Konventionen (*conventions*) [Jen 96]. Selbstverpflichtungen werden als Versprechungen angesehen, eine spezifizierte Folge von Aktionen auszuführen, wohingehend Konventionen ein Mittel darstellen, die Selbstverpflichtungen unter sich ändernden Bedingungen zu handhaben.

## 2.3 Petrinetze

Der Begriff der Petrinetze wurde von C. A. Petri Anfang der 60er Jahre eingeführt. Die Vorteile dieses Modells liegen vor allem darin, daß es abgestuft nach den verschiedenen Netzklassen, sowohl eine hohe Ausdrucksmächtigkeit bietet, aber auch Analysemethoden für Eigenschaften eines Entwurfes bereitstellt, die schon zum Entwurfzeitpunkt Aussagen über das System zulassen (z. B. Deadlock-Freiheit). Die Analysefähigkeiten sind für Netzklassen mit eingeschränkter Ausdrucksmächtigkeit (z.B. Free-Choice-Netze) jedoch wesentlich besser, als für ausdruckstarke Klassen (z.B. Pr/T-Netze).

Der zunehmende Einsatz von Multiprozessorarchitekturen bei Großrechnern (z.B. Cray T3E) und von vernetzten Rechnersystemen haben das Interesse an einem fundierten mathematischen Modell wachsen lassen. Die Attraktivität von Petrinetzen besteht in der Art und Weise, in der die grundlegenden Aspekte von verteilten Systemen sowohl konzeptionell als auch mathematisch identifiziert werden. Es sind eine Fülle von PN-Klassen entstanden, die sich ähnlich wie Programmiersprachen (Assembler oder objektorientierte Sprachen) in ihrer Ausdrucksfähigkeit unterscheiden. Man kann jedoch zeigen, daß ausdrucksfähigere PN-Typen als *Faltungen* von bestimmten einfachen Typen dargestellt werden können, das heißt die einfachen Netzklassen sind gleich mächtig wie die für den Menschen intuitiveren Typen. Somit lassen sich die Analyseverfahren auch auf höhere Netze anwenden bzw. diese Verfahren lassen sich entsprechend erweitern. Viele der Analyseproblemen sind jedoch so komplex, daß sie für größere Petrinetze nicht oder kaum mehr berechnet[1] werden können.

Petrinetze haben in vielen praktischen Projekten ihre Einsatzfähigkeit unter Beweis gestellt. So wurden sie zur Spezifikation von Kommunikationsprotokollen [HaMo 90] in verteilten Systemen verwendet oder in Anwendungen mit hohen Sicherheitsanforderungen (z.B. bei der Entsorgung von atomaren Abfall [MoPi 94], als auch bei der Validierung von Chip-Entwürfen [Rüd 92]) eingesetzt.

Die Stärke von Petrinetzen liegt in der Abstraktion von physischen Ressourcen in einem verteilten System und in der Darstellung von Abläufen von dynamischen Vorgängen. Dabei werden aktive und passive Komponenten gleichrangig behandelt. Die Spezifikation eines Systems kann sowohl auf abstrakter Ebene als auch in starker Detaillierung erfolgen.

**Definition 2.1** *allgemeine Petrinetze*

*Petrinetze* sind bipartite, gerichtete Graphen $N = (S, T; F, M_0)$, deren Knoten *Stellen S* und *Transitionen T* sind. Die Kanten werden durch eine *Flußrelation* $F \subseteq S \times T \cup T \times S$ beschrieben, welche nur Verbindungen zwischen unterschiedlichen Knotentypen zuläßt. Durch Angabe einer (*Start-*)*Markierung* $M_0$ wird das dynamische Verhalten des Petrinetzes initialisiert.

Transitionen sind die aktiven Komponenten des Netzes und repräsentieren Aktionen oder Ereignisse, wohingegen Stellen die passiven Komponenten sind, die lokale *Bedingungen* definieren, von denen das Eintreten von *Ereignissen* (man sagt auch das *Schalten* von Transitionen) abhängt. Die Abhängigkeiten eines Ereignisses von Bedingungen werden durch gerichtete Kanten beschrieben, welche durch die Flußrelation definiert sind. Dabei kann eine Transition nur von Stellen, nicht aber von anderen Transitionen unmittelbar abhängen. Ebenso ist eine Nachbarschaft von Stellen ausgeschlossen.

Zu jedem Knoten $v$ läßt sich ein *Vorbereich* $\cdot v = \{w \in S \cup T | (w, v) \in F\}$ und ein *Nachbereich* $v\cdot = \{w \in S \cup T | (v, w) \in F\}$ als die Menge der adjazenten Knoten definiert, die im Sinne der Kantenrichtung vor $v$ bzw. hinter (nach) $v$ liegen (Abbildung 2.1).

Über die Menge der Stellen ist nun eine *Markierung* (Abbildung) definiert, die jeder Stelle eine ggf. leere Menge von Marken zuweist. Befindet sich eine Marke auf einer Stelle, so ist die von der Stelle symbolisierte Bedingung durch diese Marke als eingetreten gekennzeichnet. Durch Schalten von Transitionen werden Marken von vorgelagerten Stellen abgezogen und Marken auf nachgelagerten Stellen abgelegt. Man sagt das Schalten der Transition $t$ überführt die Markierung $M$ in die Markierung $M'$ (i.Z. $M[t > M'$).

1. So sind Fragestellungen wie die Deadlock-Freiheit oder die Trace-Äquivalenz zweier Petrinetze auf das Erreichbarkeitsproblem reduzierbar. Die Komplexität der Erreichbarkeit ist mindestens EXPSPACE-schwer.

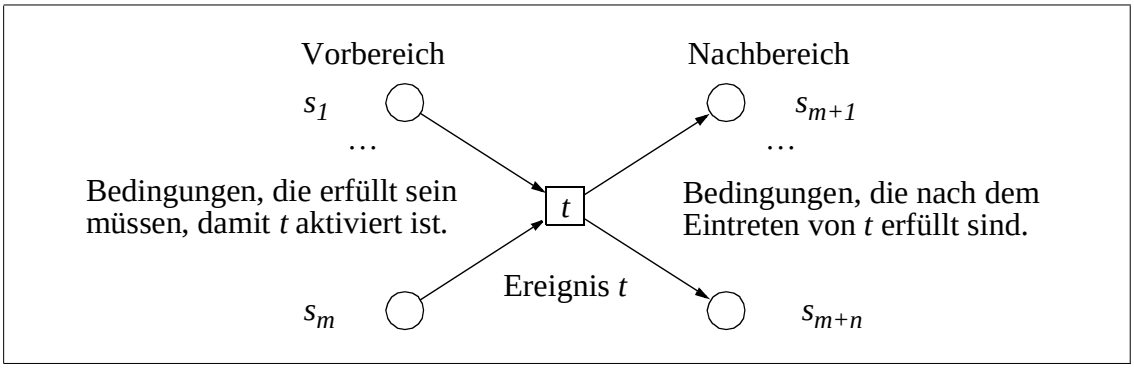


Abbildung 2.1: Bedingungen im Vor- und Nachbereich eines Ereignisses

Eine Transition $t$ kann jedoch nur schalten, wenn sie *aktiviert* ist. Dazu müssen die Bedingungen im Vorbereich von $t$ erfüllt sein und der Nachbereich in der Lage sein, die von $t$ erzeugten Marken aufzunehmen. (Bei Netztypen mit beschränkter Stellenkapazität kann es zu sog. *Kontaktsituation* kommen, d.h. daß $t$ wegen überfüllter Stellen im Nachbereich nicht schalten kann, obwohl die Vorbedingungen von $t$ erfüllt sind.)

Die Mächtigkeit von Petrinetzen besteht nicht nur im Ausdruck von kausalen Abhängigkeiten, sondern auch in der Beschreibbarkeit des dynamischen Schaltverhaltens des modellierten Netzes. Ein verteilter Systemzustand, in dem sich ein Petrinetz befindet, wird dabei als *Fall* eingeführt. In einem Fall sind eine Menge von Transitionen aktiviert, die ggf, zueinander in *Konkurrenz* stehen (d.h. durch das Schalten einer Transition würde eine andere, bereits aktivierte Transition deaktiviert). Um die Dynamik eines Petrinetzes zu beschreiben, verwendet man Mengen $U$ von paarweise unabhängigen Transitionen, die unter einem Fall $C$ aktiviert sind (i.Z. $C[U>$). In nebenläufigen Systemen gibt es üblicherweise mehrere dieser Mengen $U$ und manche von ihnen sind hinsichtlich ihrer Kardinalität maximal. Welche dieser Mengen von *nebenläufigen Transitionen* den nächsten *Schritt* des Petrinetzes bestimmt und wie diese Menge ermittelt wird, definiert die Schaltsemantik. Durch einen Schritt des Petrinetzes wird der Fall $C$ in den Fall $C'$ überführt (i.Z. $C[U>C'$). Ob dies parallel in einer atomaren Aktion erfolgt, oder zeitlich entzerrt, ist für das Systemverhalten ohne Belang. Nach Erreichen des Folgefalles $C'$ wird eine neue Menge nebenläufiger Transitionen festgelegt, die den nächsten Schritt bestimmt.

Im Folgenden werden die bekanntesten Petrinetzklassen eingeführt und die Unterschiede in ihrer Ausdrucksmächtigkeit dargestellt.

### 2.3.1 Bedingungs-Ereignis-Netze

Die elementare Petrinetzklasse sind Bedingungs-Ereignis-Netze (B/E-Netze) $N_{B/E} = (S, T; F, M_0)$ . In diesen Netzen kann eine Bedingung erfüllt (eine Stelle markiert) sein, oder nicht. Die Markierung wird als eine Abbildung der Stellen auf Wahrheitswerte definiert $M_0 : S \rightarrow \{0, 1\}$ . Anhand dieser Netzklasse wird in der Literatur in die Theorie von Petrinetzen eingeführt und grundlegende Begriffe (Kontaktfreiheit, Komplementärstellen, Fallgraphen, Petrinetzsprachen, Prozesse) hergeleitet.

### 2.3.2 Stellen-Transitionsnetze

Eine erste Verallgemeinerung von B/E-Netzen sind Stellen-Transitions-Netze (S/T-Netze) $N_{S/T} = (S, T; F, K, W, M_0)$ . Stellen können nun mehrere Marken aufnehmen, bis zu einer ihnen zugeordneten Kapazität $K : S \rightarrow \mathbb{N}_0 \cup \{\infty\}$ . Transitionen haben die Fähigkeit beim Schalten eine größere Anzahl von Marken von einer Stelle abzuziehen. Hierzu werden die Kanten mit entsprechenden Gewichten versehen $W : F \rightarrow \mathbb{N}$ . Die Anfangsmarkierung belegt die Stellen mit Marken und beachtet dabei die Kapazitäten der Stellen $M_0 : S \rightarrow \mathbb{N}_0$ mit $M_0(s) \leq K(s)$ .

Führt man für S/T-Netze inhibitorische Kanten ein, die eine Transition genau dann aktivieren, wenn sich keine Marke auf der zugehörigen Stelle befindet, so kann man damit einen Zähler modellieren. Mit zwei Zählern läßt sich eine Turing-Maschine simulieren, und somit sind S/T-Netze mit inhibitorischen Kanten turingmächtig. Das bedeutet, daß alle berechenbaren Funktionen durch S/T-Netze mit inhibitorischen Kanten berechnet werden können.

Ähnlich wie bei Turing-Maschinen besteht der Reiz von S/T-Netzen in ihrer Einfachheit. Dies ist eine wesentliche Voraussetzung für Handhabbarkeit bei Charakterisierungen des Modells und für die Charakterisierung von Problemen. So ist für das Wohlverhalten einer Systemmodellierung die Frage nach der Beschränktheit des Netzes wichtig. Ist ein Netz *k-beschränkt*, sind also für alle erreichbaren Markierungen die Anzahl der Marken auf einer Stelle kleiner gleich einem festen *k*, so ist garantiert, daß das Netz nicht entartet und nicht unbeschränkt Marken erzeugt. Jedoch ist die Komplexität des Tests der Beschränktheit PSPACE-vollständig, und damit ähnlich aufwendig wie Probleme aus NP.

Eine für einen Netzentwurf ebenfalls interessante Frage ist, ob zu einer gegebenen Startmarkierung eine bestimmte (End-)Markierung erreicht wird. Man kann zeigen, das diese Eigenschaft entscheidbar ist, jedoch ist ihre Berechnungskomplexität EXPSPACE-hart. Das heißt, es existieren keine effizienten Algorithmen, die eine solche strukturelle Eigenschaft analysieren. Akzeptiert man jedoch bei dem Entwurf eines Petrinetzes restriktive Beschränkungen hinsichtlich erlaubter Vernetzungsarten, so kann man eine Unterklasse

der S/T-Netze einsetzen, in denen bestimmte Eigenschaften gelten, oder zumindest Algorithmen existieren, welche solche Eigenschaften effizient testen.

### 2.3.3 S-Systeme

Die Unterklasse der S/T-Netze in der die Eigenschaft der Beschränktheit durch die Konstruktion sichergestellt ist, sind Zustandsmaschinen (*S-Systeme*). In Abbildung 2.2 ist ein Beispielnetz dargestellt, in dem alle zugelassenen Vernetzungen verwendet wurden. Ein Netz $N_S = (S, T; F, K, W, M_0)$ ist dann ein S-System, wenn alle Kanten mit 1 gewichtet sind ($W: F \to \{1\}$), die Kapazität der Stellen unbeschränkt ist ($K: S \to \{\infty\}$), und wenn folgende Einschränkung bei der Vernetzung zutrifft: alle Transitionen ziehen von genau einer Stelle Marken ab und legen auf genau einer Stelle Marken ab ($\forall t \in T: |\cdot t| = |t \cdot| = 1$). Nicht zugelassen sind Gabelungstransitionen, welche Nebenläufigkeit erzeugen, oder Synchronisationstransitionen, welche nebenläufige Pfade zusammenführen. Der Grad der Nebenläufigkeit wird in S-Systemen durch die Anfangsmarkierung bestimmt, da vorhandene Marken in dieser Netzklasse nicht aus dem System entfernt und neue Marken nicht generiert werden können.

$N_S = (S, T; F, K, W, M_0)$ mit

$S = \{s_1, s_2, s_3\}$

$T = \{t_1, \ldots, t_5\}$

$F = \{(s_1, t_1), (t_1, s_3), \ldots, (t_5, s_2)\}$

$K: S \to \{\infty\}$

$W: F \to \{1\}$

$M_0: \{s_1, s_2, s_3\} \to (1, 0, 1)$

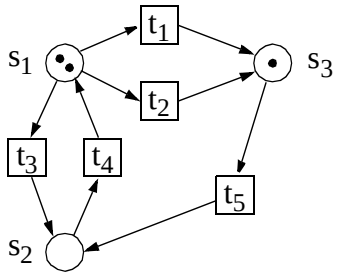


Abbildung 2.2: Beispiel eines (lebendigen) S-Systems

Systeme in denen gezielt Nebenläufigkeit oder Verteiltheit modelliert werden soll, kommen ohne Gabeltransitionen, welche Aufgaben verteilen oder Synchronisationstransitionen, welche Ergebnisse zusammenführen nicht aus.

### 2.3.4 T-Systeme

Lockert man die Einschränkungen, die S-Systeme bei der Kardinalität der Vor- bzw. Nachbereiche von Transitionen definieren, und führt sie bei Stellen ein ($\forall s \in S: |\cdot s| = |s \cdot| = 1$), so wechselt man in die Klasse der Synchronisationsgraphen

(*T-Systeme*). Ihr Strukturmerkmal sind Zyklen (Abbildung 2.3), auf denen die Anzahl der Marken konstant bleiben. Teile eines Zyklus sind unsynchronisiert d.h. das Schalten von Transitionen erfordert keinen Gleichlauf mit einem anderen Zyklus. Betrachtet man die Wegstrecke, die zwei oder mehrere Zyklen gemeinsam ist, so werden zu Beginn die Zyklen synchronisiert, auf der Wegstrecke zeitlich übereinstimmend abgearbeitet und am Ende wieder verzweigt. Beachtet man bei der Modellierung mit T-Systemen, daß jede Stelle auf einem markierten Zyklus liegt, so ist das Netz lebendig und beschränkt (mit k gleich der maximalen Anzahl von Marken, die durch die Startmarkierung auf einem Zyklus verteilt wurden).

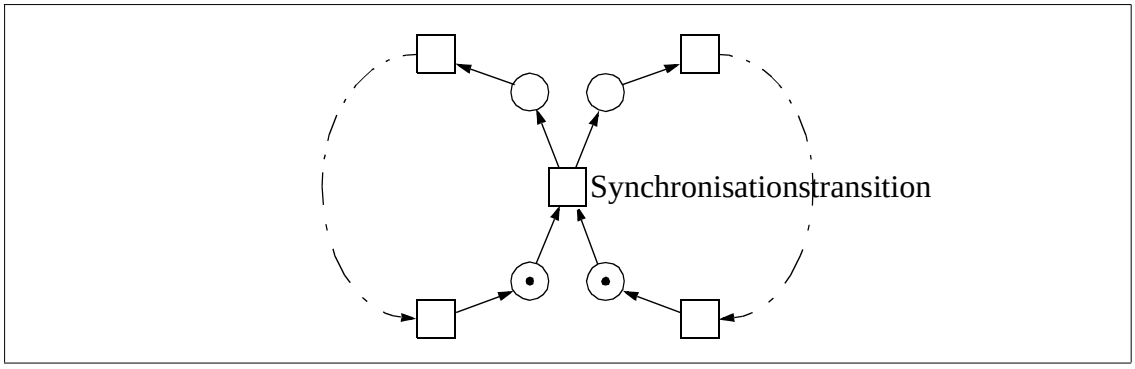


Abbildung 2.3: Stilisierte Darstellung eines T-Systems mit 2 Zyklen

In T-Systemen ist sichergestellt, daß zwei Transitionen nie zueinander in Konflikt stehen können. Somit sind die Abläufe von Systemen, die in dieser Klasse modelliert werden deterministisch. Beinhaltet das Systemdesign jedoch notwendigerweise Nichtdeterminismus, so ist diese Petrinetzklasse zur Modellierung nicht geeignet.

### 2.3.5 Free-Choice-Netze

Setzt man Konflikte im Systemdesign gezielt ein, so ist zu untersuchen, auf welche Weise sie gelöst werden: dies kann einerseits lokal erfolgen, andererseits von einem größeren Kontext abhängen. Eine Netzklasse, die nur lokale Abhängigkeiten bei Konflikten zuläßt, sind *Free-Choice-Netze*. Zum Verständnis muß man unterscheiden zwischen einem Konflikt, der aus der Netztopologie heraus vollständig beschrieben werden kann und der, dessen Eintreten zusätzlich von der Dynamik abhängt („Konfusion“). In Free-Choice-Netzen gilt folgende Einschränkung der Vernetzung: $\forall s \in S$ und $\forall t, t' \in T$ mit $t \neq t' : t, t' \in s\cdot \Rightarrow \cdot t = t'\cdot$. Hierdurch wird sichergestellt, daß bei der Auflösung des Konfliktes keinen dritten, unbeteiligten Transitionen eine Schiedsrichterfunktion zukommt. Zwar kann in dieser Netzklasse Nebenläufigkeit und Nichtdeterminismus ausgedrückt werden, jedoch besteht keine Möglichkeit, eine bedingte Verzweigung zu modellieren (Abbildung 2.4).

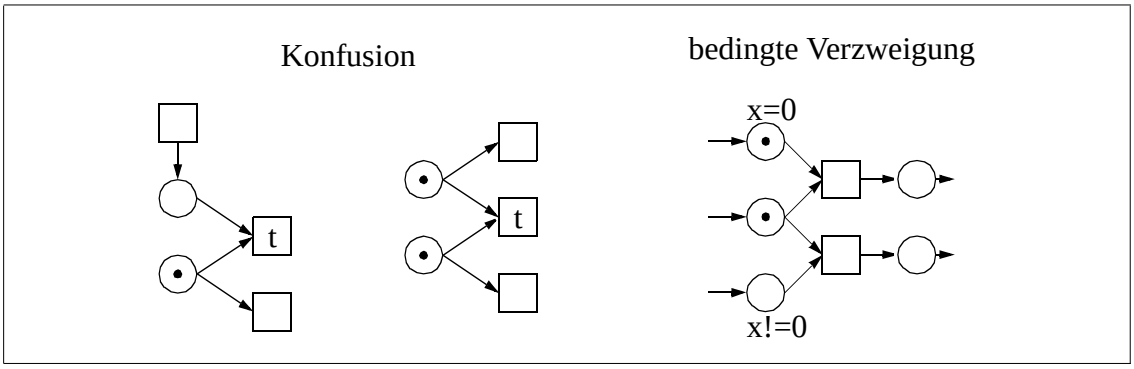


Abbildung 2.4: Unzulässige Vernetzungsarten in Free-Choice-Netzen

*Konfusion*: ob t schalten wird hängt von der Schaltreihenfolge anderer aktivierter Transitionen ab. *Bedingte Verzweigung*: verbietet man Konfusion, so sind solche Vernetzungen unzulässig.

In den bisher vorgestellten Netzklassen (S-Systeme, T-Systeme oder Free-Choice-Netze) mußten für Aussagen über Eigenschaften von Systementwürfen Beschränkungen in der Ausdrucksmächtigkeit akzeptiert werden (keine Nebenläufigkeit, kein Nichtdeterminismus oder keine bedingten Verzweigungen).

In der Netzklasse, die keine solche Beschränkungen aufweist (S/T-Netze), wird der dynamische Zustand eines modellierten Systems vollständig durch die Anzahl und die Verteilung von Marken auf Stellen beschrieben. Mit diesen Netzen ist es möglich, Systeme sowohl aus abstrakter Sicht als auch im Detail zu modellieren, kausale Abhängigkeiten zwischen Transitionen darzustellen und Nebenläufigkeit, Synchronisation und Nichtdeterminismus auszudrücken. Es besteht aber nicht die Möglichkeit einzelne Systemobjekte (Marken) beim Fluß durch die Netzstruktur zu unterscheiden und das Schaltverhalten von Transitionen von Attributen dieser Systemobjekte abhängen zu lassen.

Erfordert jedoch die Komplexität eines Systems eine solche Modellierung, so stehen hierfür Netzklassen mit Individuen als Marken zur Verfügung (vgl. [Rei86] Seite 130-163 und [Sho 97]). Hierzu zählen Prädikat-Transitionen-Netze (Pr/T-Netze) und gefärbte Petrinetze (CP-Netze). Prädikat-Transitionen-Netze bieten den Vorteil eines Variablenkonzepts, haben jedoch den Nachteil, daß keine Analysetechniken verfügbar sind, die Eigenschaften des Netzentwurfs überprüfen können. Für gefärbte Petrinetze ist hierfür ein Invariantenkalkül bekannt. Bei der Unterscheidung von Individuenmarken durch Farben sind jedoch dem Entwurf Grenzen der intuitiven Verständlichkeit gesetzt.

### 2.3.6 Prädikat-Transitions-Netze

Im folgenden werden beide Netzklassen detailliert eingeführt, da ihre Definition in späteren Kapiteln verwendet wird.

**Definition 2.2** *Prädikat-Transitionen-Netze*

Pr/T-Netze besitzen als herausragendes Merkmal unter den Netzklassen ein Variablenkonzept, welches über eine Signatur $\Sigma$ definiert wird. Das Schaltverhalten von Transitionen wird in Termen einer $\Sigma$-Algebra beschrieben:

1. $\Sigma = (\hat{S}, \Phi)$ sei eine *Signatur* über die endliche Menge der Sorten $\hat{S}$ und die Menge der Funktionssymbole $\Phi$. Jedes Funktionssymbol $\varphi \in \Phi$, $\varphi: (\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{n_\varphi}) \to \hat{s}$ habe eine Typisierung in $S$, wobei 0-stellige Funktionssymbole als Konstanten interpretiert werden. Mit $X = \bigcup_{\hat{s} \in \hat{S}} X_{\hat{s}}$ sei die Menge der typgebundenen Variablen, mit $Term(\Phi)$ sei wie üblich die Menge der Terme über $\Sigma$ definiert.

2. Die $\Sigma$-Algebra $A = (\hat{D}, \hat{F})$ sei ein Modell für die Signatur $\Sigma$, wobei $\hat{D} = \bigcup_{\hat{s} \in \hat{S}} D_{\hat{s}}$ die Grundmengen der Typen und $F$ die Interpretation der Funktionssymbole aus $\Phi$ darstellt. Sei $\beta: X \to D$ eine typerhaltende Belegung der Variablen, welche induktiv über den Aufbau der Terme fortgesetzt wird ($\beta: Term(\Phi) \to D$).

3. Ein Pr/T-Netz ist ein Tupel $N_{Pr/T} = (S, T; F, \Sigma, \sigma, A, M_0)$ so, daß gilt:

   - $\sigma$ ist eine Beschriftung von Elementen von $T \cup F$, wobei

     - Transitionen $t \in T$ mit einer $\Sigma$-Formel (Wächter) $\sigma(t) = \tau_t$ beschriftet werden können, die als boolesche Formel interpretiert wird ($\beta(\tau_t) \in \mathbb{B}$) und

     - jede Kante $(v, w) \in F$ mit einer endlichen Menge $\sigma(v, w) = \{\tau_1, \ldots, \tau_m\}$ von Termen $\tau_i \in Term(\Phi)$ beschriftet ist.

   - A ist ein Modell für die Signatur $\Sigma$

   - $M_0: S \to 2^{\hat{D}}$ ist die Startmarkierung, die jeder Stelle aus $S$ eine Menge von Elementen aus den Grundmengen der Typen zuweist.

(Oft wird in der Definition von Pr/T-Netzen zusätzlich für jede Stelle eine Menge von Sorten eingeführt und von jeder Kante verlangt, die in oder aus einer Stelle zeigt, daß sie mit je einem Term für jede Sorte der Stelle beschriftet ist. Von Markierungen des Netzes wird dann Sortentreue gefordert.)

Führt man für die Kantenbeschriftung $\sigma(v, w) = \{\tau_1, \ldots, \tau_m\}$ und Markierungen $M(s) = \{m_1, \ldots, m_r\}$ die symbolische Summenschreibweise $\sigma(v, w) = \tau_1 + \ldots + \tau_m$ respektive $M(s) = m_1 + \ldots + m_r$ ein, so kann man das Schaltverhalten eines Pr/T-Netzes wie folgt definieren:

**Definition 2.3** *Schaltregel für Prädikat-Transitionen-Netze*

Sei $N_{Pr/T} = (S, T; F, \Sigma, \sigma, A, M_0)$ ein Pr/T-Netz. Die Schaltregel wird über die Anwendbarkeit einer Belegung, die Aktiviertheit einer Transition und die Folgemarkierung nach dem Schalten der Transition definiert:

1. Eine Belegung $\beta: Term(\Phi) \to \hat{D}$ heißt auf eine Transition *t anwendbar*, wenn keine Wächterbedingung $\tau_t$ für *t* angegeben wurde, oder $\beta(\tau_t)$ wahr ist.
2. Sei der Schaltmodus einer Transition *t* definiert durch die Belegung $\beta$, kurz $t[\beta]$. Eine Transition ist im Schaltmodus $t[\beta]$ genau dann aktiviert (*t* ist $\beta$-aktiviert), wenn gilt
   - $\beta(\tau_t) \subseteq M(s)$ für alle $s \in {\cdot}t$ und $\tau \in \sigma(s, t)$,
   - $\beta(\tau_t) \cap M(s) = \varnothing$ für alle $s \in t{\cdot}$ und $\tau \in \sigma(t, s)$.
3. Ist *t* unter der Markierung *M* $\beta$-aktiviert und schaltet *t* im Modus $t[\beta]$, so ergibt sich folgende Markierung (vgl. Abbildung 2.5)

$$M'(s) = \begin{cases} M(s) & \text{für } s \notin {\cdot}t \cup t{\cdot} \\ M(s) - \beta(\sigma(s, t)) & \text{für } s \in {\cdot}t - t{\cdot} \\ M(s) + \beta(\sigma(t, s)) & \text{für } s \in t{\cdot} - {\cdot}t \\ M(s) - \beta(\sigma(s, t)) + \beta(\sigma(t, s)) & \text{für } s \in {\cdot}t \cap t{\cdot} \end{cases} \tag{2.1}$$

Die Schreibweise $M[t[\beta]>M'$ bezeichne, daß die Transition *t* im Modus $t[\beta]$ aktiviert ist und durch das Schalten von *t* die Markierung *M* in die Markierung $M'$ gemäß (2.1) überführt wird.

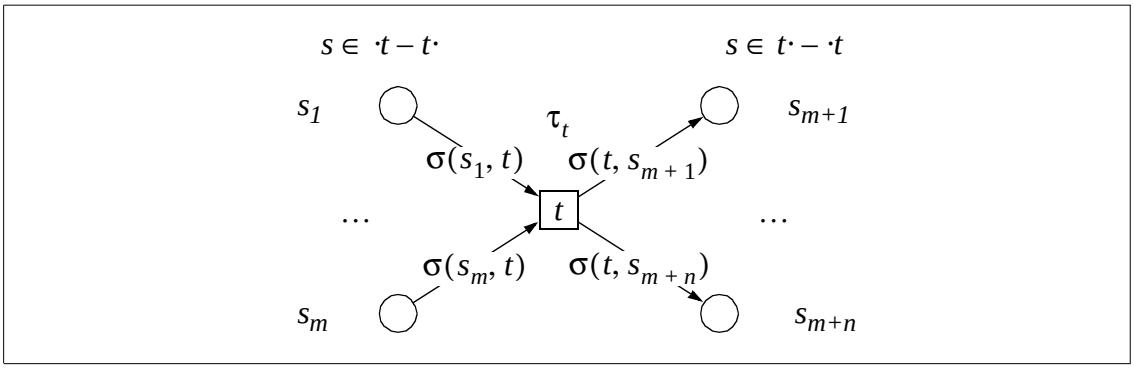


Abbildung 2.5: Beschriftungen der Schaltbedingung einer Transition (Pr/T-Netz)

Sind die Anforderungen, die eine Modellierung an die Unterscheidung von individuellen Marken stellt, geringer, als die Mächtigkeit der Pr/T-Netze mit ihrem Variablenkonzept, so reichen gefärbte Petrinetze (CP-Netzen) meist aus.

In Pr/T-Netzen sind Marken von einem bestimmten Datentyp. Auf einer Stelle befinden sich nie zwei Marken eines Typs mit demselben Wert (strikte Pr/T-Netze). In CP-Netzen hingegen, besitzen Marken als Klassifikationsmerkmal eine Farbe. Marken gleicher Farbe können mehrfach auf einer Stelle vorhanden sein, und sind voneinander nicht zu unterscheiden.

### 2.3.7 Gefärbte Netze

In Pr/T-Netzen werden Schaltmodi $t[\beta]$ eingeführt, die in Abhängigkeit von der Belegung der freien Variablen das Schaltverhalten einer Transition definieren. In CP-Netzen werden für Transitionen ebenfalls Schaltmodi in Form von Farben eingeführt, in denen eine Transition schalten kann. A priori besteht zwischen der Farbe eines Schaltmodus und der Farbe von Marken kein Zusammenhang, jedoch wird für beide Konzepte die selbe Farbenmenge benutzt.

**Definition 2.4** *gefärbte Petrinetze*

Ein CP-Netz $N_{CP} = (S, T; F, C, W, M_0)$ wird über die Menge der Farben $F$ wie folgt definiert:

1. Die Abbildung $C$ ordne jeder Stelle oder Transition eine nicht leere Menge von Farben zu ($C$: $S \cup T \rightarrow 2^F - \{\emptyset\}$).
2. Die Abbildung $W$ ordne jeder Kante ($\{v, w\} = \{s, t\}$, $(v, w) \in F$) eine Menge von Abbildungen wie folgt zu: zu jeder Farbe $f$ von $t$ (zu jedem Schaltmodus $t[f]$) wird für jede Farbe von $s$ die Anzahl der Marken angegeben, die über diese Kante abgezogen bzw. erzeugt werden ($W(v, w) = C(t) \rightarrow (C(s) \rightarrow \mathbb{N})$).
3. wobei die Startmarkierung als eine Abbildung definiert ist, die für jede Farbe einer Stelle die Anzahl der Marken angibt, die auf dieser Stelle liegen ($M_0(s) = C(s) \rightarrow \mathbb{N}$).

Die Menge der Stellen $S = \{s_1, \ldots, s_n\}$, der Transitionen $T = \{t_1, \ldots, t_m\}$ und die Menge der Farben $F = \{f_1, \ldots, f_l\}$ haben eine beliebige, aber feste Ordnung. Dann kann man die Markierung einer Stelle als einen $|C(s)|$-stelligen Vektor in $\mathbb{N}$ darstellen, sowie jede Beschriftung einer Kante $(s, t)$ oder $(t, s)$ als eine $|C(s)| \times |C(t)|$-Matrix in $\mathbb{N}$ darstellen, wobei ein Element die Anzahl der Marken zu einer bestimmten Farbe angibt (vgl. Abbildung 2.6).

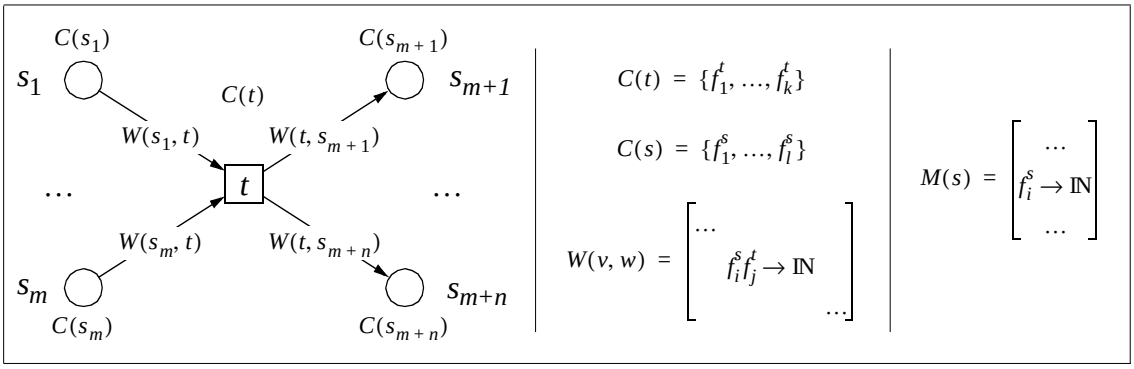


Abbildung 2.6: Beschriftungen der Schaltbedingung einer Transition (CP-Netz)

**Definition 2.5** *Schaltregel für CP-Netze*

Sei $N_{CP} = (S, T; F, C, W, M_0)$ ein CP-Netz.

1. Eine Transition ist unter der Farbe $f$ (Schaltmodus $t[f]$) genau dann aktiviert ($t$ ist $f$-aktiviert), wenn für jede Kante $(s, t)$ mit $s \in {\cdot}t$ gilt, daß $W(s, t)(f) \leq M(s)$.
2. Ist $t$ unter der Markierung $M$ $f$-aktiviert und schaltet im Modus $t[f]$, so ergibt sich folgende Markierung

$$M'(s) = \begin{cases} M(s) & \text{für } s \notin {\cdot}t \cup t{\cdot} \\ M(s) - W(s, t)(f) & \text{für } s \in {\cdot}t - t{\cdot} \\ M(s) + W(t, s)(f) & \text{für } s \in t{\cdot} - {\cdot}t \\ M(s) - W(s, t)(f) + W(t, s)(f) & \text{für } s \in {\cdot}t \cap t{\cdot} \end{cases} \tag{2.2}$$

Die Schreibweise $M[t[f]>M'$ bezeichne, daß die Transition $t$ im Modus $t[f]$ aktiviert ist und durch das Schalten von $t$ die Markierung $M$ in die Markierung $M'$ gemäß (2.2) überführt wird.

### 2.3.8 Zeitbehaftete Netze

Die bisher vorgestellten Netzklassen können zwar Nebenläufigkeit, Nichtdeterminismus, bedingte Verzweigungen und Individuen abbilden, jedoch sind in ihnen keine Aussagen über das zeitliche Verhalten eines modellierten Systems möglich. In verteilten Systemen ist es nicht möglich, einen globalen Zeittakt zu definieren, so daß z.B. zwei räumlich entfernte Aktionen synchronisiert stattfinden könnten. Aufgrund von technischen Einrichtungen, wie z.B. dem in Braunschweig abgestrahlten Zeitsignal, ist es bei Kenntnis der

Übertragungszeit fortlaufend möglich die lokale Uhr hinreichend genau an die globale Uhr zu adaptieren. In Kenntnis der Hardware-Anforderungen, die hierdurch an reale Systeme gestellt werden, kann man nun den Zeitbegriff in Petrinetzmodelle einführen ([Sta 90] S. 192-221). Es gibt verschiedene Ansätze, Petrinetze mit Zeit zu attributieren, die für unterschiedliche Modellierungen genutzt werden:

1. Stellen können mit einer Verweilzeit attributiert werden: Marken, die über eine eingehende Kante auf die Stelle abgelegt werden, aktivieren erst nach Verstreichen der Verweilzeit eine Transition im Nachbereich.
2. Marken können mit einem individuellen Zeitstempel arttributiert werden: Der Zeitstempel gibt an, ab welchem Zeitpunkt eine Marke vom Zustand “nicht verfügbar” in den Zustand “verfügbar” wechselt und dann erst Transitionen aktivieren kann.
3. Kanten können mit einer Verzögerungszeit attributiert werden: Marken, die entlang dieser Kanten wandern, werden entsprechend dem Zeitwert verzögert. Eine Transition feuert erst dann, wenn alle Marken, die sie in ihrem Schaltmodus aktiviert haben, sie über die eingehenden Kanten erreicht haben.
4. Transitionen können mit einer Schaltdauer attributiert werden:

   Beim 3-Phasen Schalten werden im ersten Schritt Marken, die eine Transition in einem Schaltmodus aktiviert haben, von den Stellen im Vorbereich abgezogen, danach schaltet die Transition mit der angegebenen Dauer und im letzten Schritt werden Marken auf Stellen im Nachbereich abgelegt.

   Beim atomaren Schalten, verbleiben die Marken während der angegebenen Dauer auf den Stellen im Vorbereich der aktivierten Transition. Es wird ein Timer gestartet, nach dessen Ablauf die Transition in einem Schritt schaltet. Bei diesem Petrinetzmodell ist anzugeben, ob die Marken beim Starten des Timers als reserviert gekennzeichnet werden, oder ob es zwischen zwei in Konflikt stehenden Transitionen zu einem Wettlauf um die Marken kommen kann, den diejenige gewinnt, deren Timer als erstes abgelaufen ist.
5. Transitionen können mit einer exponentialverteilten Schaltverzögerung attributiert werden: Jede Transition $t$ wird durch mit $\lambda_t$ der Verteilungsrate für die Schaltverzögerung attributiert. Sei $t$ zum Zeitpunkt $x = 0$ aktiviert, so wird die Wahrscheinlichkeit, daß $t$ bis zum Zeitpunkt $x > 0$ schaltet, durch die Verteilungsfunktion $F(x) = 1 - e^{-\lambda_t x}$ beschrieben.

# Kapitel 3

# Interaktionsmodell

Gegenstand des vorliegenden Kapitels ist die Einführung eines allgemeinen Modells verteilter Interaktionsmechanismen.

Im Rahmen dieses Kapitels werden zunächst die Anforderungen zusammengetragen, welche an ein allgemeines Modell verteilter Interaktionsmechanismen gestellt werden. Im folgenden Abschnitt werden die grundlegenden Lösungskonzepte vorgestellt. In den nachfolgenden Abschnitten werden dann die einzelnen Konzepte inkrementell zu einem umfassenden formalen Modell zusammengefügt. Zur besseren Verständlichkeit werden verschiedene Versionen des Modells definiert, um dem Leser den Zugang zu dem komplexen Gesamtmodell zu erleichtern. Zugleich wird dabei ersichtlich, welche Modellelemente aufgrund der Verwendung bestimmter Konzepte hinzukommen und wie sich daraus das Zusammenspiel der einzelnen Komponenten ergibt. Ein Beispiel, welches jeweils am Ende einer Iterationsstufe besprochen wird, soll die Verwendung der formal definierten Modellelemente verdeutlichen. Im letzten Abschnitt (Abschnitt 3.5.7) wird zusammengefaßt, wie die Anforderungen aus Abschnitt 3.1 durch das Modell abgedeckt werden.

Die durchgängige Formalisierung in den Definitionen ermöglicht gegenüber einer reinen Prosaform, eine prägnantere und damit nachvollziehbarere Darstellung des Modells. Dennoch wird versucht, die formalen Ausdrücke weitgehend mit verbalen Beschreibungen anzureichern, so daß der Leser einen konstanten Lesefluß beibehalten kann. An den Stellen, wo dies nicht geglückt ist, möge man es dem Autor nachsehen, wenn er der Kompaktheit einer Aussage den Vorzug gegenüber einer wortreichen Darstellung gegeben hat.

Dem eiligen Leser sei der Abschnitt 3.2 über grundlegende Lösungskonzepte zur Lektüre nahegelegt. Zur Motivation können die Abschnitte über das Auktionsbeispiel (Abschnitt 3.3.5, Abschnitt 3.4.5 und Abschnitt 3.5.6) vorgezogen werden.

## 3.1 Anforderungen an eine Entwurfssprache für MAS

Der Entwurf von verteilten Anwendungen und insbesondere von Multiagentensystemen (MAS) ist gekennzeichnet durch eine hohe Komplexität der Softwarearchitektur. Einzusetzende Entwurfskonzepte und Entwurfsmethoden unterscheiden sich in den Abstraktionsmöglichkeiten, welche es erlauben, eine Systemkomponente zu beschreiben [Bal 96] (S. 643).

Das einer konkreten Softwarearchitektur zugrundeliegende Architekturkonzept läßt sich in ein *Operationsprinzip* und eine dem Operationsprinzip genügende *Struktur* unterteilen. Das Operationsprinzip in Multiagentensystemen ist definiert durch die *Autonomie* der Agenten. Dabei definiert [Lev 96] Autonomie als ein dem Agenten übertragenes *Recht,* wobei entsprechende *Selbständigkeit* in seinen Aufgabenbereichen von ihm gefordert wird. Selbständigkeit bedeutet nach [Lev 96]:

- eigenständige *Entscheidungsfähigkeit* durch *Wahrnehmen, Überwachen, Planen, Schlußfolgern* und *Abschätzen der Konsequenzen* seines Handelns;
- aufgabenbezogenes Handeln entsprechend den *Zielvorgaben*, welche durch eine *Kombination aus Planungs- und Überwachungsschritten* gewährleistet werden;
- *Lernen* und Beseitigen von Störungen;
- *Fähigkeit zur Kooperation* als eine Fähigkeit *Zielvorgaben abzustimmen* und eigene Pläne mit denen anderer zu *koordinieren,* um *gemeinsame Ziele zu erreichen.*

Der Begriff der *Autonomie* wird aufbauend auf dem Begriff der Selbständigkeit definiert, indem einem Agenten eineseits das *Recht* gegeben wird, selbständig zu agieren und andererseits indem dieses Recht durch *Vorgaben, Regelungen* und *Normen* geschützt und damitdie *Unabhängigkeit* des Agenten gewährleistet wird. Erst wenn dies erfüllt ist, kann man von der *Autonomie* eines Agenten sprechen.

Die dem Architekturkonzept zugrundeliegende Struktur ist durch die aus der Selbständigkeit der autonomen Agenten heraus induzierte Interaktion zwischen den Agenten gegeben. Die Struktur kann bei gleichbleibenden Aufgaben fest sein, oder in einem dynamischen Umfeld sich selbstorganisatorisch umbilden.

Im folgenden betrachten wir Multiagentensysteme aus Sicht der Disziplin des Softwareengineerings und gelangen ausgehend von dem Entwurfsprozeß einer Softwarearchitektur zu den Anforderungen an eine Entwurfssprache für Multiagentensysteme. Das Ergebnis des Entwurfs einer Softwarearchitektur ist nach [Bal 96] S. 642:

- Zerlegung des definierten Systems in *Systemkomponenten* (oder Subsysteme, die dann wieder zu Systemelementen zerlegt werden)
- Strukturierung des Systems durch geeignete *Anordnung* der Systemkomponenten

- Beschreibung der *Beziehungen* zwischen den Systemkomponenten

Soll eine Anwendung in Form eines Multiagentensystems entworfen werden, so erfolgt die Zerlegung des Systems in *autonome Agenten* als Subsysteme der Anwendung. Multiagentensysteme sind häufig als offene Systeme konzipiert, das heißt zur Laufzeit können neue Subsysteme hinzukommen, oder bestehende entfernt werden. Charakteristisch für die Strukturierung eines Multiagentensystems ist, daß die Anordnung der Subsysteme im Allgemeinen nicht zum Entwurfszeitpunkt vorgegeben wird. Beziehungen zwischen den Agenten werden dynamisch etabliert und von den Agenten autonom verwaltet. Dies umfaßt bei einer unreglementierten Umsetzung der Autonomie auch, daß eine Beziehung einseitig von einem Partner abgebrochen werden kann, sobald dieser aufgrund interner Bewertungsfunktionen keinen Nutzen mehr für sich in der Fortsetzung der Beziehung sieht.

Agenten etablieren Beziehungen zu anderen Agenten, um mit ihnen zu interagieren. Der Bedarf an einer Interaktion entsteht z. B. dadurch, daß entweder ein Konflikt mit einem anderen Agenten erkannt wurde, welcher über eine Koordination als eine Form der Interaktion behoben werden soll. Oder ein Agent stellt fest, daß er nicht in der Lage ist, ein eigenes Ziel in ausreichender Qualität (z. B. Zeit) umzusetzen, jedoch durch Kooperation mit anderen Agenten, welche verwandte Ziele verfolgen, die erwünschte Qualität doch erreichen könnte. Der Nutzen, den ein anderer Agent durch kooperatives Verhalten erlangt, muß für egoistische Agenten in einer Verringerung des Aufwandes (z. B. Kosten) für das Erreichen der *eigenen* Ziele zu messen sein, oder für altruistische Agenten in einer Verringerung des Aufwandes für das Erreichen der Ziele *aller* beteiligten Agenten. Grundsätzlich steht es den Agenten frei, wann und mit wem sie Interaktionsbeziehungen etablieren.

In Anwendungsgebieten, welche eine hohe Interaktion der Systemkomponenten erfordern, wie das z. B. in Robotersystemen der Fall ist, wird häufig die Autonomie in der Ausgestaltung der Beziehungen durch den Systementwurf reglementiert. So wird den Agenten als Ziel vorgegeben, daß sie sich bis zu einer gewissen Grenze altruistisch verhalten. So wäre es für das erwünschte Systemverhalten eines Roboters hinderlich, wenn z. B. ein bildgebender Sensoragent keine Bereitschaft hätte, mit anderen bildinterpretierenden Agenten in eine Interaktion zu treten.

Die Anordnung der Agenten zu einem funktionstüchtigen System wird also durch die Beziehungen zwischen den Agenten bestimmt, über welche die Agenten untereinander interagieren. Somit ist ein Kernelement bei dem Entwurf eines Multiagentensystems die Spezifikation der Beziehungen, welche das Agentensystem aufspannen. Diese sind geeignet zu formalisieren und durch eine exakte Beschreibungssprache und einer klar definierten Semantik zu unterstützen.

Aus Sicht des Systementwurfs wird eine geeignete Abstraktion gefordert, auf der die Menge der das Systemziel erreichenden, verteilten Algorithmen beschrieben werden können. Diese Klasse von Algorithmen ordnet die Agenten an und weist ihnen Zuständigkeiten zu. Die Abstraktion sollte so gewählt sein, daß zur Ausgestaltung der Zuständigkeiten die Agenten ihre Selbständigkeit nutzen können.

Weiterhin fordern wir, Interaktionsbeziehungen einem im System einheitlichen, normativen Reglement unterwerfen zu können. Somit sollte z. B. die Selbständigkeit der Agenten hinsichtlich der Ausgestaltung einer Beziehung dahingehend beschränkt werden können, daß ihnen der Abbruch einer Beziehung zu gewissen Zeiten untersagt werden kann. Das Reglement soll im Allgemeinen für jede Beziehungsart weiter detaillierbar sein. Unter alternativen Reglements soll ausgewählt werden können, um den spezifischen Gegebenheiten der aktuellen Situation Rechnung zu tragen.

Aus der Sicht des Entwurfs einzelner Agenten ist der Informationsgrad hinsichtlich einer Beziehung von Bedeutung. Greifen die Aufgaben der einzelnen Agenten ineinander, so wird eine Synchronisation erforderlich. Durch geeignete Beschreibungskonstrukte der Abstraktion sollen die einer Synchronisation unterliegenden Schritte einer Beziehung ausdrückbar sein.

Im Folgenden präzisieren wir die aufgestellten Anforderungen an eine Entwurfssprache für Multiagentensysteme:

**Anforderung I (Abstraktion)**
Zur Beschreibung von Interaktionsbeziehungen soll von konkreten Instanzen und Individuen abstrahiert werden. Die Beschreibung hat das Reglement zu definieren, an welches sich alle Interaktionsteilnehmer zu halten haben, sowie eine entsprechende Semantik, welche dies zusichert. Davon abgesehen wird das Verhalten eines Teilnehmers durch seine Funktion und Autonomie bestimmt.

**Anforderung II (Adaptivität und Wiederverwendung)**
Eine Interaktionsbeziehung soll einem Schema folgen, welches zur Laufzeit aus einer Menge von alternativen Schemata durch die beteiligten Interaktionspartner ausgewählt und dann instantiiert wird.

**Anforderung III (Wandelbarkeit)**
Die Menge der zur Auswahl stehenden Interaktionsschemata soll sich zur Laufzeit ändern können.

**Anforderung IV (Informiertheit)**
Die Interaktionspartner sollen Wissen über den Zustand der Interaktionsbeziehung besitzen. Der Grad des Wissens soll hinsichtlich dem zur Verfügung stehenden Netzwerkdienst maximal sein. Abhängig von der Interaktionsbeziehungen kann es ausreichend sein, wenn dies nur für Teilzustände der Beziehung zugesichert wird.

**Anforderung V (Robustheit)**
Die Architektur darf keine zentrale Komponente besitzen, deren Ausfall laufende Interaktionen blockieren würde.

**Anforderung VI (Offenheit)**
Offene Interaktionsbeziehungen, in denen weitere Interaktionspartner aufgenommen oder aus denen beteiligte Interaktionspartner entlassen werden, sollen unterstützt werden.

# 3.2 Grundlegende Konzepte

## 3.2.1 Agentenkonzept

Der Begriff des Agenten ist in der Literatur nicht durch eine allgemein akzeptierte Definition versehen. Der Grund hierfür ist, daß verschiedene Anwendungsbereiche unterschiedliche Schwerpunkte auf einzelne Eigenschaften des Agenten setzen, wobei der Begriff der Autonomie eine zentrale Bedeutung für das Agentenkonzept besitzt (Abschnitt 3.1). Jedoch ist auch dieser nicht einheitlich definiert. [Woo 99] definiert den Agenten wie folgt:

> An *agent* is a computer system that is *situated* in some *environment*, and that is capable of *autonomous action* in this environment in order to meet its design objectives. (S. 29, Zeile 6-8)

Agenten besitzen eine interne Struktur, die sich von der anderer Agenten unterscheiden kann. Um dennoch über die interne Struktur eines Agenten sprechen zu können, eignet sich eine einheitliche abstrakte Sicht auf diese Struktur.

Agenten agieren in einer physischen und informationstechnischen Umwelt, zu der sie selbst gehören. Agenten grenzen sich untereinander ab,

- durch ihren *internen Bereich*, welcher die “mentalen Zustände” eines Agenten entsprechend seiner internen Struktur[1] umfaßt;
- durch ihre *Zuständigkeit* im System, also die Befugnis zur Ausübung einer bestimmten Tätigkeit und zur Wahrnehmung eines damit verbundenen Aufgabenkreises. Die Ausübung erfolgt dabei grundsätzlich nach eigenem, autonomen Ermessen des Agenten. Eine wesentliche Zuständigkeit jedes einzelnen Agenten ist die Ausgestaltung der Koordination (Konflikterkennung und Konfliktbehebung) und der Kooperation (gegenseitiges Unterstützen) mit anderen Agenten nach einem im System einheitlichen Reglement.
- durch den *Autonomiebereich*, innerhalb dessen sie agieren. Dieser läßt sich unterteilen in einen *Eigentumsbereich*, in dem der Agent volle Nutzungs- und Verfügungsgewalt besitzt und in einem übertragenen Bereich, dem *Zuständigkeitsbereich*, in dem der Agent entsprechend seiner Zuständigkeit Rechte und (Handlungs-) Pflichten besitzt.

1. Die mentalen Zustände sind in *deliberativen* (planenden) Agentenarchitekturen explizit modelliert, wohingegen sie in *verhaltensbasierten* Agentenarchitekturen implizit in den Reiz-Antwort-Schemata und der sie steuernden Entscheidungskomponente identifizierbar sind. Die im Folgenden eingeführten Begriffe sollen der abstrakten Beschreibung eines Agenten dienen, jedoch keine Vorwegnahme des internen Aufbaus darstellen.

Der Agent gewinnt seine Identität aus dem internen Bereich, insbesondere

- aus seinem *Weltmodell*, als die Beschreibung seines Verständnisses von seiner Umwelt. Das Weltmodell kann unterteilt werden in einen Wissensbereich und in einen Hypothesenbereich.
- aus seinen *Zielen*, welches in Wünsche, Ziele und Absichten unterteilt werden kann.
- aus seinen *Plänen*, welche seine Fähigkeiten bestimmen. Elementare Bausteine der Pläne sind Aktionen mit denen der Agent auf sich und seine Umwelt wirken kann.

Agenten folgen ihren Zielen, sind dabei jedoch an ihr Weltmodell und insbesondere an die Restriktionen aus dem Zuständigkeitsbereich gebunden. Grundsätzlich gehören sie zu der Klasse der geregelten Systeme, die über eine Wahrnehmungskomponente eine dynamische Situation überwachen und bei Abweichungen von der Zielsituation durch geeignet geplante Aktionen auf ihre Umwelt einwirken.

Agenten, welche Teil eines Multiagentensystems sind, benötigen die Fähigkeit zur Interaktion und Koordination mit anderen Agenten. Agenten besitzen eine gemeinsame Sprache (oder sind in der Lage eine solche sich anzueignen) und kommunizieren untereinander in dieser Sprache (Abschnitt 3.2.2).

### 3.2.2 Konzept eines Multiagentensystems

Ein System, welches aus mindestens drei[1] interagierenden Agenten besteht, wird als Multiagentensystem bezeichnet. Während Systeme mit einer geringen Anzahl von Agenten weitgehend beherrscht werden, sind große, aus heterogenen Agenten bestehende Multiagentensysteme und deren Strukturierung noch ein weitgehend offenes Problem [Les 98].

Dieses Problem ist eng damit verbunden, geeignete Koordinationsmechnismen zu entwickeln, welche auf unterschiedlichen Granularitätsstufen Ausschnitte des Systems steuern. In der gröbsten Granularitätsstufe, also auf der obersten Abstraktionsebene des Systems, wird das Treffen von Entscheidungen mit systemweiten Auswirkungen koordiniert.

[Les 98] schlägt für große, heterogene Multiagentensysteme eine Einteilung in drei Schichten vor: der *Organisationsschicht* (agent organization layer), der *Koordinationsschicht* für kleine Agentengruppen (small agent group coordination layer) und der *Ter-*

1. Auf der MAAMAW ‘96 [BoVa 96] wurde diskutiert, ab wann ein System ein Multiagentensystem ist: Ein aus zwei Agenten bestehendes System wird danach noch nicht als Multiagentensystem angesehen, da durch sie nur ein Dialog, nicht jedoch eine multilaterale Verhandlung modelliert werden kann. Bei einem System, in welchem die Umwelt in Form eines Agenten modelliert ist, reichen zwei weitere Agenten aus, um es als Multiagentensystem zu klassifizieren.

*min- und Ressourcenplanungsschicht* der einzelnen Agenten (local agent scheduling layer), wobei die höheren Schichten den tieferen Schichten eine in Form von Restriktionen beschriebene Strategie (policy) vorgeben, welche dann ihre Entscheidungen hinsichtlich ihrer Steuerungsaufgabe darauf abstimmen. Umgekehrt stellen tiefere Schichten den höheren Information bereit, die dazu führt, daß Restriktionen abgeändert werden, falls sie nicht erreichbar sind oder zu unerwünschtem Verhalten führen würden.

Die Notwendigkeit zur Bildung von sich koordinierenden Gruppen liegt einerseits in der arbeitsteiligen Funktionalität der Agenten und andererseits in der redundanten Auslegung der Autonomiebereiche begründet. Üblicherweise sind die Autonomiebereiche und davon insbesondere die Zuständigkeitsbereiche der Agenten eines Multiagentensystems untereinander nicht überschneidungsfrei. Anders ausgedrückt, verfügt ein Multiagentensystem über einen gewissen Grad an Redundanz in der Funktionalität der Agenten. Genauer betrachtet kann man unter den Agenten zwischen Generalisten, welche eine vergleichsweise große Palette an Fähigkeiten besitzen, und Spezialisten unterscheiden, welche eine eingeschränkte Palette an Fähigkeiten besitzen, diese aber im Allgemeinen mit höherer Qualität vollbringen.

Als Voraussetzung für eine dynamische Anordnung der Agenten zu einem funktionstüchtigen Multiagentensystem wird in dieser Arbeit von einem einheitlichen Modell der in dem System vertretenen Fähigkeiten ausgegangen. Dieses ist aus dem Anwendungsbereich heraus zu definieren und ist Teil der Schnittstelle zwischen den Agenten. Dieses auf Mechanismen der Generalisierung und Vererbung basierendes *Fähigkeitsmodell* fungiert gleichzeitig als operationale Schnittstelle zu der Funktionalität eines die Fähigkeit besitzenden Agenten. Aus Entwurfssicht eines Agenten gibt es eine Abbildung von der Menge der von dem Agenten unterstützten Fähigkeiten in die Menge der Pläne des Agenten. Allen Agenten gemeinsam ist die Fähigkeit, ihre eigenen Fähigkeiten aufzuzählen.

Mit der Definition des Fähigkeitsmodells wird eine wesentliche Entwurfsentscheidung getroffen: Zweck der operationalen Schnittstelle ist es, unter noch zu beschreibenden Voraussetzungen die Fähigkeit der Agenten einer Interaktionssteuerung zur Verfügung zu stellen. Die Interaktionssteuerung soll jedoch nur in dem Rahmen Einfluß auf die Ausgestaltung einer Fähigkeit ausüben, in dem die Autonomie des Agenten weiterhin gewahrt bleibt. Wie dies erfolgt, wird durch das Fähigkeitsmodell definiert.

Vorstellbar ist folgende Gestaltung der Schnittstelle: Eine Handlungsprimitive, welche Teil einer Fähigkeit ist, braucht ein Agent nur dann auszuführen, wenn die Ausführung seine internen Restriktionen nicht verletzt. Mit Hilfe der Restriktionen läßt sich dann eine Entscheidungsfunktion modellieren, welche die Nutzung der Fähigkeit überwacht. Die Schnittstelle kann dann diese Restriktionen offenlegen, so daß es der Interaktionssteuerung möglich ist, die Entscheidungsfunktion des Agenten zu berücksichtigen.

Allgemein obliegt es dem Agenten zu entscheiden, welche seiner Fähigkeiten er wann in eine Interaktion einbringt. Somit bestimmt letztlich er, welchen Grad an Entscheidungsfreiheit er bereit ist abzutreten.

Außer Agenten kommen in einem Multiagentensystem häufig auch passive Objekte vor, welche nicht in der Lage sind zu Handeln – also keine Fähigkeiten besitzen – sondern an denen Handlungen vorgenommen werden – also Operationen ausgeführt werden. Bei-

spiele sind im allgemeinen informationstechnische Dokumente oder im Beispiel eines Produktionssystems physische Objekte wie Werkstücke. Abhängig vom Systementwurf, kann jedoch selbst einem an sich passiven Objekt die Fähigkeit (in Form eines ihm zugeordneten Agenten) gegeben werden, an Verhandlungen teilzunehmen. So könnten im Beispiel des Produktionssystems die Werkstücke untereinander verhandeln, zu welchen Losen sie sich gruppieren, um gemeinsam durch die Bearbeitungskette aus Produktionsmaschinen geschleust zu werden. Ist ein solcher Entwurf in kleinen Systemen ein praktikabler Weg, so kann dies ab einer gewissen Systemgröße zu Engpässen im Kommunikations- und im Rechensystem führen. Gleichfalls kann eine derartige Modellierung für gegebene Entwurfsanforderungen nicht adäquat sein.

Aus diesem Grund existieren in dem hier vorgestellten Interaktionsmodell neben den *Agenten* eine Menge von *Ressourcen*, welche passive Anwendungsobjekte repräsentieren. Diese unterstützen Operationen, durch die Änderungen an ihnen (und ihren Daten) vorgenommen werden können.

Der informationstechnische Anteil der Ressourcen kann in Datenbanken persistent gespeichert sein. Im Sinne der objektorientierten oder der objektrelationalen Datenbanken werden Operationen durch eine Objektklasse definiert, der die Ressource angehört. Aufgrund von Mehrfachvererbung oder Assoziationen ist es jedoch möglich, daß eine Ressource eine Vielzahl von Operationen unterstützt. Diese sollen entsprechend den Fähigkeiten der Agenten in einem einheitlichen Modell definiert sein. Die Verwaltung der Operationen sowie deren Ausführung könnte dann von der Datenbank übernommen werden. Das Datenbanksystem müßte hierzu um die Funktionalität erweitert werden, an einer Interaktion als Agent teilzunehmen. Die Zuständigkeit dieses Agenten wäre es, die Funktionalität der von ihm verwalteten Ressourcen in entsprechende Interaktionsbeziehungen einzubringen.

Die Strukturierung des Systems im Sinne einer Systemarchitektur erfolgt durch die dynamische Anordnung der Agenten. Das bereits genannte, die Beziehungen gestaltende Reglement wird durch die Spezifikation von Interaktionsmechanismen in Form von *Interaktionsverfahren* definiert Interaktionsverfahren sind in einer Programmiersprache zu definieren, welche eine eindeutige Semantik besitzt. Diese Verfahren steuern die Koordination oder Kooperation zwischen den zugeordneten Agenten und bauen dabei auf den Fähigkeiten der Agenten auf.

Startpunkt eines Multiagentensystems (Abbildung 3.1) ist das *Basisinteraktionsverfahren*, an dem alle im System bekannten Agenten teilnehmen und über welches neue Interaktionsverfahren instantiiert werden. Das Basisinteraktionsverfahren existiert über die gesamte Laufzeit des Systems. Von ihm ausgehend werden ein oder mehrere Verfahren gestartet, welche unter den Agenten unterschiedliche Organisationen einrichten. Eine Organisation besteht aus einer Menge von projekt- oder zielbezogenen *Team-Interaktionsverfahren*, und aus einem *Organisations-Interaktionsverfahren*, welches die Teams koordiniert. In der Zuständigkeit einer Organisation liegt es, das Reglement festzulegen, dem sich die Mitglieder der Organisation unterwerfen. Hinsichtlich der in einer Organisation zugelassenen Interaktionsformen kann die Organisation sogenannte Positivlisten führen. Die Teilnahme an einem Interaktionsverfahren aus dieser Liste ist dann erlaubt

und hinsichtlich der Sicherheitsanforderungen und der modellierten Fähigkeiten und Berechtigungen geprüft.

In einzelnen Phasen der Team-Interaktionsverfahren können vordefinierte *Subinteraktionsverfahren* (z. B. eine Terminvereinbarung oder eine Budgetverhandlung) verwendet werden, wenn sie auf der Positivliste der Organisation vermerkt sind. Während der Teilnahme an einem Subinteraktionsverfahren ist eine Rolle im aufrufenden Interaktionsverfahren solange blockiert, bis sie aus dem Subinteraktionsverfahren zurückkehrt. Der Beitritt zu einem Interaktionsverfahren wird nicht von einer Rolle sondern von dem Agenten selbst ausgelöst.

Die Team-Interaktionsverfahren lassen sich als dynamische Netzwerke [Ste 93] (S. 410) auffassen, welche aufgrund eines Koordinationsbedarfs eingerichtet werden. Da ein Agent gleichzeitig an mehreren Interaktionsverfahren beteiligt sein kann, ist die Anordnung der Agenten zu einem System hochflexibel. Organisationen können Team-Interaktionsverfahren nach außen öffnen, so daß auch Agenten, welche nicht Teil der Organisation sind, sich an ihnen beteiligen können.

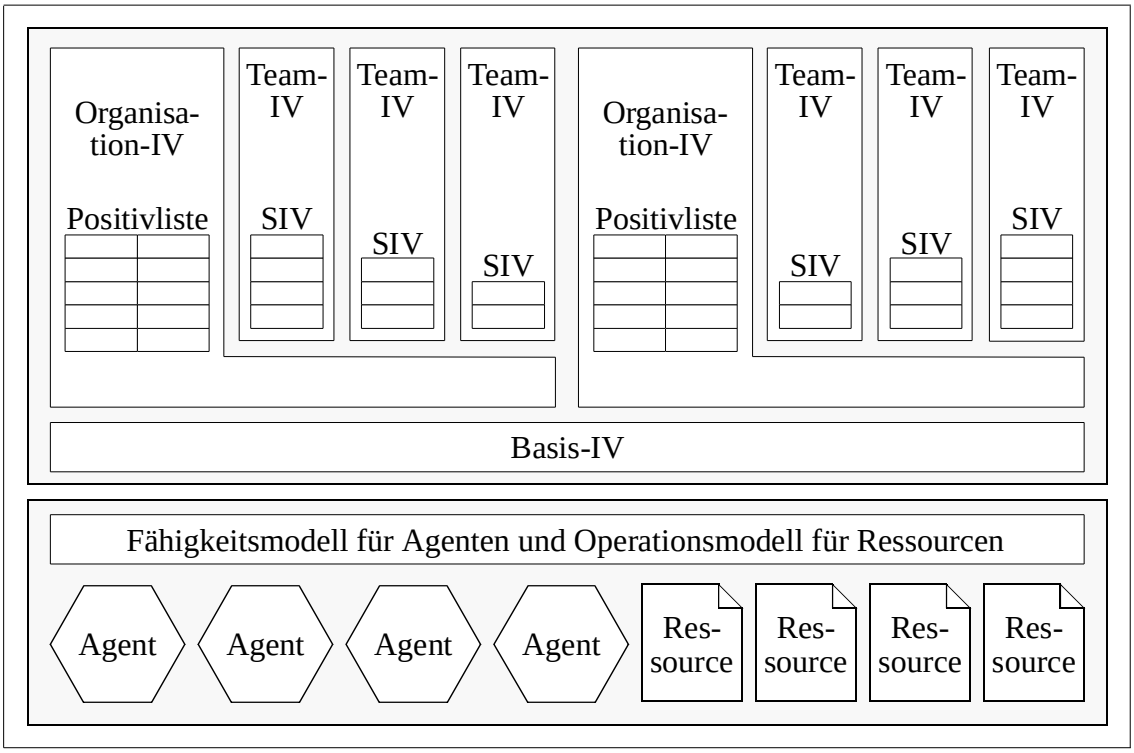


Abbildung 3.1: Struktur eines auf Interaktionsverfahren(IV) und Subinteraktionsverfahren (SIV) aufbauenden Multiagentensystems

Ein Multiagentensystem wird also durch sein Fähigkeits- und Operationsmodell, durch die Menge der an ihm beteiligten Agenten und Ressourcen und durch die Menge der Interaktionsverfahren definiert. Ein ausgezeichnetes Interaktionsverfahren, das Basisinteraktionsverfahren, definiert den Startpunkt des Systems.

### 3.2.3 Interaktionsmuster

Das Identifizieren von Mustern ist eine effizienzsteigernde Maßnahme in der Entwicklung von Lösungskonzepten. Ein Muster beschreibt zu einer Problemstellung, mit welcher man immer wieder von neuem konfrontiert ist, das Schlüsselkonzept der Lösung in einer Form, die es dem Entwickler ermöglicht, das Konzept immer dann zu verwenden, wenn er vor die Problemstellung gestellt wird. Im Gegensatz zu der reinen Wiederverwendung von Softwaremoduln auf Codebasis ist die Verwendung von Mustern eine Wiederverwendung im Entwurf. Das wohl bekannteste Entwurfsmuster ist das Model-View-Controller (MVC) Paradigma [KrPo 88], welches heute durchgehend zur Konstruktion von Benutzerschnittstellen eingesetzt wird.

Muster werden zu sogenannten Bibliotheken zusammengefaßt, von denen es im Bereich des objektorientierten Entwurfs [GHJV 94] und im Bereich der Softwarearchitekturen [Bus 96], [Dur 99] fundierte und weit verbreitete Beispiele gibt.

Im Bereich der Multiagentensysteme steht der Entwurf von Mustern für die Entwicklung von Multiagentensystemen noch am Anfang. Ein allgemein bekanntes und vielfach diskutiertes und verwendetes Interaktionsmuster ist das Anbieter/Nachfrage-Muster, welches dem *Contract-Net-Protocol* [Smi 80] zu Grunde liegt. Anderen Koordinationsmechanismen fehlt es an Allgemeingültigkeit der Problemstellung und der Allgemeingültigkeit der Annahmen hinsichtlich des Umfeldes der Problemlösung.

In naher Zukunft wird sich die Anzahl der Arbeiten zu diesem Thema erhöhen: Auf der Agent'99 stellen Sandra Hayden, Christina Carrick und Yang Qiang eine Arbeit mit dem Titel "*Architectural Design Patterns for Multiagent Coordination*" vor. Im Oktober 1999 wird das Symposium ASA/MA'99 (*First International Symposium on Agent Systems and Applications (ASA'99), Third International Symposium on Mobile Agents (MA'99)* und *Featuring the Third Dartmouth Workshop on Transportable Agents (DWTA'99)*) in Palm Springs, California, USA sich unter anderem mit Entwurfsmustern für Multiagentensysteme befassen. In der Zeitschrift "*Autonomous Agents and Multi-Agent Systems*" wird 1999 ein Sonderband über *Coordination Mechanisms and Patterns for Web Agents* erscheinen.

Das in dieser Arbeit vorgestellte Interaktionsmodell ist für die Beschreibung von Interaktionsmustern dahingehend geeignet, daß es die Struktur der Interaktion, die Teilnehmer mit ihren Zuständigkeiten sowie deren Zusammenarbeit beschreiben kann, ohne von einer spezifischen Agentenarchitektur abhängig zu sein.

Im folgenden werden wir den Begriff des *Interaktionsverfahrens* für einen Algorithmus verwenden, welcher aus einzelnen, von Agenten ausführbaren Aktionen, besteht und diese Aktionen koordiniert und terminiert. Soll aus dem Kontext heraus die Betonung mehr auf die Lösungsidee als den Algorithmus gelegt werden, so wird der Begriff des *Interaktionsmechanismus* verwendet.

### 3.2.4 Rollen- und Objektkonzept

Die zentrale Komponente im Interaktionsmodell ist das *Rollenkonzept*, durch welches die Abstraktion bereitgestellt wird, mit der ein Interaktionsmechanismus unabhängig von konkreten Individuen definiert wird. Allgemein betrachtet, wird ein Interaktionsmechanismus durch eine Menge von Rollen beschrieben, welche nach einem zugeordneten Verfahren untereinander interagieren.

Rollen bilden die Schnittstelle zwischen dem Interaktionsverfahren und den Agenten. Ein Verfahren, welches in einzelne Phasen zerlegt wird, verfügt in jeder Phase über eine Menge von Rollen. Die Spezifikation der in einer Phase ablaufenden Interaktion erfolgt durch die Definition der jeder beteiligten Rolle zugeordneten Aktion. Die Aktion welche selbst wieder aus einzelnen Anweisungen zusammengesetzt ist, kann durch sie Handlungen von konkreten Individuen auslösen, Entscheidungen treffen und Ergebnisse anderen Rollen der Phase sichtbar machen.

Ein Agent durchläuft während einer Teilnahme an einem Interaktionsverfahren eine Kette von Rollen. Nimmt der Agent mehrfach an dem Verfahren teil, so wird jeder Teilnahme eine eigene Kette zugeordnet. In einer Kette ist für einen Agenten zu einem Zeitpunkt immer nur eine Rolle instantiiert und damit aktiv. Handlungen, welche eine Rolleninstanz auf einem Individuum auslösen, werden dem zugehörigen Agenten zugeordnet. Eine Rolleninstanz kann gleichzeitig nur einem Agenten zugeordnet sein, jedoch können von einer Rolle für verschiedene Ketten gleichzeitig mehrere Instanzen existieren.

Mit dem Rollenkonzept werden also für jede Interaktionsphase sowohl die Handlungen eines Teilnehmers als auch dessen Informationsaustausch mit anderen Teilnehmern beschrieben. Das Rollenkonzept bietet damit eine verteilte, agentenorientierte Sicht auf eine Interaktion, jedoch mit dem Vorteil, daß agenteninterne Aspekte verborgen sind. Interaktionsverfahren bieten darüberhinaus eine globale Sicht auf die gesamte Interaktion.

Das *Objektkonzept* stellt analog zum Rollenkonzept die Abstraktion bereit, mit der ein Interaktionsmechanismus unabhängig von durch die Agenten gemeinsam genutzten, konkreten Ressourcen (z. B. Dokumenten) definiert wird. Objekte modellieren die Schnittstelle, mit der die Rollen auf die Daten der Ressource zugreifen. Der Zugriff auf gemeinsame Daten stellt an sich wieder eine Interaktion dar. Hierfür gibt es bereits ausgereifte Mechanismen (z. B. Teamobjekte nach [Rüd 92]), die hier zu einem Objekt zusammengefaßt werden. Ein Interaktionsverfahren, welches in anderen Verfahren eingebunden werden soll, läßt sich im Interaktionsmodell durch ein Objekt beschreiben.

Analog zu Rollen können Ressourcen in komplexen Interaktionen durch eine Verkettung von einzelnen Objekten modelliert werden. Jedes einzelne Objekt definiert dann eine spezifische Sicht und einen gewissen Ausschnitt aus den möglichen Operationen auf der Ressource.

### 3.2.5 Interaktionsverfahren

Ein Interaktionsverfahren stellt eine komplex strukturierte normative Regel dar, auf die sich Individuen, welche an der Interaktion beteiligen sind, verlassen können, aber die sie auch einhalten müssen. Ordnet sich ein Individuum einer normativen Regel unter, was durch den Teilnehmer an dem Verfahren zugesichert wird, so gibt es willentlich Teile seiner Entscheidungsfreiheit zugunsten des Funktionierens der Regel auf.

Verwenden wir kurz zur Veranschaulichung als Beispiel die Straßenverkehrsordnung und ihre Anwendung in Mitteleuropa. Fassen wir den Verkehr als eine komplexe Interaktion auf, welche durch ein genauer zu spezifizierendes, auf der Straßenverkehrsordnung basierendes Verfahren gesteuert wird. Als Verkehrsteilnehmer verlassen wir uns (obwohl unsere Gesundheit davon abhängt) darauf, daß andere Verkehrsteilnehmer sich an das Verfahren halten. Der kritische Leser möge anmerken, daß er sich keineswegs auf das korrekte Verhalten anderer verlasse. Ihm sei jedoch entgegnet, daß auch er an einer seit längerer Zeit grünen Ampel ohne anzuhalten vorbeifährt, und eben nicht sichergeht, daß eine Kollision mit kreuzendem Verkehr ausgeschlossen ist. Durch die Teilnahme am Straßenverkehr ordnet sich jeder einzelne der Straßenverkehrsordnung unter, wenn nicht in allen Details, so doch in den Grundzügen. Hierbei gibt er Teile seiner Entscheidungsfreiheit auf.

Betrachtet man die Anwendung der Straßenverkehrsordnung im südeuropäischen Raum, so mag man sich an die veränderte Bedeutung der Leuchtzeichen einer Ampel erinnern, welche eine eher vorfahrtsregelnde Bedeutung zu haben scheinen. Dennoch gilt das oben gesagte dahingehend, daß der einzelne Teilnehmer Entscheidungsfreiheit abgibt und entsprechend des Reglements Vorfahrt gewährt.

Im Gegensatz zu den Sozialwissenschaften hat die Informatik den Vorteil, ihre Systeme konsistent zu den umzusetzenden Konzepten entwerfen zu können. Diesen Vorteil nutzen wir bei der Definition der Abgabe von Entscheidungsfreiheit: Ein Agent, welcher an einem Interaktionsverfahren teilnimmt und dort eine Folge von Rollen annimmt, führt *alle* Handlungen aus, die von den Rollen in ihm ausgelöst werden.

Welchen Grad an Entscheidungsfreiheit ein Agent bereit ist, an eine Interaktion abzutreten, ist dann die eigentliche Fragestellung. Dieses ist durch ein entsprechendes Fähigkeitsmodell zu definieren. Welche Fähigkeit ein Agent in eine Interaktion letztendlich einbringt, entscheidet der Agent selbst.

Eine Interaktion kann in sich nebenläufig und nicht-deterministisch ablaufen. Die Beschreibung einer Interaktion verfolgt dabei eine zustandsorientierte Sicht in Form der einzelnen Interaktionsschritte. Als Entwurfssprache wird aufgrund dieser Charakteristika eine um das Rollen- und das Objektkonzept erweiterte Petrinetzklasse verwendet. Neben den genannten Eigenschaften besitzen Petrinetze den Vorteil eines soliden theoretischen Fundaments und sind nach [Bal 96] (S. 319) “das einzige weit verbreitete Basiskonzept, zur Modellierung kooperierender Prozesse”.

Petrinetze sind unflexibel hinsichtlich strukturellen Veränderungen. Die durch die Netzstruktur vorgegebene Flußrelation kann im Allgemeinen während des Ablaufs eines

Petrinetzprozesses nicht mehr verändert werden. Dahingegen bieten Petrinetze das Konzept der Marken, welches den dynamischen Zustand des Netzes beschreibt. Marken können während des Ablaufs eines Prozesses erzeugt und vernichtet werden.

In der Entwurfssprache der Interaktionsverfahren machen wir uns dieses Konzept zu Nutze, indem wir eine Kette von Rollen, welche eine Teilnahme eines Agenten an einem Interaktionsverfahren symbolisiert, als eine Rollenmarke modellieren, welche durch Schaltvorgänge von Transitionen eine Folge von Rollenwechsel erfährt. Das heißt: im Allgemeinen erzeugen und vernichten die in einer Interaktion modellierten Transitionen keine Marken, sondern leiten die selbe Anzahl von Marken aus dem Vorbereich in den Nachbereich (verändert) weiter. Analog dazu modellieren wir eine Objektmarke, welche den Sichtbarkeitszyklus einer Ressource in einem Interaktionsverfahren beschreibt.

In der Modellierung von Petrinetzen werden üblicherweise Transitionen dafür eingesetzt, Veränderungen an der Umwelt vorzunehmen. Im Gegensatz dazu ist in unserem Modell die Rollenmarke die handelnde Komponente. Zustandsänderungen, welche sie vollzieht, können im Zusammenspiel mit anderen Marken Transitionen aktivieren. Eine Transition leitet die sie aktivierenden Marken weiter und führt während des dabei zu vollziehenden Rollenwechsels einen Informationsaustausch zwischen den Marken durch. Transitionen vollziehen durch ihr Schalten nur nach, was die Gruppe der Marken schon durch Einnehmen des Zustandes vorweggenommen hat, welcher die Transition aktivierte. Schaltet eine Transition, so ändert sich die Markierung, wird Information zwischen den beteiligten Rollen und Objekten ausgetauscht und ein Rollen- und Objektwechsel durchgeführt.

Durch das Schalten einer Transition, werden die Rollen, welche die Transition aktivierten, synchronisiert. Somit wird der Abschluß der vorangegangenen Phase und der Beginn der folgenden Phase für diese Rollen allgemein bekannt (common knowledge[1]), wovon die Rollen in der folgenden Phase ausgehen können. Dies vereinfacht wesentlich den Aufwand für die korrekte Modellierung.

## 3.3 Interaktionsmodell mit Rollen

Im folgenden Abschnitt wird das Interaktionsmodell in der ersten Iterationsstufe eingeführt. Der Abschnitt schließt mit einem Beispiel und einer Diskussion ab. Im Zentrum der Darstellung steht das abstrakte Interaktionsmodell, welches die Semantik der Sprache festgelegt, mit welcher das Schema eines Interaktionsverfahren definiert wird. Zur Laufzeit wird aus dem Schema eine Instanz erzeugt, auf welcher die Interaktion abläuft.

1. Im Allgemeinen ist Common Knowledge in asynchronen Netzen (mit nicht garantierter Zustellzeit einer Nachricht) durch Kommunikation nicht erreichbar. In sicheren Netzen mit einer maximalen Verzögerungszeit $\varepsilon$ ist unter Berücksichtigung des Sendezeitpunkts $t_0$ Common Knowledge zum Zeitpunkt $t_0 + \varepsilon$ erreichbar [HaMo 90].

Der Abschnitt beginnt mit der Definition der Schnittstelle zwischen einem Agentenmodell und dem abstrakten Interaktionsmodell.

### 3.3.1 Agentenmodell

In einer Anwendung ist die Aufgabenverteilung zwischen den Interaktionsmechanismen und den an ihnen beteiligten Agenten wie folgt: Die an einer Koordination beteiligten Agenten wählen geeignete Interaktionsmechanismen aus und ordnen sich ihrer Kontrolle unter. Anders ausgedrückt, steuern die Interaktionsmechanismen die an der Koordination beteiligten Agenten, welche als gesteuerter Teil ihre Fähigkeiten dem Interaktionsmechanismus zur Verfügung stellen.

Fähigkeiten von Agenten lassen sich in sensomotorische und kognitive Fähigkeiten unterteilen. Beispiele für sensomotorische Fähigkeiten sind bei einem Kamerakopf die Bildaufnahme aus verschiedenen Richtungen oder bei einer Roboterplattform die Beweglichkeit entsprechend dem vorhandenen Antrieb. Beispiele für kognitive Fähigkeiten sind Erkennung und Lokalisation von Objekten, die Interpretation einer Szene oder die Navigation zu einem Raumpunkt.

**Definition 3.1** *Fähigkeit*

Eine Fähigkeit ist das Können, in einer bestimmten Art und Weise bewußt[1] und kompetent auf sich als Agenten und auf die Umwelt einzuwirken. Im einzelnen ist eine Fähigkeit durch eine Menge von *Handlungsprimitiven* und durch einen *Handlungskontext* definiert.

Die Schnittstelle zwischen dem Agentenmodell und dem Interaktionsmodell basiert also auf der Menge der Fähigkeiten, welche die Agenten innerhalb einer Anwendung besitzen. Diese Schnittstelle wird in Form von Diensten realisiert, über die Interaktionsverfahren die Fähigkeiten beteiligter Agenten ansprechen.

**Definition 3.2** *Dienst*

Ein Dienst $d$ sei durch das Paar $d = (k, p)$ beschrieben, wobei $k$ den *Dienstkontext* und $p$ die Menge der *Dienstprimitiven* bezeichnet. Mit $D$ wird die Menge aller Dienste bezeichnet.

1. Das bewußte Ausführen einer Fähigkeit soll dabei bedeuten, daß der Agent die Konsequenzen seines Handelns abschätzt und sich im Sinne der Autonomie (Abschnitt 3.1) für die Ausführung entscheidet. Dies unterscheidet den Begriff der Fähigkeit von dem einer API (Application Programmer Interface).

**Bemerkung 3.1**

Ein Leser mit Kenntnissen im Corba-basierten Entwurf von verteilten Objekten [Red 96] wird sich fragen, wie man sich die Abbildung der Dienstdefinition auf die Konstrukte der Beschreibungssprache IDL vorstellen soll. Ein Dienst aus Definition 3.2 wird durch ein IDL-Modul beschrieben, welches aus Typ-, Konstanten-, Exception- und Interface-Definitionen aufgebaut sein kann. Dienstdefinitionen, welche aus einer Menge von Interfaces bestehen, besitzen als Eintrittspunkt eine “Fassade” [GHJV 94]. Über “Fabrikmethoden” erzeugt die Fassade Instanzen von konkreten, durch die Dienstimplementierung bestimmten Hilfsklassen, deren Schnittstelle durch Interface-Definitionen des IDL-Moduls beschrieben sind.

Der Dienstkontext aus Definition 3.2 ergibt sich aus den Attributen der Fassade, sowie aus den Instanzen der Hilfsklassen und ihren Attributen, welche einem Dienstnutzer bekannt sind. Die Dienstprimitive ergeben sich dann aus den Methoden der Fassade und aus den Methoden der Hilfsklassen, welche dem Dienstnutzer bekannt sind.

Abschnitt 3.4.5 enthält ein Beispiel einer aus mehreren Interfaces aufgebauten Dienstdefinition.

Das abstrakte Interaktionsmodell fordert von einem zugehörigen Agentenmodell folgende Minimaleigenschaft:

**Definition 3.3** *Agent*

Einen Agenten *ag* definieren wir als eine Menge von Fähigkeiten und den zugehörigen *Diensten*, welche die Schnittstelle zu den Fähigkeiten bilden. Mit $A$ bezeichnen wir die Menge aller Agenten.

**Bemerkung 3.2**

In Agentenmodellen werden meist weitergehende Eigenschaften von einem Agenten gefordert, wie z. B. Autonomie und Rationalität, welche durch Bewertungs- und Entscheidungsfunktionen modelliert werden. Mit der Definition 3.3 wollen wir den konkreten Modellen, die mit dem Interaktionsmodell eingesetzt werden können, nicht einschränken.

Jedoch fordern wir [BML 98], daß konkrete Agentenmodelle eine einheitliche Konnotation von verwendeten *Konzepten* definieren, welche die semantische Kompatibilität der Agenten sicherstellt. Agenten, welche sich z. B. mit Marktmechanismen koordinieren, benötigen die Konzepte Geld, Ware und Handel. Weiterhin fordern wir, daß aufbauend auf den Konzepten eine Dienststruktur definiert wird, welche die syntaktische Kompatibilität sicherstellt. Über diese Dienststruktur werden Mindestfähigkeiten von Agenten definiert, sowie typische Kombinationen von Fähigkeiten beschrieben. Diese Kombinationen definieren Klassen „gleichfähiger“ Agenten. Die Dienststruktur bildet dann die Schnittstelle zwischen den Interaktionsverfahren und den Agenten.

### 3.3.2 Handelnde (Rollenkonzept)

Der Grundgedanke bei dem Entwurf des Modells ist es, den Interaktionsmechanismus unabhängig von individuellen Agenten beschreiben zu können, den Agenten jedoch die Möglichkeit zu individuellen Entscheidungen zu belassen. Das Modell soll insbesondere dazu geeignet sein, verteilte, agentenbasierte Interaktionsmechanismen zu beschreiben. Hierfür wird der zentrale Begriff eines Handelnden eingeführt:

**Definition 3.4** *Rolle*

Eine Rolle *r* wird durch ein Tripel $r = (D_{Ref}, \vec{z}, ak)$ beschrieben. Dabei ist $D_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Dienste*, $\vec{z}$ ein Vektor von *Zuständen* und *ak* eine *Aktion*. Die Aktion *ak* besteht aus Dienstprimitiven aus $D_{Ref}$ und Operationen auf Zustände (Komponenten) aus $\vec{z}$. Wir bezeichnen mit $\mathrm{D_{Ref}}(r)$ die Menge der referenzierten Dienste, mit $\mathrm{z}(r)$ den Vektor $\vec{z}$ der Zustände beziehungsweise mit $\mathrm{ak}(r)$ die Aktion der Rolle *r*. Mit $R$ bezeichnen wir die Menge aller Rollen.

Der Begriff einer Rolle ist eine Abstraktion von individuellen Agenten und ihrer Aktionen in einem bestimmten Bereich des Interaktionsverfahrens. Das Laufzeitverhalten der Rolle kann jedoch von individuell zu treffenden Entscheidungen des Agenten abhängen: Eine Instanz einer Rolle – im Folgenden als Rollenmarke definiert – führt die Aktion der Rolle aus, welche über Dienste auf den ihr zugeordneten Agenten zugreift. Ergebnisse aus den Aufrufen von Dienstprimitiven können Zustände der Rolleninstanz verändern, Zustände der Rolleninstanz können die Auswahl und die Parametrisierung von Dienstprimitiven beeinflussen.

**Definition 3.5** *Rollenmarke*

Eine Rollenmarke ist als ein Tripel $\rho = (r, ag, \nu)$ definiert, wobei $r \in R$ eine *Rolle*, $ag \in A$ einen *Agenten* und $\nu\colon \mathrm{D_{Ref}}(r) \rightarrow ag$ den *Namenskontext* definiert, welcher den von der Aktion der Rolle referenzierten Diensten die Fähigkeiten des Agenten zuordnet. Wir bezeichnen mit $\mathrm{r}(\rho)$ die Rolle der Marke.

Die Struktur des Rollenbegriffs veranschaulicht Abbildung 3.2.

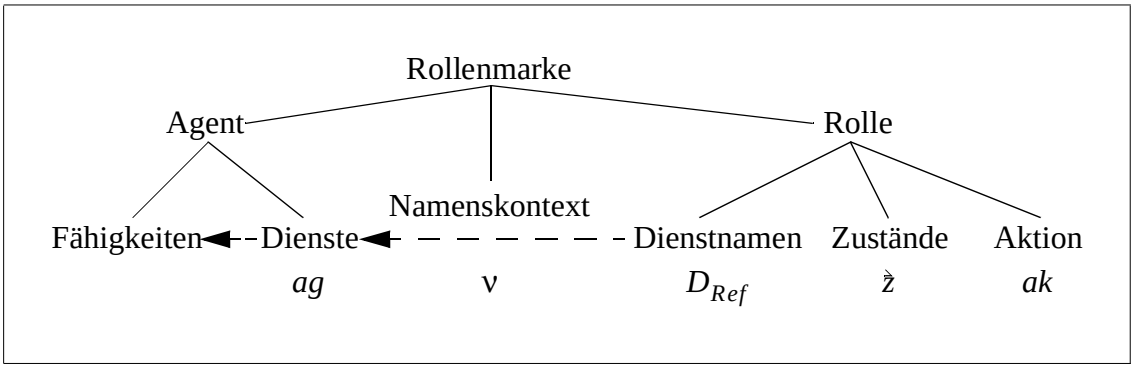


Abbildung 3.2: Struktur der Begriffe des Rollenkonzepts

### 3.3.3 Interaktionsnetz mit Rollenkonzept

Ein Interaktionsverfahren wird als eine Sequenz von Phasen aufgefaßt, welche auch parallele, voneinander unabhängige Pfade enthalten kann. In einer Phase interagieren Rollen hinsichtlich der Ausführung eines Interaktionsschritts. Die Abbruchbedingung eines Schritts und die Aktivierung des Folgeschritts wird durch einen Phasenübergang beschrieben.

Die Beschreibung eines Interaktionsverfahrens in Form von Phasen und Phasenübergängen läßt sich als Pr/T-Netz (Abschnitt 2.3.6) auffassen, dessen Netzmodell um das Rollenkonzept erweitert ist.

Zuerst führen wir den Begriff der Phase formal ein und ordnen ihm den Rollenbegriff unter. Dann definieren wir das Interaktionsnetz als die Erweiterung der Pr/T-Netze durch das Rollenkonzept und erklären die Schaltregel der Transitionen über die Instanzen der Rollen. Diese werden durch Marken im Interaktionsnetz repräsentiert.

**Definition 3.6** *Phase*

Eine Phase *p* ist ein Schritt in einem Interaktionsverfahren, den eine oder mehrere Rollen interaktiv ausführen. Sei *P* die Menge der Phasen in einem Interaktionsverfahren. Dann definieren wir eine Abbildung $\gamma$: $P \rightarrow 2^R$ , welche jede Phase mit der Menge von Rollen beschriftet, die an ihr beteiligt sind. Wir bezeichnen mit $\gamma_p$ die Rollenmenge der Phase *p*.

Abbildung 3.3 stellt eine Phase dar, in der sich eine Rollenmarke befindet.

Abbildung 3.4 stellt beispielhaft die Beschriftung einer Transition sowie ihrer ein- und ausgehenden Kanten mit Termen dar. Die Signatur und Interpretation der Terme werden in der Definition 3.7 formal eingeführt.

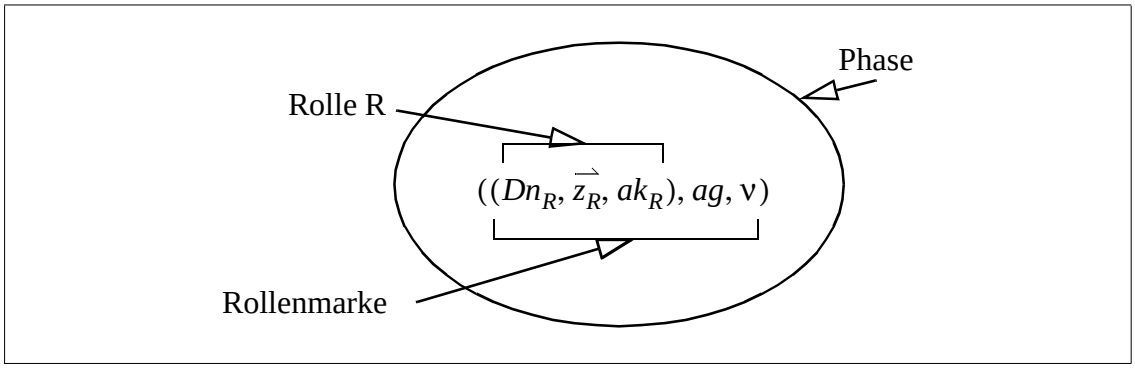


Abbildung 3.3: Stelle mit einer Marke

Mit dem Beispiel aus Abbildung 3.4, welches einen Ausschnitt eines laufenden Interaktionsverfahren zeigt, möchten wir einen ersten Einblick in die Beschreibungssprache geben: In Phase 1 ($\gamma_1 = \{R, S\}$) befinden sich zum Zeitpunkt der Betrachtung vier Rollenmarken, den ersten beiden ist die Rolle *R* und den anderen beiden ist die Rolle *S* zugeordnet. Die Phase 2 ($\gamma_2 = \{T, U\}$) sei leer. Der Zustand des betrachteten Ausschnittes ist gegeben durch die Markierung *M* der Phasen 1 und 2:

$$M(1) = \{((Dn_R, \overrightarrow{z_{R,0}}, ak_R), ag_0, \nu_{R,0}), ((Dn_R, \overrightarrow{z_{R,1}}, ak_R), ag_1, \nu_{R,1}),$$
$$((Dn_S, \overrightarrow{z_{S,2}}, ak_S), ag_2, \nu_{S,2}), ((Dn_S, \overrightarrow{z_{S,3}}, ak_S), ag_3, \nu_{S,3})\}$$
$$M(2) = \varnothing$$

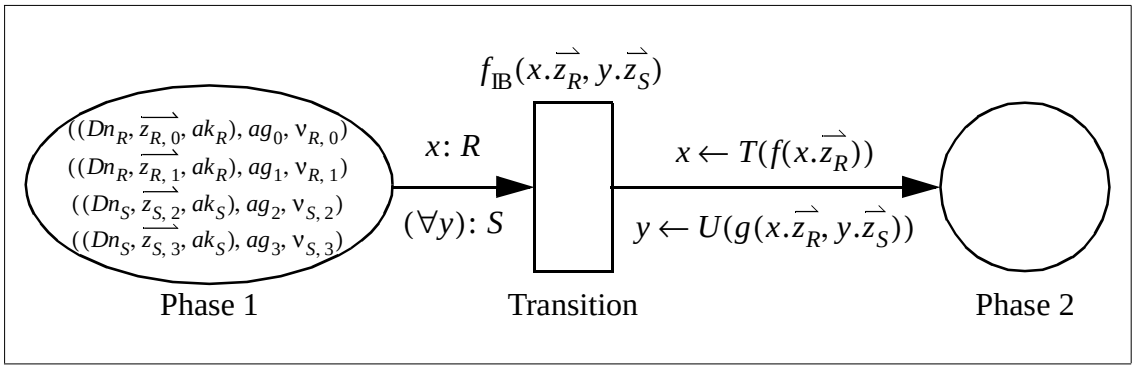


Abbildung 3.4: Beschriftungen einer Transition und ihrer Kanten

Die Schaltbedingung der Transition wird durch den booleschen Term $f_{IB}$ beschrieben, welcher zwei freie Variablen (*x* und *y*) besitzt. Die Beschriftung der eingehenden Kante definiert *x* als eine Variable vom Typ (Rolle) *R* und y vom Typ *S*. Der Variablen *y* steht ein Allquantor voraus, welcher alle Marken dieses Typs bindet, unter denen $f_{IB}$ erfüllt ist. Die ausgehende Kante ist mit je einem Term für die Variablen *x* und *y* beschriftet. Der Term für *x* spezifiziert, daß die an *x* gebundene Marke ihre bisherige Rolle abgibt und in

der nächsten Phase die Rolle *T* annimmt. Dabei findet ein Informationstransfer von der Rolle *R* zur Rolle *T* statt, welcher durch *f* definiert ist. Für *y* beschreibt *g* den Informationsaustausch beim Rollenwechsel zu *U*.

Die Schaltbedingung kann nun durch unterschiedliche Belegungen der freien Variablen mit Marken aus Phase 1 erfüllt sein, wobei *x* mit genau einer Marke und *y* mit einer Menge von Marken belegt wird. Aus diesen Belegungen sei nicht-deterministisch diejenige Belegung ausgewählt, welche *x* mit $\text{Marke}_1$, und *y* mit $\{\text{Marke}_2, \text{Marke}_3\}$ belegt. Nach Schalten der Transition ergibt sich nun folgende Markierung $M'$:

$$M'(1) = \{((Dn_R, \overrightarrow{z_{R,0}}, ak_R), ag_0, \nu_{R,0})\}$$

$$M'(2) = \{((Dn_T, f(\overrightarrow{z_{R,1}}), ak_T), ag_1, \nu_{T,1}), ((Dn_U, g(\overrightarrow{z_{R,1}}, \overrightarrow{z_{S,2}}), ak_U), ag_2, \nu_{U,2}),$$
$$((Dn_U, g(\overrightarrow{z_{R,1}}, \overrightarrow{z_{S,3}}), ak_U), ag_3, \nu_{U,3})\}$$

**Bemerkung 3.3**

Es sei noch auf eine Besonderheit bei der Verwendung des Allquantors hingewiesen: Einer durch einen Allquantor gebundenen Variable (im Beispiel *y*) wird beim Schalten der Transition eine Menge von Marken zugeordnet. Terme referenzieren diese Menge von Marken durch Verwendung der Konstruktion $\forall y$: z. B. kann das Maximum von $z_i$ aller *y* zugeordneten Marken durch $\text{Max}(\forall y.z_i) := \text{Max}(\{y.z_i \mid \forall y\})$ beschrieben werden.

Man beachte, daß in dem Term (im Beispiel $y \leftarrow U(g(x.\overrightarrow{z_R}, y.\overrightarrow{z_S}))$), welcher den Rollenwechsel für alle der Variable zugeordneten Marken beschreibt, die Variable ohne den Quantor verwendet wird. Durch diesen Term wird definiert, wie der Informationstransfer für *jede einzelne* Marke der Rolle *S* zur zukünftigen Rolle *U* abläuft.

In allen anderen Termen ausgehender Kanten kann mit einer universell quantifizierten Variable (z. B. *y*) nur auf die Menge der zugeordneten Marken zugegriffen werden. Für nicht quantifizierte Variablen (z. B. *x*) wird auf genau eine, *die* ihr zugeordnete Marke zugegriffen.

**Bemerkung 3.4**

Die Verwendung des Allquantors ist in Petrinetzen allgemein nicht möglich. In Prädikat-Transitionsnetzen, die sich dadurch auszeichnen, daß dem Modellierer strukturierte Marken zur Verfügung stehen, läßt sich die Verwendung des Allquantors aber als eine abkürzende Schreibweise auffassen. In Abbildung 3.5 ist dies an einem einfachen Beispiel verdeutlicht:

Die Phase 1 enthält eine unbekannte Anzahl von Marken des Typs *S*, die beispielsweise von einer vorgelagerten Transition 1 einzeln erzeugt wurden. Ist Transition 2 aktiviert (z. B. aufgrund anderer eingehender Kanten die hier nicht dargestellt sind) und schaltet, so werden alle Marken des Typs *S* aus der Phase 1 abgezogen.

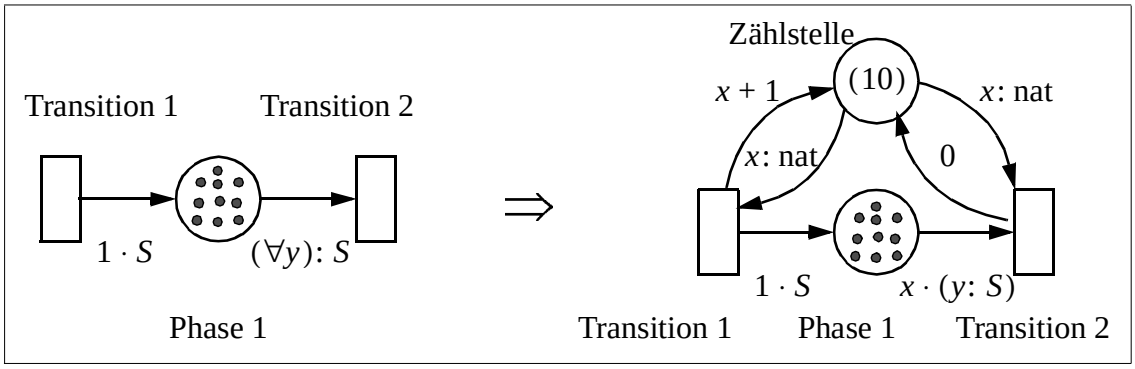


Abbildung 3.5: Entfalteter Netzausschnitt ohne Allquantor

Das entfaltete Netz besitzt eine zusätzliche Zählstelle, welche genau eine Marke enthält, die einen natürlichzahligen Wert annehmen kann. Der Wert dieser Marke gibt an, wieviele Marken des Typs *S* in der Phase 1 liegen. Jede Transition, welche Marken in der Phase 1 erzeugt, inkrementiert den Wert dieser Marke. Die Beschriftung der Kante von Phase 1 zur Transition 2 gibt an, daß genau soviele Maken des Typs *S* abgezogen werden, wie der Wert der Marke in Zählstelle angibt.

Allgemein benötigt man für jeden Allquantor und für jede Bedingung des Wächters, welche ein Selektionskriterium hinsichtlich der an die Variable gebundenen Marken definiert, eine eigene Zählstelle und für jede Zählstelle eine inkrementierende und eine dekrementierende Transition. Können wie in dieser Arbeit die Marken ihren Zustand in einer Phase selbständig ändern, so ist der Zustand der Marke um je ein Bit für jede Zählstelle zu erweitern, das angibt, ob ihr Zustandswechsel bereits in der entsprechenden Zählstelle berücksichtigt ist. Erst wenn sie von allen Zählstellen korrekt erfaßt ist, kann sie zur Aktivierung einer Transition beitragen.

**Bemerkung 3.5**

Im folgenden bezeichnen $\hat{S}$, $\hat{R}$, $\hat{Z}$, … die syntaktische Menge von Sorten, Rollen, Zuständen etc. Damit unterscheiden wir die Komponenten der Syntax von außerhalb des Interaktionsnetzes definierten Begriffen.

**Definition 3.7** *Interaktionsnetz*

Ein Interaktionsnetz ist ein Pr/T-Netz, welches um das Rollenkonzept erweitert ist.

Kurze Erinnerung an die Definition: $P$ = Menge der Phasen, $\gamma(p) \subseteq R$ Rollen der Phase, $z(r)$ = Zustandsvektor, $\rho$ = Rollenmarke, $r(\rho)$ = Rolle der Marke. Mit $|\vec{z}|$ wird die Länge eines Vektors $\vec{z}$ bezeichnet.

1. Die Erweiterung wird über die Signatur $\Sigma = (\hat{S}, \Phi)$ wie folgt definiert: Wir definieren die Menge der *Sorten* $\hat{S}$ entsprechend Tabelle 3.1.

| Menge der *Sorten* ... | $\hat{S} = \hat{R} \cup \hat{Z} \cup \hat{RM} \cup 2^{\hat{RM}}$ |
|---|---|
| *der Rollen* | $\hat{R} = \bigcup_{p \in P} \gamma(p)$ |
| *der Zustände der Rollen* | $\hat{Z} = \bigcup_{\hat{r} \in \hat{R}} \{\hat{S} \mid \hat{S} \text{ Sorte von } z(\hat{r})_i, 1 \le i \le \lvert z(\hat{r})\rvert\}$ |
| *der Rollenmarken* | $\hat{RM}_{\hat{r}} = \{\rho \mid r(\rho) = \hat{r}\}$, $\hat{RM} = \bigcup_{\hat{r} \in \hat{R}} \hat{RM}_{\hat{r}}$ |

Tabelle 3.1: Definition der Sorten des Interaktionsnetzes

Weiterhin definieren wir die Menge der *Funktionssymbole* $\Phi$ entsprechend Tabelle 3.2. Dabei bezeichnet $\pi_i$ die *Projektion* auf die *i*-te Komponente eines Zustandsvektors, mit $\kappa_{\hat{r}}$ bezeichnen wir die *Komposition* der Rolle $\hat{r}$ und mit $\varsigma$ die *Substitution* der Rolle einer Rollenmarke (auch *Rollenwechsel* genannt). Mit $\varphi$ bezeichnen wir solche Funktionen, welche durch Methoden der Zustandselemente im Sinne der Objektorientierung gegeben sind.

| Menge der *Funktionssymbole* ... | $\Phi = \Phi_{\hat{R}} \cup \Phi_{\hat{Z}}$ |
|---|---|
| *auf Rollen und Rollenmarken* | $\Phi_{\hat{R}} = \{\pi_i \mid \pi_i\colon \hat{RM} \to \hat{Z}, i\} \cup \{\varsigma \mid \varsigma\colon \hat{RM} \times \hat{R} \to \hat{RM}\} \cup \bigcup_{\hat{r} \in \hat{R}} \{\kappa_{\hat{r}} \mid \kappa_{\hat{r}}\colon \hat{S}_1 \times \ldots \times \hat{S}_{\lvert z(\hat{r})\rvert} \to \{\hat{r}\}; \hat{S}_i \text{ Sorte von } z(\hat{r})_i\}$ |
| *mit Typisierung in* $\hat{Z}$ | $\Phi_{\hat{Z}} = \{\varphi \mid \varphi\colon \hat{S}_1 \times \ldots \times \hat{S}_k \to \hat{S}_{k+1}; \hat{S}_i \in \hat{Z}\}$ |

Tabelle 3.2: Definition der Funktionssymbole des Interaktionsnetzes

Weiterhin sei $X$ die Menge der *typgebundenen Variablen*, aus der insbesondere *Individuenvariablen* des Typs $\hat{RM}$ und *Mengenvariablen* des Typs $2^{\hat{RM}}$ verwendet werden. Mit $Term(\Phi, X)$ sei wie üblich die Menge der *Terme* über $\Sigma$ mit freien Variablen aus $X$ definiert.

2. Die $\Sigma$-*Algebra* $A = (\hat{D}, \hat{F})$ sei ein Modell für die Signatur $\Sigma$, wobei $\hat{D} = \hat{D}_{\hat{R}} \cup \hat{D}_{\hat{RM}} \cup 2^{\hat{D}_{\hat{RM}}} \cup \hat{D}_{\hat{Z}}$ die *Grundmenge* der Sorten aus Tabelle 3.1 bezeichnet. Die *Interpretation der Funktionssymbole* auf Rollen $\Phi_{\hat{R}}$ und auf Zustände der Rollen $\Phi_{\hat{Z}}$ wird durch $\hat{F} = \hat{F}_{\hat{R}} \cup \hat{F}_{\hat{Z}}$ bezeichnet.

   Die Interpretation $\hat{F}_{\hat{R}}$ der Funktionssymbole auf Rollen aus $\Phi_{\hat{R}}$ ist dabei wie folgt (vgl. Tabelle 3.3): Die Interpretation der *Projektion* (in Zeichen $z(\rho)_i$) legen wir als den Zugriff auf die *i*-te Komponente des Zustandsvektors der Rollenmarke $\rho$ fest. Die Interpretation der *Komposition* (in Zeichen $R(\vec{z}_R)$) legen wir als die Instantiierung der Rolle *R* mit dem Zustandsvektor $\vec{z}_R$ fest. Die Interpretation der *Komposition-Substitution* (in Zeichen $\rho \leftarrow R(\overrightarrow{z_R})$) legen wir als die Zuweisung einer instantiierten Rolle *R* an eine existierende Rollenmarke $\rho$ fest. Die Rollenmarke $\rho$ wechselt die Rolle und übernimmt damit den durch $\overrightarrow{z_R}$ definierten Zustandvektor der neuen Rolle *R*. Die Zuordnung des Agenten zu der Marke wird von der Substitution nicht betroffen. Hingegen setzen wir die Existenz einer Abbildung voraus, welche im Fall der Substitution den neuen Namenskontext $\nu$ geeignet definiert.

| **Term** | **vor Anwendung** | **nach Anwendung** |
|---|---|---|
| *Projektion:* $z = \pi_i(\rho)$ | ./. | $z = z(r(\rho))[i]$ |
| *Komposition:* $r = \kappa_R(\overrightarrow{z_R})$ | ./. | $r = (D_{\mathrm{Ref}}(R), \overrightarrow{z_R}, \mathrm{ak}(R))$ |
| *Komposition-Substitution:* $\rho' = \varsigma(\rho, \kappa_R(\overrightarrow{z_R}))$ | $\rho = (r(\rho), ag, \nu_{ag})$, mit $r(\rho) = (D_{Ref}, \vec{z}, ak)$ | $\rho' = (r(\rho'), ag, \nu^R_{ag})$, mit $r(\rho') = (D_{\mathrm{Ref}}(R), \overrightarrow{z_R}, \mathrm{ak}(R))$ |

Tabelle 3.3: Interpretation der Funktionssymlole der $\Sigma$-Algebra

   Sei weiterhin $\beta$ eine *Belegung* der Variablen, welche über den Aufbau der Terme fortgesetzt wird. Wir interpretieren einen Term $\tau \in Term(\Phi, X)$ als *booleschen Term*, falls $\beta(\tau) \in \mathbb{B}$. Weiterhin bezeichnen wir einen Term $\rho' = \varsigma(\rho, \kappa_R(\overrightarrow{z_R}))$ als *Substitutionsterm.* Schließlich wird ein Term, welcher ausschließlich aus einer Individuenvariable *r* der Sorte $\hat{RM}_R$ oder aus einer Mengenvariable der Sorte $2^{\hat{RM}_R}$ besteht, als *Deklarationsterm* (in Zeichen $r\colon R$ oder $\forall r\colon R$) bezeichnet.

3. Ein *Interaktionsnetz* ist dann ein Tupel $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ so, daß gilt:

   - *P* ist eine endliche Menge von *Phasen* (Stellen),
   - *T* ist eine endliche Menge von Phasenübergängen (*Transitionen*),
   - $F \subset P \times T \cup T \times P$ ist die *Flußrelation*,

- $\sigma$ ist eine *Beschriftung* von Phasen, Kanten und Transitionen, wobei
  - jede Phase $p \in \hat{P}$ mit der Menge der Rollenmarken beschriftet ist $\sigma(p) = \bigcup_{\hat{r} \in \gamma(p)} \hat{RM_r}$, welche eine Rolle aus *p* verkörpern (Typisierung von *p*),
  - jede Kante $(p, t) \in F \cap P \times T$ mit einer endlichen Menge von ggf. mit einem Allquantor versehenen Deklarationstermen konsistent[1] beschriftet ist: $\sigma(p, t) = \{r_1 : R_1, \ldots, \forall r_i : R_i, \ldots\}$,
  - jede Kante $(t, p) \in F \cap T \times P$ mit einer endlichen Menge von Substitutionstermen $\sigma(t, p) = \{r_1 \leftarrow R_1(\overrightarrow{z_{R_1}}), \ldots\}$ konsistent[2] beschriftet ist, und
  - jede Transition $t \in T$ mit einem booleschen Term (Wächter) konsistent[3] beschriftet ist.
  - von $\sigma$ gefordert wird, daß das Schaltverhalten einer Transition markentreu ist, also die Anzahl und Identität der abgezogenen Marken der Anzahl und Identität der abgelegten Marken entspricht. Einzige Ausnahme sind Transitionen ohne Nachbereich, welche ausschließlich Marken abziehen.
- *A* ist ein Modell für die Signatur $\Sigma$
- $M_0 : P \rightarrow 2^{\hat{D}_{\hat{RM}}}$ ist die *Anfangsmarkierung*, die jeder Phase $p \in P$ eine Menge von Rollenmarken mit Rollen aus der Grundmenge der zu *p* gehörenden Rollenmenge $\gamma(p)$ zuweist.

Das Interaktionsnetz ist somit eine auf Pr/T-Netzen aufbauende Petrinetzklasse, welche um das Rollenkonzept erweitert ist.

In Abbildung 3.6 ist die Beschriftung einer Transition in einem Interaktionsnetz dargestellt.

1. Eine Kante $(p, t) \in F \cap P \times T$ heißt genau dann mit einer endlichen Menge von *Deklarationstermen* $\sigma(p, t) = \{r_1 : R_1, \ldots, \forall r_i : R_i, \ldots\}$ *konsistent beschriftet*, wenn die Deklaration jeder Variablen in der Transition eindeutig ist. Oder formal ausgedrückt, wenn für jeden Term $\tau \in \sigma(p, t), \tau = r_i : R_i$ oder $\tau = \forall r_i : R_i \; R_i \in \gamma(p)$ gilt, sowie $r_i \notin \sigma(q, t)$ für alle $q \in {\cdot t} \setminus \{p\}$ gilt.
2. Eine Kante $(t, p) \in F \cap T \times P$ heißt genau dann mit einer endlichen Menge von *Substitutionstermen* $\sigma(t, p) = \{r_1 \leftarrow R_1(\overrightarrow{z_{R_1}}), \ldots\}$ *konsistent beschriftet*, wenn jede verwendete Variable definiert ist und die ihr zugewiesene Rolle existiert. Oder formal ausgedrückt, wenn für jeden Substitutionsterm $\tau \in \sigma(t, p), \tau = r \leftarrow R(\overrightarrow{z_R}) \; R \in \gamma(p)$ gilt, sowie $\exists q \in {\cdot t}$, so daß $r \in \sigma(q, t)$.
3. Eine Transition $t \in T$ heißt genau dann mit einem booleschen Term $\tau = \sigma(t)$ *konsistent beschriftet*, wenn jede freie Variablen in $\tau$ durch genau einen Deklarationsterm $\tau' \in \sigma(p, t)$ mit $p \in {\cdot t}$ gebunden ist.

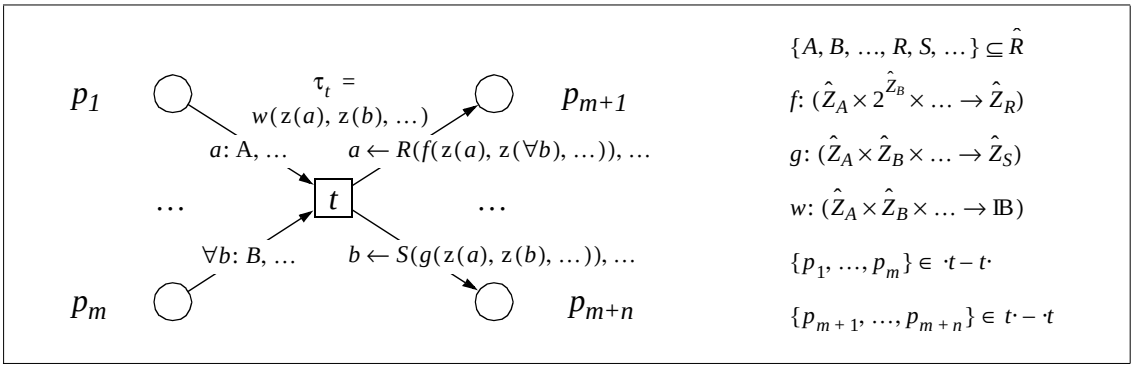


Abbildung 3.6: Beschriftungen der Schaltbedingung einer Transition (Interaktionsnetz)

Dabei bezeichne $\hat{Z}_A = \hat{S}_1^A \times \ldots \times \hat{S}_{n_{z(A)}}^A$ mit $\hat{S}_i^A = \{\hat{S} \mid \hat{S}$ Sorte von $z(A)_i\}$ die Sorte des Zustandsvektors von $A$. Weiterhin bezeichne $\hat{z}_a = z(r(\alpha))$ den Zustandsvektor der durch die Belegung $\beta$ an $a$ gebundenen Rollenmarke $\alpha = (a: A)[\beta]$.

**Bemerkung 3.6**

Es ist nicht vorgesehen, daß Transitionen Rollenmarken spalten oder verschmelzen. Das Konzept der Nebenläufigkeit wird bereits durch die von der Anfangsmarkierung erzeugten Rollenmarken modelliert. So kann ein Agent, vertreten durch mehrere Rollen, mehrfach an einem Interaktionsverfahren teilnehmen.

**Definition 3.8** *Anwendbarkeit einer Belegung*

Eine Belegung $\beta: Term(\Phi) \to \hat{D}$ heißt auf eine Transition $t$ *anwendbar*, wenn der Wächterterm $\tau_t$ nach *wahr* auswertet, d. h. $\beta(\tau_t)$ = wahr, und die Belegung $\beta(\forall r) \subseteq \hat{RM}$ jedes Deklarationsterms der Form $\forall r$: $R$ *maximal* ist (d. h. es gibt keine weitere Rollenmarke aus der aktuellen Markierung, die zu der Belegung des Deklarationsterms hinzugefügt werden könnte, so daß der Wächterterm weiterhin nach *wahr* auswertet).

Die *Schaltregel* für Interaktionsnetze ist dann entsprechend der Schaltregel von Pr/T-Netzen definiert (vgl. Definition 2.3). Ist eine Transition $t \in T$ $\beta$-aktiviert, so schaltet sie sofort.

Der Unterschied der Interaktionsnetze zu den Pr/T-Netzen besteht einerseits darin, daß durch die vorstrukturierte Signatur $\Sigma$ das Rollenkonzept verankert ist. Andererseits können in Interaktionsnetzen die Marken die Aktivierung von Transitionen durch Veränderung ihres Zustandes beeinflussen.

In herkömmlichen Petrinetzen gibt es keine Marken, die in Stellen aktiv sind. Führt man wie bei den Interaktionsnetzen Aktionen ein, so ist das Zusammenwirken zwischen den

Aktionen mit der Aktivierung von Transitionen und dem Schalten von Transitionen zu definieren:

**Definition 3.9** *Semantik des Schaltvorgangs hinsichtlich einer Rollenmarke*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz und $t \in T$ eine im Schaltmodus $t[\beta]$ aktivierte Transition. Sei ferner $p \in P$ eine Phase aus dem Vorbereich der Transition, $p \in {\cdot}t$, und $\tau \in \sigma(p, t)$ ein Deklarationsterm aus der Beschriftung der Kante $(p, t) \in F$. Nun schalte $t$ im Modus $t[\beta]$: Sei $\rho = \tau[\beta]$ eine Rollenmarke $\rho \in M(p)$, die am Schaltvorgang beteiligt ist. Der Schaltvorgang läuft in folgenden Stufen ab:

1. Deaktivieren der Rollenmarken:

   Ist die Aktion $\mathrm{ak}(\mathrm{r}(\rho))$ bereits terminiert, so heißt die Rollenmarke *deaktiviert*. Ist die Aktion noch nicht terminiert, so wird sie geordnet abgebrochen.

2. Schalten:

   Sei $\sigma(t, q)$ die Beschriftung einer Kante $(t, q) \in F$. Sei $\tau' \in \sigma(t, q)$ der Substitutionsterm, welcher den Rollenwechsel für die Rollenmarke $\rho$ angibt. Schaltet die Transition in dem Schritt $M[t[\beta] > M'$, so wird $\rho$ aus der Phase $p$ entfernt ($\rho \in M(p) - M'(p)$) und substituiert (vgl. Tabelle 3.3) in die Phase $q$ abgelegt ($\rho' \in M'(q) - M(q)$ mit $\rho' = \tau'[\beta]$).

3. Aktivieren der Rollenmarken:

   Die Aktion $\mathrm{ak}(\mathrm{r}(\rho'))$ jeder Marke aus $\{\rho' | \rho' = \tau'[\beta], \tau' \in \sigma(t, q)), q \in t{\cdot}\}$ wird gestartet. Eine Rollenmarke $\rho'$ heißt nun *aktiviert.*

**Bemerkung 3.7**

Das Deaktivieren einer Rollenmarke, also das geordnete Abbrechen einer noch laufenden Aktion, kann sinnvoll und notwendig sein, falls einem Interaktionsmechanismus ein Any-time-Algorithmus zu Grunde liegt. Aufgrund eines von außen kommenden Signals (Ablauf eines Time-out oder Auftritt eines Ausnahmefalls) kann dann die Interaktion auf dem erreichten Stand eingefroren werden.

Das geordnete Abbrechen einer Aktion erfordert auch das Abbrechen von Diensten, welche im zugeordneten Agenten aktiviert wurden. Dies muß beim Entwurf der Dienste vorgesehen worden sein. Läßt die Semantik einer Phase einen Abbruch laufender Aktionen nicht zu, so muß der Zustand *deaktiviert* aller am Schaltvorgang beteiligten Rollenmarken in den Wächterbedingungen der Transitionen zugesichert werden.

**Bemerkung 3.8**

Die Sofortschaltregel von Interaktionsnetzen und die Nebenläufigkeit der Rollenmarken erfordert besondere Vorkehrungen bei der Umsetzung der die Schaltsemantik realisierenden Ablaufsteuerung, da Zeitgleichheit in verteilten Systemen nicht definiert ist:

Jeder von einer Rolleninstanz ausgelöste, schreibende Zugriff auf einen Zustand kann zum Aktivieren einer Transition, aus dem Nachbereich der Phase in der die Rolleninstanz sich befindet, führen. Ein nebenläufig stattfindender Schreibzugriff könnte dazu führen, daß die Transition nicht mehr aktiviert wäre. Nun muß sichergestellt werden, daß die Schreibzugriffe und Schaltvorgänge von Transitionen für alle verteilten Instanzen der Ablaufsteuerung so serialisiert werden, daß zu jeder Zeit ein konsistenter Zustand des Interaktionsnetzes garantiert ist. Dies kann durch Einsatz von speziellen Broadcast-Protokollen, wie z. B. dem *Atomic Broadcast* erfolgen [ChMa 84].

Hierbei wird jeder schreibende Zugriff und jeder Schaltvorgang an alle Instanzen der Ablaufsteuerung gemeldet. Der Sender wartet jedoch mit der Ausführung der Aktion so lange, bis auch er seine eigene Meldung aus dem Empfangspuffer ließt. Das Protokoll garantiert, daß alle Instanzen die Nachrichten in der selben Reihenfolge erhalten. Ist beim Eintreffen der Nachricht die Vorraussetzung für die Ausführung der Aktion gegeben (Rollenmarke befindet sich noch in der Phase oder Transition ist noch aktiviert), so wird die Aktion bei jedem Empfänger ausgeführt. Andernfalls wird die Aktion von allen Empfängern gleichermaßen verworfen.

Abschließend definieren wir noch den Begriff eines terminierten und eines blockierten Interaktionsverfahren verstanden wird:

**Definition 3.10** *Terminieren eines Interaktionsverfahrens*

Ein Interaktionsverfahren heißt *terminiert*, wenn alle Rollenmarken das Interaktionsverfahren verlassen haben: $M(p) = \emptyset$, für alle Phasen $p$.

**Definition 3.11** *Blockieren eines Interaktionsverfahrens*

Ein Interaktionsverfahren heißt *blockiert*, wenn es nicht terminiert ist, alle Rolleninstanzen deaktiviert sind, jedoch keine Transition schalten kann: $\forall \rho \in M(p), p \in P$ gilt: $\rho$ ist deaktiviert und $\forall \beta, \forall t \in T$ gilt: $t$ ist nicht $\beta$-aktiviert.

### 3.3.4 Anforderungen der Rollenkette

Durch die Anfangsmarkierungen werden Startglieder einer Rollenkette definiert, welche den am Interaktionsverfahren beteiligten Agenten zugeordnet werden. Für jede Teil-

nahme wird eine Rollenmarke erzeugt, der eine Startrolle zugeordnet wird. Die Marken schalten anschließend durch das Netz und erhalten in jeder Phase neue Rollen zugeordnet. Um eine Rolle $r$ übernehmen zu können, muß ein Agent die von $r$ referenzierten Dienste $D_{Ref}$ unterstützen. Die Anforderung, die ein Agent erfüllen muß, damit er an einem Interaktionsverfahren teilnehmen kann, sind somit durch die referenzierten Dienste aller Rollen definiert, in die er während der Teilnahme an dem Verfahren wechseln wird. Hierfür definieren wir die Anforderungen aus der Erreichbarkeit von Rollen aus der Anfangsmarkierung.

**Definition 3.12** *Erreichbarkeit von Rollen in einem Schritt*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz, $p \in P$ eine Phase, $\rho \in M(p)$ eine Rollenmarke aus der Markierung von $p$ und $R = r(\rho)$ die Rolle von $\rho$. Sei $t \in p\cdot$ eine Transition im Nachbereich von $p$, für die ein Deklarationsterm $\tau \in \sigma(p, t)$ aus der Beschriftung der Kante von $p$ nach $t$ existiert, der Form $r\colon R$ oder $\forall r\colon R$.

Sei ferner $\beta$ eine Belegung, welche $\tau$ mit der Rollenmarke $\rho$ belegt ($\tau[\beta] = \rho$).

Sei $q \in t\cdot$ eine Phase mit $S \in \gamma(q)$, deren eingehende Kante $(t, q)$ mit einem Substitutionsterm $\tau' \in \sigma(t, q)$ beschriftet ist, so daß $S = r(\tau'[\beta])$ die Rolle bezeichnet, in die $\rho$ beim Schalten der Transition unter der Belegung $\beta$ wechseln würde.

Dann heißt *S in einem Schritt* von $\rho$, $R = r(\rho)$ aus *erreichbar.* Die Menge dieser Rollen wird als $E^1(R)$ bezeichnet.

**Definition 3.13** *Erreichbarkeit von Rollen*

Die transitive Hülle von $E^1(R)$ wird mit $E^*(R)$ bezeichnet und gibt die Rolle an, die von $R$ aus erreichbar sind.

**Bemerkung 3.9**

$E^1(R)$ ist über die syntaktische Struktur des Interaktionsnetzes definiert und vernachlässigt laufzeitabhängige Einflüsse, wodurch die Erreichbarkeit berechenbar ist.

Den durch die Relation $E^1(R)$ erzeugten, gerichteten Graphen bezeichnen wir als *Rollenkette*. Während einer Teilnahme durchläuft ein Agent den Graphen auf einem Pfad von einer Start- zu einer Endrolle. Die Entscheidung, welchen Pfad er nimmt, wird durch den lokalen Zustand (Markierungen) des Petrinetzes an jedem Verzweigungspunkt bestimmt.

Nun läßt sich die Anforderung definieren, die von einem Agenten erfüllt sein muß, damit er eine durch die Rollenkette definierte Teilnahme an einer Interaktion übernehmen kann.

**Definition 3.14** *übernehmbar*

Eine Rollenmarke $\rho \in M_o$ aus der Anfangsmarkierung heißt für einen Agenten übernehmbar, falls der Agent $ag \supseteq \{d \mid d \in \mathrm{D}_{\mathrm{Ref}}(r), r \in E^*(\mathrm{r}(\rho))\}$ jeden referenzierten Dienst einer von $\rho$ aus erreichbaren Rolle implementiert.

### 3.3.5 Beispiel einer Auktion als Interaktionsverfahren

Zur Veranschaulichung soll anhand eines Beispiels verdeutlicht werden, wie ein Interaktionsverfahren innerhalb des Interaktionsmodells der ersten Iterationsstufe spezifiziert wird. Dem Beispiel liegt ein Abstimmungsmechanismus zugrunde, wie er in Auktionen angewendet wird: Zur Vereinfachung wird angenommen, daß genau ein Auktionsgegenstand versteigert werden soll. Es gibt einen ausgezeichneten Agenten, welcher die Aufgaben eines Auktionators übernimmt und eine endliche Menge von Auktionsteilnehmern, die an dem Auktionsgegenstand interessiert sind. Die Kardinalität der Menge der Auktionsteilnehmern wird mit *N* bezeichnet.

Die Preisbildung findet dadurch statt, daß der Auktionator den Gegenstand mit einem Mindestgebot aufruft. Auktionsteilnehmer geben durch Handheben ein Gebot ab, welches vom Auktionator wahrgenommen wird und zu einer inkrementellen Erhöhung des Preises führt. Werden innerhalb einer Zeitspanne, die in drei Phasen gegliedert ist, keine weiteren Gebote gemacht, so erhält das letzte Gebot den Zuschlag. Den Abschluß einer dieser Phasen wird durch ein “akustisches” Signal (Hammerschlag) signalisiert, welches einen Phasenübergang bei allen Teilnehmern auslöst.

| **Dienst** | **Dienstkontext** | **Dienstprimitive** |
|---|---|---|
| **Handel-Verkäufer** | *Gegenstand:* Ware<br>*Mindestpreis:* Geld<br>... | *(*Ware, Geld*) anbieten (*void*)*<br>void *abschließen (*Geld, Agent*)*<br>... |
| **Handel-Käufer** | *Gegenstand:* Ware<br>*Höchstpreis:* Geld<br>... | Geld *bewerten (*Ware*)*<br>bool *bieten (*Countdown, Geld*)*<br>void *scheitern (*Geld*)*<br>void *abschließen (*Geld, Agent*)* |
| **Timer** | *Time-out:* Zeit | void *setzen (*Zeit*)*<br>void *löschen (*void*)* |

Tabelle 3.4: Ausschnitt aus der Dienststruktur

Gegeben sei die in Tabelle 3.4 angegebene Dienststruktur, welche von einem gedachten Agentenmodell definiert wird, und die Schnittstelle zwischen dem Agentenmodell und dem Interaktionsverfahren bildet. Ferner seien Agenten vorhanden, welche die Dienste Handel-Verkäufer und Timer beziehungsweise Handel-Käufer bereitstellen. Die Agenten

haben das Ziel Ware gegen Geld zu tauschen und wählen zur Preisbildung den Interaktionsmechanismus in Form einer Auktion.

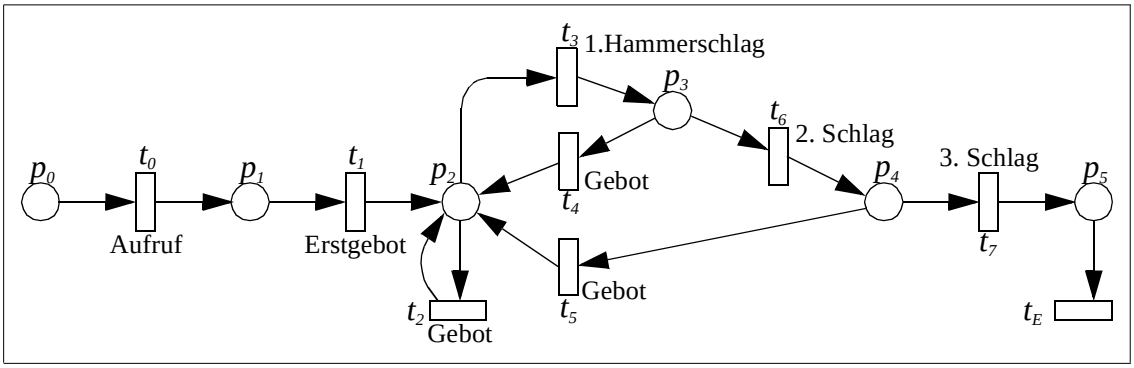


Abbildung 3.7: Netzstruktur des Interaktionsverfahren Auktion

Die Auktion sei durch ein Interaktionsnetz $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ beschrieben, wobei die Netzstruktur wie in Abbildung 3.7 dargestellt gegeben ist. Die Beschriftung der Phasen sei wie in Tabelle 3.5 angegeben und die der Kanten und Transitionen wie in Tabelle 3.6. Die Angabe der Signatur entfällt aus Gründen der Übersichtlichkeit. Die $\Sigma$-Algebra sei gegeben durch die übliche Interpretation der Sorten und Funktionssymbole. Die Anfangsmarkierung sei wie folgt:

$$M_0 = \begin{cases} \left\{(I_0, ag_{Auktionator}, \nu_{ag}^{I_0})\right\} \cup \left\{(A_{0'}, ag_j, \nu_{ag}^{A_0}) \,\middle|\, 1 \le j \le N\right\} & \text{für } p_0, \\ \varnothing & \text{sonst.} \end{cases}$$

Damit ist das Interaktionsverfahren spezifiziert. Der Ablauf des Verfahrens erfolgt entsprechend der Semantik des Schaltvorgangs hinsichtlich der Rollenmarken und der Schaltregel für Pr/T-Netze.

Der eilige Leser sei auf den Markentransport, auf das erzeugte gemeinsame Wissen und auf den Einsatz der Rollen in dem Beispiel hingewiesen:

1. Das Interaktionsverfahren ist ein gutes Beispiel für eine synchron ablaufende Interaktion. Alle Marken betreten das Interaktionsnetz (definiert durch die Anfangsmarkierung) in $p_0$. Von dort aus werden jeweils alle Marken gemeinsam von einer Phase in eine Folgephase geschaltet.

2. Durch die durch den Schaltvorgang herbeigeführte Synchronisierung wird sichergestellt, daß das jeweilige Gebot allen Teilnehmern bekannt ist. Sollte ein Teilnehmer seine Aktion in der vorangegangenen Phase noch nicht beendet haben, so wird diese abgebrochen. Durch Starten der neuen Rolle und ihrer Aktion geht das Wissen (in die-

| **Phase** | **Rollen mit denen eine Phase $p_i$ beschriftet ist** |
|---|---|
| $p_0$, $\gamma(p_0) = \{I_0, A_0\}$ | $I_0 = (D_{Ref} = \{\text{Handel-Verkäufer}\}, \vec{z} = (\mathit{Gegenstand}, \mathit{Mindestgebot}),$ $ak = <(\mathit{Gegenstand}, \mathit{Mindestgebot}) = \text{Handel-Verkäufer.anbieten}() >)$<br>$A_0 = (D_{Ref} = \emptyset, \vec{z} = (\emptyset), ak = \emptyset)$ |
| $p_1$, $\gamma(p_1) = \{I_1, A_1\}$ | $I_1 = (D_{Ref} = \{\emptyset\}, \vec{z} = (\mathit{Mindestgebot}), ak = \emptyset)$<br>$A_1 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{Gegenstand}, \mathit{Mindestgebot}, \mathit{Gebot}),$ $ak = <\textbf{if } (\text{Handel-Käufer.bewerten}(\mathit{Gegenstand}) \geq \mathit{Mindestgebot}) \textbf{ then}$ $\mathit{Gebot} = \text{true}$ $\textbf{fi} >)$ |
| $p_2$, $\gamma(p_2) = \{I_2, B_2, A_2\}$ | $I_2 = (D_{Ref} = \{\text{Timer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{letzterBieter}, \mathit{TimeOut}, \mathit{Hammer}),$ $ak = <\mathit{Hammer} = \text{Timer.setzten}(\mathit{TimeOut}) >)$<br>$B_2 = (D_{Ref} = \emptyset, \vec{z} = \emptyset, ak = \emptyset)$<br>$A_2 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{Inkrement}, \mathit{Gebot}),$ $ak = <\textbf{if } (\text{Handel-Käufer.bieten}(2, \mathit{letztesGebot}(1 + \mathit{Inkrement}))) \textbf{ then}$ $\mathit{Gebot} = \text{true};$ $\mathit{letztesGebot} = \mathit{letztesGebot} * (1 + \mathit{Inkrement})$ $\textbf{fi} >)$ |
| $p_3$, $\gamma(p_3) = \{I_3, B_3, A_3\}$ | $I_3 = (D_{Ref} = \{\text{Timer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{letzterBieter}, \mathit{TimeOut}, \mathit{Hammer}),$ $ak = <\mathit{Hammer} = \text{Timer.setzten}(\mathit{TimeOut}) >)$<br>$B_3 = (D_{Ref} = \emptyset, \vec{z} = \emptyset, ak = \emptyset)$<br>$A_3 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{Inkrement}, \mathit{Gebot}),$ $ak = <\textbf{if } (\text{Handel-Käufer.bieten}(1, \mathit{letztesGebot}(1 + \mathit{Inkrement}))) \textbf{ then}$ $\mathit{Gebot} = \text{true};$ $\mathit{letztesGebot} = \mathit{letztesGebot} * (1 + \mathit{Inkrement})$ $\textbf{fi} >)$ |
| $p_4$, $\gamma(p_4) = \{I_4, B_4, A_4\}$ | $I_4 = (D_{Ref} = \{\text{Timer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{letzterBieter}, \mathit{TimeOut}, \mathit{Hammer}),$ $ak = <\mathit{Hammer} = \text{Timer.setzten}(\mathit{TimeOut}) >)$<br>$B_4 = (D_{Ref} = \emptyset, \vec{z} = \emptyset, ak = \emptyset)$<br>$A_4 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{letztesGebot}, \mathit{Inkrement}, \mathit{Gebot}),$ $ak = <\textbf{if } (\text{Handel-Käufer.bieten}(0, \mathit{letztesGebot}(1 + \mathit{Inkrement}))) \textbf{ then}$ $\mathit{Gebot} = \text{true};$ $\mathit{letztesGebot} = \mathit{letztesGebot} * (1 + \mathit{Inkrement})$ $\textbf{fi} >)$ |
| $p_5$, $\gamma(p_5) = \{I_5, B_5, A_5\}$ | $I_5 = (D_{Ref} = \{\text{Handel-Verkäufer}\}, \vec{z} = (\mathit{Preis}, \mathit{Käufer}),$ $ak = <\text{Handel-Verkäufer.abschließen}(\mathit{Preis}, \mathit{Käufer}) >)$<br>$B_5 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{Preis}, \mathit{Verkäufer}),$ $ak = <\text{Handel-Käufer.abschließen}(\mathit{Preis}, \mathit{Verkäufer}) >)$<br>$A_5 = (D_{Ref} = \{\text{Handel-Käufer}\}, \vec{z} = (\mathit{Preis}),$ $ak = <\text{Handel-Käufer.scheitern}(\mathit{Preis}) >)$ |

Tabelle 3.5: Rollendefinitionen für das Interaktionsverfahren Auktion

| T | **Beschriftungen der In-Kanten, des Wächters und der Out-Kanten einer Transition $t_i$; Kanten sind jeweils mit Mengen von Termen beschriftet** |
|---|---|
| $t_0$ | $\sigma(p_0, t_0) = \{i\colon I_0, \forall a_j\colon A_0\}$<br>$\sigma(t_0) = \text{Deaktiviert}(i)$<br>$\sigma(t_0, p_1) = \{i \leftarrow I_1(i.Mindestgebot), a_j \leftarrow A_1(i.Gegenstand, i.Mindestgebot, \text{false})\}$ |
| $t_1$ | $\sigma(p_1, t_1) = \{i\colon I_1, a_1\colon A_1, \forall a_j\colon A_1\}$<br>$\sigma(t_1) = a_1.Gebot \wedge \text{Deaktiviert}(a_1) \wedge \text{Deaktiviert}(\forall a_j)$<br>$\sigma(t_1, p_2) = \{i \leftarrow I_2(i.Mindestgebot, a_1, 5s, \text{false}), a_1 \leftarrow B_2(\emptyset),$<br>$a_j \leftarrow A_2(i.Mindestgebot, 1/10, \text{false})\}$ |
| $t_2$ | $\sigma(p_2, t_2) = \{i\colon I_2, b\colon B_2, a_1\colon A_2, \forall a_j\colon A_2\}$<br>$\sigma(t_2) = a_1.Gebot$<br>$\sigma(t_2, p_2) = \{i \leftarrow I_2(a_1.letztesGebot, a_1, 5s, \text{false}), a_1 \leftarrow B_2(\emptyset),$<br>$a_j \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false}),$<br>$b \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false})\}$ |
| $t_3$ | $\sigma(p_2, t_3) = \{i\colon I_2, b\colon B_2, \forall a_j\colon A_2\}$<br>$\sigma(t_3) = i.Hammer$<br>$\sigma(t_3, p_3) = \{i \leftarrow I_3(i.letztesGebot, b, 5s, \text{false}), b \leftarrow B_3(\emptyset),$<br>$a_j \leftarrow A_3(i.letztesGebot, 1/10, \text{false})\}$ |
| $t_4$ | $\sigma(p_3, t_4) = \{i\colon I_3, b\colon B_3, a_1\colon A_4, \forall a_j\colon A_3\}$<br>$\sigma(t_4) = a_1.Gebot$<br>$\sigma(t_4, p_2) = \{i \leftarrow I_2(a_1.letztesGebot, a_1, 5s, \text{false}), a_1 \leftarrow B_2(\emptyset),$<br>$a_j \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false}),$<br>$b \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false})\}$ |
| $t_5$ | $\sigma(p_4, t_5) = \{i\colon I_4, b\colon B_4, \forall a_j\colon A_4\}$<br>$\sigma(t_5) = a_1.Gebot$<br>$\sigma(t_5, p_2) = \{i \leftarrow I_2(a_1.letztesGebot, a_1, 5s, \text{false}), a_1 \leftarrow B_2(\emptyset),$<br>$a_j \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false}),$<br>$b \leftarrow A_2(a_1.letztesGebot, 1/10, \text{false})\}$ |
| $t_6$ | $\sigma(p_3, t_6) = \{i\colon I_3, b\colon B_3, \forall a_j\colon A_3\}$<br>$\sigma(t_6) = i.Hammer$<br>$\sigma(t_6, p_4) = \{i \leftarrow I_4(i.letztesGebot, b, 5s, \text{false}), b \leftarrow B_4(\emptyset),$<br>$a_j \leftarrow A_4(i.letztesGebot, 1/10, \text{false})\}$ |
| $t_7$ | $\sigma(p_4, t_7) = \{i\colon I_4, b\colon B_4, \forall a_j\colon A_4\}$<br>$\sigma(t_7) = i.Hammer$<br>$\sigma(t_7, p_5) = \{i \leftarrow I_5(i.letztesGebot, \text{ag}(b)), b \leftarrow B_5(i.letztesGebot, \text{ag}(i)),$<br>$a_j \leftarrow A_5(i.letztesGebot)\}$ |
| $t_E$ | $\sigma(p_5, t_E) = \{i\colon I_5, b\colon B_5, \forall a_j\colon A_5\}$<br>$\sigma(t_E) = \text{Deaktiviert}(i) \wedge \text{Deaktiviert}(b) \wedge \text{Deaktiviert}(\forall a_j)$ |

Tabelle 3.6: Beschriftung $\sigma$ für das Interaktionsverfahren Auktion

sem Fall das Gebot) unmittelbar in die Handlungsweise der neuen Rolle ein. Somit liegt zu Beginn jeder Phase gemeinsames Wissen über das letzte Gebot vor (Abschnitt 3.2.5).

3. Wie aus Tabelle 3.5 ersichtlich, gibt es die Rollen *Auktionator* (*I* für Initiator), und *Auktionsteilnehmer* (*A*). Sobald das erste Gebot (ab Phase $p_2$) von einem Teilnehmer abgegeben wurde (Schalten der Transition $t_1$), existiert die Rolle des letzten *Bieters* (*B*). Beim Schalten der Transitionen $\{t_1, t_2, t_4, t_5\}$ (siehe Tabelle 3.6) finden jeweils zwei Rollenwechsel statt: Der Auktionsteilnehmer, dessen Gebot gewählt wurde, erhält in der nachfolgenden Phase die Rolle des Bieters zugewiesen, und der bisherige Bieter (den es bei $t_1$ jedoch noch nicht gibt) wird wieder zu einem Auktionsteilnehmer.

Weiterhin sei darauf hingewiesen, daß es durch die vorgenommene Spezifikation der Rolle Bieter sichergestellt ist, daß ein Agent sein eigenes Gebot nicht überbietet.

Technisch betrachtet, finden bei jedem Schaltvorgang für jede transportierte Marke ein Rollenwechsel statt. Der Rollenwechsel kann zu einem Wechsel zur selben Rolle entarten. Die neue Rolle wird mit dem an der Kantenbeschriftung spezifizierten Zustandsvektor initialisiert und mit ihrer Aktion gestartet. In dem Beispiel wäre es möglich die Rollen $\{I_2, I_3, I_4\}$ und $\{B_2, B_3, B_4\}$ zu jeweils einer Rolle zusammenzufassen und die Rollen $\{A_2, A_3, A_4\}$ zu vereinheitlichen.

Um die Dienstschnittstelle der Agenten von dem Interaktionsverfahren weitgehend unabhängig zu machen, dem Agenten aber dennoch die Nutzung einer Bietstrategie zu ermöglichen, wurde ein Countdown eingeführt, welcher in der Phase vor dem letzten Hammerschlag auf Null steht.

### 3.3.6 Diskussion

Der vorgestellte Ansatz, welcher nur die Rollenabstraktion unterstützt, hat eine Reihe von Beschränkungen, welche nun im Einzelnen diskutiert werden. Die hieraus gewonnenen Erkenntnisse fließen in die nächsten Iterationsstufen des Interaktionsmodells ein.

1. Informationsfluß zwischen Rollen:

   Das Interaktionsmodell bietet in der ersten Iterationstufe als einzigen, expliziten Mechanismus den Phasenübergang, um einen Informationsaustausch und damit eine Interaktion zwischen Rolleninstanzen zu modellieren. Beim Schalten einer Transition werden die Rollenmarken in nachfolgende Phasen transportiert, in denen sie entsprechende Rollen annehmen. Die Zustände dieser Rollen werden durch Substitutionsterme initialisiert, welche unter anderem auch aus Komponenten der Zustandvektoren anderer beim Schaltvorgang beteiligter Instanzen gebildet werden. Im Beispiel interagieren die Instanzen durch die sich gegenseitig überbietenden Gebote. So wird durch

die Schaltbedingung der Transition $t_2$ (vgl. Tabelle 3.6) das Gebot des aktuellen Bieters verwendet ($a_1.Inkrement$) um die Komponente *letztesGebot* aller Instanzen zu initialisieren.

Ein weiterer impliziter Mechanismus, um eine Interaktion zu modellieren, ist der, die Rolleninstanzen über die Fähigkeiten der ihnen zugeordneten Agenten interagieren zu lassen. Im Beispiel wird dies in der Phase $p_5$ (vgl. Tabelle 3.5) durch den Aufruf der Dienstprimitive "Handel-Verkäufer.abschließen(*Preis*, *Käufer*)" und der Dienstprimitive "Handel-Käufer.abschließen(*Preis*, *Verkäufer*)" spezifiziert. Dieser Mechanismus ist für Fälle vorgesehen, in denen wie hier Handlungen in der physische Welt ausgelöst werden. Dieser Mechanismus könnte jedoch dazu mißbraucht werden, z. B. Kommunikation und damit Interaktion zwischen einzelnen Rolleninstanzen zu modellieren. Hiermit würde jedoch das Ziel der agentenunabhängigen Modellierung von Interaktionsmechanismen nicht erreicht werden, da Agenten für ein Interaktionsverfahren spezielle Interaktionsdienste realisieren müßten. Besser wäre es, hierfür ein eigenes Interaktionsverfahren zu definieren.

In der zweiten Iterationsstufe wird durch die Einführung des Objektkonzepts den Rollen ein Mechanismus zur Verfügung gestellt, mit dem sie über eine Dienstschnittstelle, unabhängig vom Schalten einer Transition miteinander kommunizieren können. Die hierzu notwendigen Ressourcen werden von außerhalb des Interaktionsverfahrens eingebracht.

2. Mehrfachdefinition von Rollen:

   Der kritische Leser wird festgestellt haben, daß in der Darstellung des Beispiels, in der die Rollen jeweils innerhalb der einzelnen Phasen definiert sind, und somit Mehrfachdefinitionen von Rollen erzeugt werden (vgl. Rollen $I_2, I_3, I_4$ aus Tabelle 3.5). Dies ist für das Beispiel richtig, jedoch nicht für das Modell. Im Modell werden Rollen in der Algebra definiert und den Phasen zugeordnet. Somit ist eine Zuordnung einer Rolle zu mehreren Phasen möglich.

3. Keine Modularisierung:

   Petrinetze bieten keinen Mechanismus zur Modularisierung. Statt dessen werden Konzepte verwendet, um Stellen oder Transitionen einer höheren Abstraktionsstufe mit Netzausschnitten einer niedrigerer Abstrakionsstufe verfeinern zu können. Logisch entsteht jedoch ein Netz. Es wird also ein Ersetzungsmechanismus statt dem aus Programmiersprachen bekannten Mechanismus des Bindens verwendet. Wir werden in der dritten Itearationsstufe einen Ansatz zur Modularisierung einführen.

4. Rollen aus der Anfangsmarkierung:

   Zumindest für die Übernahme einer Rolle aus der Anfangsmarkierung wird ein Mechanismus benötigt, welcher die Fähigkeiten eines Agenten auf die Anforderungen an Diensten überprüft.

   Allgemeiner wird ein Mechanismus zur Rollenauswahl und Rollenzuordnung benötigt. Wir werden auf den Mechanismus nach Einführung der Modularisierung eingehen.

5. Beschränkungen durch das Konzept der Anfangsmarkierung

   Im Interaktionsmodell der ersten Iterationsstufe werden die Marken durch die Anfangsmarkierung erzeugt. Dies hat eine Reihe von Beschränkungen:

   - Formal sind die Interaktionsnetze, welche ein und das selbe Interaktionsverfahren beschreiben, jedoch unterschiedliche Anfangsmarkierungen besitzen, verschieden. Die Petrinetze müßten einen Mechanismus erhalten, mit denen sie hinsichtlich ihrer Anfangsmarkierung parametrisierbar wären.

   - Ein Interaktionsverfahren kann erst starten, nachdem alle in der Anfangsmarkierung aufgeführten Marken (welche jeweils eine Teilnahme eines Agenten symbolisieren) angelegt sind. Bricht die Verbindung zu einem einzelnen Agenten ab, scheitert damit auch die Initialisierung des Interaktionsnetzes. Dies wäre auch dann der Fall, wenn die Teilnahme des Agenten für die Interaktion nicht notwendig ist (z. B. kann die Auktion auch mit einem Teilnehmer weniger stattfinden).

   - Zum Zeitpunkt des Startens eines Interaktionsverfahrens müßten alle beteiligten Agenten, sowie ihre Rollenzuordnung feststehen. Ein nachträglicher Eintritt eines Agenten in einer Interaktion ist in der jetzigen Iterationsstufe nicht möglich.

Mit der Anfangsmarkierung werden wir uns detailliert in der dritten Iterationsstufe befassen, wenn wir das Modulkonzept einführen. Wir werden sehen, daß das Konzept der Anfangsmarkierung für die Modellierung von Teilnahmen an einem Interaktionsverfahren zu starr ist.

## 3.4 Interaktionsmodell mit Objekten

Im Abschnitt 3.3 wurde das Interaktionsmodell mit Rollen vorgestellt, auf deren Grundlage nun Erweiterungen eingeführt werden, um die mit dem Ansatz verbundenen Beschränkungen schrittweise zu beheben. Das Interaktionsmodell wird in dieser zweiten Iterationsstufe um das Objektkonzept erweitert, welches von passiven Anwendungsobjekten abstrahiert.

In einer Interaktion stehen Objekte den Rollen wie Requisiten den Charakteren in einer Szene eines Theaterstücks zur Verfügung. Ein Requisit kann von ein oder mehreren Charaktären verwendet werden, kann zu Beginn der Szene vorhanden sein, oder erst nachträglich hinzukommen. Es kann zum Informationsaustauch genutzt werden, wie z. B. ein Brief oder ein Telephon und ist im Allgemeinen für die ablaufende Interaktion ein wesentlicher Bestandteil.

### 3.4.1 Ressourcenmodell

Dem Rollenkonzept liegt das Verständnis zu Grunde, daß sich eine Gruppe von Agenten einem Interaktionsmechanismus unterordnen und Teile ihrer Entscheidungsfreiheit an eine Rolleninstanz abtreten. Auf das Objektkonzept soll diese Sicht nun übertragen werden. In einem System existieren neben Agenten auch passive Anwendungsobjekte auf welchen Agenten einzeln agieren oder über die sie miteinander interagieren. Diese Anwendungsobjekte bezeichnen wir als Ressourcen eines Multiagentensystems. Der Unterschied zu den *Agenten* besteht darin, daß Ressourcen ihre Funktionalität in ein Interaktionsverfahren einbringen, dort aber im Gegensatz zu den Agenten nicht Rollen annehmen, die als Subjekt handeln, sondern als *Objekte* den Rollen entsprechend des Verfahrens zur Verfügung stehen.

Analog zu der Definition 3.3 der Agenten definieren wir die Ressource wie folgt:

**Definition 3.15** *Ressource*

Eine Ressource *rs* wird von einer Menge von Operationen oder *Diensten* beschrieben, welche die Schnittstelle zu der Funktionalität der Ressource darstellt. Mit $RS$ bezeichnen wir die Menge aller Ressourcen.

**Bemerkung 3.10**

Im Gegensatz zu den Diensten, welche Agenten unterstützen, besitzen die Dienste von Ressourcen keine Bewertungs- und Entscheidungsfunktionen. Da die Ressourcen aus Sicht des Interaktionsmodells Objekte darstellen, haben sie ihre Entscheidungsbefugnis hinsichtlich ihrer Verwendung mit der Teilnahme an einem Interaktionsverfahren abgetreten. Die Entscheidung, ob sie an einem speziellen Verfahren teilnehmen wollen, kann ihnen jedoch nicht genommen werden.

### 3.4.2 Interaktionsobjekte (Objektkonzept)

Der Begriff eines Objekts ist die Abstraktion von einer spezifischen Ressource, welche in einem bestimmten Bereich eines Interaktionsverfahrens von Rollen genutzt werden kann.

**Definition 3.16** *Objekt*

Ein Objekt wird durch das Tupel $o = (D_{Def}, D_{Ref}, D_{Impl}, \dot{z})$ beschrieben. Dabei ist $D_{Def} \subseteq D$ die Menge der von dem Objekt *bereitgestellten Dienste*, $D_{Ref} \subseteq D$ die Menge der von dem Objekt *referenzierten Ressourcendienste*, $D_{Impl} \subseteq D$ die *Implemen-*

*tierung* der Menge der bereitgestellten Dienste und $\vec{z}$ ein Vektor von (*beobachteten*) *Zuständen*. Wir bezeichnen mit $D_{Def}(o)$ die Menge der bereitgestellten Dienste, mit $D_{Ref}(o)$ die Menge der referenzierten Dienste, mit $D_{Impl}(o)$ die Menge der Implementierungen und mit $z(o)$ den Vektor $\vec{z}$ der durch das Objekt *o* (beobachteten) Zustände. Mit $O$ bezeichnen wir die Menge aller Objekte.

**Bemerkung 3.11**

Ein Objekt stellt eine phasenspezifische Sicht auf eine Ressource dar. Ein Objekt ist selbst zustandslos. Im Zustandvektor $\vec{z}$ sind Zustände der Ressource modelliert, welche das Objekt beobachtet. Die Ressource stellt das Modell dar, in dem die Zustände gespeichert sind.

Analysiert man das Objekt hinsichtlich der benutzten Entwurfsmuster [GHJV 94], so entspricht der Zustandvektor dem Beobachtermuster (observer pattern) und die bereitgestellten Dienste entsprechen dem Strategiemuster (strategy pattern).

Analog zum Rollenkonzept definieren wir eine Objektmarke als die Komposition aus einer Ressource und einem Objekt:

**Definition 3.17** *Objektmarke*

Eine Objektmarke ist als ein Tripel $\omega = (o, rs, \nu)$ definiert, wobei $o \in O$ ein Objekt, $rs \in RS$ eine Ressource und $\nu: D_{Ref}(o) \rightarrow rs$ den Namenskontext definiert, welcher den referenzierten Diensten von *o* die Funktionalität der Ressource zuordnet. Wir bezeichnen mit $o(\omega)$ das Objekt der Marke.

Die Struktur des Objektbegriffs in Verbindung mit dem Rollenbegriff veranschaulicht Abbildung 3.8. Dabei symbolisieren die Kreise Dienste, welche von der mit einer durchgezogenen Linie verbundenen Instanz definiert werden und von der mit einer unterbrochenen Linie verbundenen Instanz genutzt werden.

**Bemerkung 3.12**

Objekte existieren in zwei Ausprägungen, der des *1-Benutzer-Objekts* und der des *n-Benutzer-Objekts*. Unter einem 1-Benutzer-Objekt wird die Abstraktion von einer Ressource verstanden, auf die genau eine Rolle gleichzeitig Zugriff hat. Dies kann beispielsweise eine Transaktion auf einer Datenbank sein. Unter einem *n*-Benutzer-Objekt wird die Erweiterung von einem 1-Benutzer-Objekt verstanden, so daß *n* verschiedene Benutzer Zugriff haben. Beispiele für *n*-Benutzer-Objekte sind Teamobjekte nach [Rüd 92] oder Blackboards.

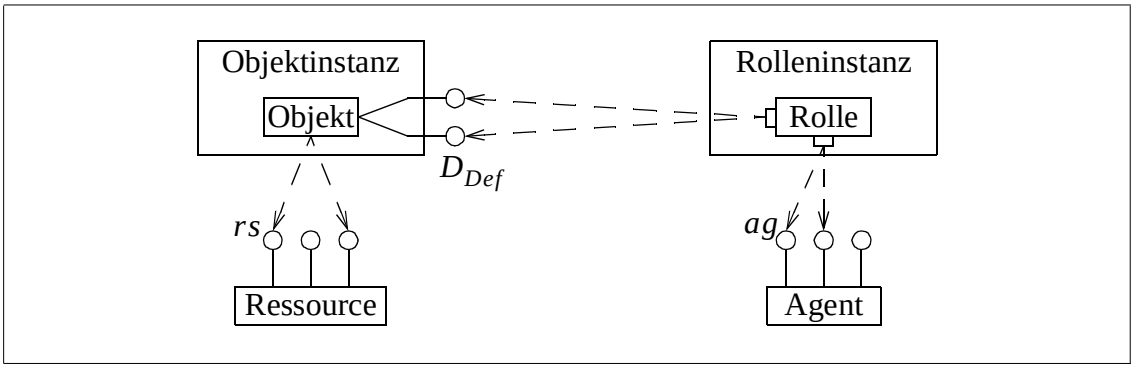


Abbildung 3.8: Struktur des Objekt- und des Rollenkonzepts

### 3.4.3 Interaktionsnetz mit Rollen- und Objektkonzept

Das Interaktionsmodell der zweiten Iterationsstufe ist gegenüber dem Interaktionsmodell der ersten Iterationsstufe um Objekte erweitert, welche zur Spezifikation der Interaktion auf Ressourcen verwendet werden. Hierzu erweitern wir (vgl. Abschnitt 3.3.3) zuerst die Rollen um Dienstnamen, welche auf Dienste von Objekten verweisen. Anschließend wird die Phasen um Objekte erweitert und danach die Signatur des Interaktionsnetzes um Terme auf Objekten und um Terme auf Rollen und Objekten erweitert.

**Definition 3.18** *Rolle*

Eine Rolle $r$ wird durch das Tripel $r = (D^A_{Ref}, D^O_{Ref}, \vec{z}, ak)$ beschrieben. Dabei ist $D^A_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Agentendienste*, $D^O_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Objektdienste*, $\vec{z}$ ein Vektor von *Zuständen* und *ak* eine *Aktion*. Die Aktion *ak* besteht aus Dienstprimitiven aus $D^A_{Ref} \cup D^O_{Ref}$ und Operationen auf Zustände (Komponenten) aus $\vec{z}$. Wir bezeichnen mit $\mathrm{D}^A_{\mathrm{Ref}}(r)$ und $\mathrm{D}^O_{\mathrm{Ref}}(r)$ die Menge der referenzierten Dienste, mit $\mathrm{z}(r)$ den Vektor $\vec{z}$ der Zustände beziehungsweise mit $\mathrm{ak}(r)$ die Aktion der Rolle *r*. Mit $R$ bezeichnen wir die Menge aller Rollen.

**Definition 3.19** *Rollenmarke*

Eine Rollenmarke ist als ein Tripel $\rho = (r, ag, \nu)$ definiert, wobei $r \in R$ eine *Rolle*, $ag \in A$ einen *Agenten* und $\nu$: $(\mathrm{D}^A_{\mathrm{Ref}}(r) \to ag) \cup (\mathrm{D}^O_{\mathrm{Ref}}(r) \to \mathrm{D}_{\mathrm{Def}}(o))$ den *Namenskontext* definiert, welcher den von der Aktion der Rolle referenzierten Diensten die

Fähigkeiten des Agenten zuordnet bzw. die von einem Objekt bereitgestellten Dienste zuordnet, auf welche die Rollen Zugriff hat. Wir bezeichnen mir $r(\rho)$ die Rolle der Marke.

**Definition 3.20** *Phase*

Wir definieren nun eine Phase *p* als einen Schritt in einem Interaktionsverfahren. Sei *P* die Menge der Phasen in einem Interaktionsverfahren. Dann definieren wir eine injektive Abbildung $\gamma$: $P \rightarrow 2^{R \cup O}$, welche jede Phase mit der Menge von Rollen und Objekten beschriftet, die an ihr beteiligt sind. Wir bezeichnen mit $\gamma_p$ die Rollen- und Objektmenge der Phase *p*. Weiterhin bezeichnen wir die Restriktion von $\gamma$ auf die Menge der Rollen mit $\gamma|_R$ und mit $\gamma|_O$ die Restriktion auf die Menge der Objekte.

**Bemerkung 3.13**

Im folgenden bezeichnen $\hat{S}, \hat{R}, \hat{ZR}, \ldots$ die syntaktische Menge von Sorten, Rollen, Zuständen der Rollen etc. Damit unterscheiden wir die Komponenten der Syntax von außerhalb des Interaktionsnetzes definierten Begriffen.

**Definition 3.21** *Interaktionsnetz*

Ein Interaktionsnetz ist ein Pr/T-Netz, welches um das Rollen- und das Objektkonzept erweitert ist:

Kurze Erinnerung an die Definition: $P$ = Menge der Phasen, $\gamma(p) \subseteq R \cup O$, $\gamma|_R(p) \subseteq R$, $\gamma|_O(p) \subseteq O$ Rollen und Objekte der Phase, $z(r)$, $z(o)$ = Zustandsvektor, $\rho$ = Rollenmarke, $r(\rho)$ = Rolle der Marke, $\omega$ = Objektmarke, $o(\omega)$ = Objekt der Marke. Mit $|\vec{z}|$ wird die Länge eines Vektors $\vec{z}$ bezeichnet.

1. Die Erweiterung wird über die Signatur $\Sigma = (\hat{S}, \Phi)$ wie folgt definiert: Wir definieren die Menge der *Sorten* $\hat{S}$ wie in Tabelle 3.7.
   Weiterhin definieren wir die Menge der *Funktionssymbole* $\Phi$ wie in Tabelle 3.8. Dabei bezeichnet $\pi_i$ die *Projektion* auf die *i*-te Komponente des Zustandsvektors, mit $\kappa$ bezeichnen wir die *Komposition* einer Rolle bzw. eines Objektes und mit $\varsigma$ die *Substitution* der Rolle einer Rollenmarke (auch Rollenwechsel genannt), respektive des Objekts einer Objektmarke (auch Objektwechsel genannt). Mit $\nu$ bezeichnen wir den *Namenskontext* zwischen referenzierenden Diensten einer Rolle und definierenden Diensten eines Objektes für ein Paar aus Rollen- und Objektmarke. Mit $\varphi$ bezeichnen wir solche Funktionen, welche durch Methoden der Zustandselemente im Sinne der Objektorientierung gegeben sind.

| | |
|---|---|
| Menge der *Sorten* ... | $\hat{S} = (\hat{R} \cup \hat{ZR} \cup \hat{RM} \cup 2^{\hat{RM}} \cup \hat{DN}) \cup (\hat{O} \cup \hat{ZO} \cup \hat{OM} \cup 2^{\hat{OM}} \cup \hat{DF})$ |
| *der Rollen* | $\hat{R} = \bigcup_{p \in P} \gamma\|_R(p)$ |
| *der Zustände der Rollen* | $\hat{ZR} = \bigcup_{\hat{r} \in \hat{R}} \{\hat{S} \mid \hat{S}$ Sorte von $z(\hat{r})_i, 1 \le i \le \|z(\hat{r})\|\}$ |
| *der Rollenmarken* | $\hat{RM}_{\hat{r}} = \{\rho \mid r(\rho) = \hat{r}\}$, $\hat{RM} = \bigcup_{\hat{r} \in \hat{R}} \hat{RM}_{\hat{r}}$ |
| *der Namen der von den Rollen referenzierten Objektdienste* | $\hat{DN} = \bigcup_{\hat{r} \in \hat{R}} D^{O}_{Ref}(\hat{r})$ |
| *der Objekte* | $\hat{O} = \bigcup_{p \in P} \gamma\|_O(p)$ |
| *der Zustände der Objekte* | $\hat{ZO} = \bigcup_{\hat{o} \in \hat{O}} \{\hat{S} \mid \hat{S}$ Sorte von $z(\hat{o})_i, 1 \le i \le \|z(\hat{o})\|\}$ |
| *der Objektmarken* | $\hat{OM}_{\hat{o}} = \{\omega \mid o(\omega) = \hat{o}\}$, $\hat{OM} = \bigcup_{\hat{o} \in \hat{O}} \hat{OM}_{\hat{o}}$ |
| *der von den Objekten bereitgestellten Dienste* | $\hat{DF} = \bigcup_{\hat{o} \in \hat{O}} D_{Def}(\hat{o})$ |

Tabelle 3.7: Definition der Sorten des Interaktionsnetzes

Weiterhin sei $X$ die Menge der *typgebundenen Variablen*, aus der insbesondere *Individuenvariablen* des Typs $\hat{RM}$ bzw. $\hat{OM}$ und *Mengenvariablen* des Typs $2^{\hat{RM}}$ bzw. $2^{\hat{OM}}$ verwendet werden. Mit $Term(\Phi)$ sei wie üblich die Menge der *Terme* über $\Sigma$ definiert.

| | |
|---|---|
| Menge der *Funktionssymbole* ... | $\Phi = \Phi_{\hat{R}} \cup \Phi_{\hat{O}} \cup \Phi_{\hat{RMOM}} \cup \Phi_{\hat{Z}}$ |
| *auf Rollen und Rollenmarken* | $\Phi_{\hat{R}} = \{\pi_i \mid \pi_i: \hat{RM} \to \hat{Z}_{\hat{R}}, i\} \cup \{\varsigma \mid \varsigma: \hat{RM} \times \hat{R} \to \hat{RM}\} \cup \bigcup_{\hat{r} \in \hat{R}} \{\kappa_{\hat{r}} \mid \kappa_{\hat{r}}: (\hat{S}_1 \times \ldots \times \hat{S}_{\|z(\hat{r})\|}) \to \{\hat{r}\}; \hat{S}_i$ Sorte von $z(\hat{r})_i, \forall i\}$ |
| *auf Objekten und Objektmarken* | $\Phi_{\hat{O}} = \{\pi_i \mid \pi_i: \hat{OM} \to \hat{Z}_{\hat{O}}, i\} \cup \{\varsigma \mid \varsigma: \hat{OM} \times \hat{O} \to \hat{OM}\} \cup \bigcup_{o \in \hat{O}} \{\kappa_{\hat{o}} \mid \kappa_{\hat{o}}: \hat{S}_1 \times \ldots \times \hat{S}_{\|z(\hat{o})\|} \to \hat{o}; \hat{S}_i$ Sorte von $z(\hat{o})_i\}$ |
| *auf Rollen- und Objektmarken* | $\Phi_{\hat{RMOM}} = \{\nu \mid \nu: \hat{RM} \times \hat{OM} \to \hat{RM}\}$ |
| *mit Typisierung in* $\hat{Z}$ | $\Phi_{\hat{Z}} = \{\varphi \mid \varphi: \hat{S}_1 \times \ldots \times \hat{S}_k \to \hat{S}_{k+1}; \hat{S}_i \in \hat{Z}_{\hat{R}} \cup \hat{Z}_{\hat{O}}\}$ |

Tabelle 3.8: Definition der Funktionssymbole des Interaktionsnetzes

2. Die $\Sigma$*-Algebra* $A = (\hat{D}, \hat{F})$ sei ein Modell für die Signatur $\Sigma$, wobei $\hat{D}$ die *Grundmenge* entsprechend Tabelle 3.9 bezeichnet. Die *Interpretation der Funktionssymbole* auf Rollen und Rolleninstanzen ($\Phi_{\hat{R}}$ wird interpretiert von $\hat{F}_{\hat{R}}$), auf Objekten und

Objektinstanzen ($\Phi_{\hat{O}}$ wird interpretiert von $\hat{F}_{\hat{O}}$), auf Rollen- und Rolleninstanzen ($\Phi_{\hat{R}\hat{M}\hat{O}\hat{M}}$ wird interpretiert von $\hat{F}_{\hat{R}\hat{M}\hat{O}\hat{M}}$) und auf Zustände der Rollen und der Objekte ($\Phi_{\hat{Z}}$ wird interpretiert von $\hat{F}_{\hat{Z}}$) bezeichnen wir mit $\hat{F} = \hat{F}_{\hat{R}} \cup \hat{F}_{\hat{O}} \cup \hat{F}_{\hat{R}\hat{M}\hat{O}\hat{M}} \cup \hat{F}_{\hat{Z}}$.

| *Grundmenge ...* | $\hat{D} = (\hat{D}_{\hat{R}} \cup \hat{D}_{\hat{Z}\hat{R}} \cup \hat{D}_{\hat{R}\hat{M}} \cup \hat{D}_{2^{\hat{R}\hat{M}}} \cup \hat{D}_{\hat{D}\hat{N}}) \cup (\hat{D}_{\hat{O}} \cup \hat{D}_{\hat{Z}\hat{O}} \cup \hat{D}_{\hat{O}\hat{M}} \cup \hat{D}_{2^{\hat{O}\hat{M}}} \cup \hat{D}_{\hat{D}\hat{F}})$ |
|---|---|
| *der Rollen* | $\hat{D}_{\hat{R}} = \bigcup_{\hat{r} \in \hat{R}} D_{\hat{r}}$ |
| *der Zustände der Rollen* | $\hat{D}_{\hat{Z}\hat{R}} = \bigcup_{\hat{zr} \in \hat{Z}\hat{R}} D_{\hat{zr}}$ |
| *der Rollenmarken* | $\hat{D}_{\hat{R}\hat{M}} = \bigcup_{\hat{rm} \in \hat{R}\hat{M}} D_{\hat{rm}}$ |
| *der Dienstnamen* | $\hat{D}_{\hat{D}\hat{N}} = \bigcup_{\hat{dn} \in \hat{D}\hat{N}} D_{\hat{dn}}$ |
| *der Objekte* | $\hat{D}_{\hat{O}} = \bigcup_{\hat{o} \in \hat{O}} D_{\hat{o}}$ |
| *der Zustände der Objekte* | $\hat{D}_{\hat{Z}\hat{O}} = \bigcup_{\hat{zo} \in \hat{Z}\hat{O}} D_{\hat{zo}}$ |
| *der Objektmarken* | $\hat{D}_{\hat{O}\hat{I}} = \bigcup_{\hat{om} \in \hat{O}\hat{M}} D_{\hat{om}}$ |
| *der Dienste* | $\hat{D}_{\hat{D}\hat{F}} = \bigcup_{\hat{df} \in \hat{D}\hat{F}} D_{\hat{df}}$ |

Tabelle 3.9: Grundmenge der $\Sigma$-Algebra

Die Interpretation $\hat{F}_{\hat{R}}$ und $\hat{F}_{\hat{O}}$ der Funktionssymbole auf Rollen und Objekten ist dabei wie folgt (vgl. Tabelle 3.10): Die Interpretation der *Projektion* legen wir als den Zugriff auf die *i*-te Komponente des Zustandsvektors der Rollen $\rho$ bzw. des Objekts $\omega$ fest. Die Interpretation der *Komposition* legen wir als die Instantiierung der Rolle *R* bzw. des Objekts *O* fest. Die Interpretation der *Substitution* mit einer *Komposition* legen wir als die Zuweisung einer Rolle *R* bzw. Objekt *O* an eine existierende Rollenmarke $\rho$ bzw. Objektmarke $\omega$ fest. Die Interpretation des *Kontextes* legen wir fest als eine Abbildung $\nu$: $DN(\mathrm{r}(\rho)) \rightarrow DF(\mathrm{o}(\omega))$ von der Menge der referenzierenden Dienste der Rolle einer Rollenmarke auf die Menge der definierenden Dienste des Objekts einer Objektmarke.

Sei weiterhin $\beta$ eine *Belegung* der Variablen, welche über den Aufbau der Terme fortgesetzt wird. Wir interpretieren einen Term $\tau \in Term(\Phi)$ als *booleschen Term*, falls $\beta(\tau) \in \mathbb{B}$. Weiterhin bezeichnen wir einen Term $\rho' = \varsigma(\rho, \kappa_R(\overrightarrow{z_R}))$ bzw. $\omega' = \varsigma(\omega, \kappa_O(\overrightarrow{z_O}))$ als *Substitutionsterm* und einen Term der Form $\rho' = \nu(\rho, \omega)$ als *Kontextterm*. Schließlich wird ein Term, welcher ausschließlich aus einer Individuen-

| **Term** | **vor Ausführung** | **nach Ausführung** |
|---|---|---|
| *Projektion:*<br>$z = \pi_i(\rho)$<br>$z = \pi_i(\omega)$ | ./. | $z = \mathrm{z}(r(\rho))[i]$<br>$z = \mathrm{z}(o(\omega))[i]$ |
| *Komposition:*<br>$r = \kappa_R(\vec{z_R})$<br>$o = \kappa_O(\vec{z_O})$ | ./. | $r = (\mathrm{D}^A_{\mathrm{Ref}}(R), \mathrm{D}^O_{\mathrm{Ref}}(R), \vec{z_R}, \mathrm{ak}(R))$<br>$o = (\mathrm{D}_{\mathrm{Def}}(O), \mathrm{D}_{\mathrm{Ref}}(O), \mathrm{D}_{\mathrm{Impl}}(O), \vec{z_O})$ |
| *Substitution* mit einer *Komposition:*<br>$\rho' = \varsigma(\rho, \kappa_R(\vec{z_R}))$<br>$\omega' = \varsigma(\omega, \kappa_O(\vec{z_O}))$ | $\rho = (\mathrm{r}(\rho), ag, \nu_{ag})$, mit<br>$\mathrm{r}(\rho) = (D^A_{Ref}, D^O_{Ref}, \vec{z}, ak)$ | $\rho' = (\mathrm{r}(\rho'), ag, \nu^R_{ag})$, mit<br>$\mathrm{r}(\rho') = (\mathrm{D}^A_{\mathrm{Ref}}(R), \mathrm{D}^O_{\mathrm{Ref}}(R), \vec{z_R}, \mathrm{ak}(R))$ |
| | $\omega = (\mathrm{o}(\omega), rs, \nu_{rs})$, mit<br>$\mathrm{o}(\omega) = (D_{Def}, D_{Ref}, D_{Impl}, \vec{z})$ | $\omega' = (\mathrm{o}(\omega'), rs, \nu^O_{rs})$, mit<br>$\mathrm{o}(\omega') = (\mathrm{D}_{\mathrm{Def}}(O), \mathrm{D}_{\mathrm{Ref}}(O), \mathrm{D}_{\mathrm{Impl}}(O), \vec{z_O})$ |
| *Kontext:*<br>$\rho' = \nu(\rho, \omega)$ | $\rho = (r, ag, \nu_{ag})$ | $\rho' = (r, ag, \nu_{ag} \cup \nu(\rho, \omega))$ |

Tabelle 3.10: Interpretation der Funktionssymbole der $\Sigma$-Algebra

variable $r$ der Sorte $\hat{RM}$ bzw. $o$ der Sorte $\hat{OM}$ oder aus einer Mengenvariable der Sorte $2^{\hat{RM}}$ bzw. der Sorte $2^{\hat{OM}}$ besteht, als *Deklarationsterm* bezeichnet (in Zeichen $r\colon R$ bzw. $o\colon O$ oder $\forall r\colon R$ bzw. $\forall o\colon O$).

3. Ein *Interaktionsnetz* ist dann ein Tupel $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ so, daß gilt:

   - $P$ ist eine endliche Menge von *Phasen*,
   - $T$ ist eine endliche Menge von Phasenübergängen (*Transitionen*),
   - $F \subset P \times T \cup T \times P$ ist die *Flußrelation*,
   - $\sigma$ ist eine *Beschriftung* von Phasen, Kanten und Transitionen, wobei

     - jede Phase $p \in P$ mit der Menge der Rollen- bzw. Objektmarken beschriftet ist, welche eine Rolle bzw. ein Objekt aus $p$ verkörpern (Typisierung von $p$):
       $\sigma(p) = \bigcup_{\hat{r} \in \gamma|_R(p)} RM_{\hat{r}} \cup \bigcup_{\hat{o} \in \gamma|_O(p)} OM_{\hat{o}}$
     - jede Kante $(p, t) \in F \cap P \times T$ mit einer endlichen Menge von ggf. mit einem Allquantor versehenen Deklarationstermen konsistent[1] beschriftet ist:
       $\sigma(p, t) = \{r_1\colon R_1, \ldots, \forall r_i\colon R_i, o_1\colon O_1, \ldots, \forall o_j\colon O_j, \ldots\}$

1. Eine Kante $(p, t) \in F \cap P \times T$ heißt genau dann mit einer endlichen Menge von *Deklarationstermen* $\sigma(p, t) = \{r_1\colon R_1, \ldots, \forall r_i\colon R_i, o_1\colon O_1, \ldots, \forall o_j\colon O_j, \ldots\}$ *konsistent beschriftet*, wenn die Deklaration jeder Variablen in der Transition eindeutig ist. Oder formal ausgedrückt, wenn für jeden Term $\tau \in \sigma(p, t), \tau = x\colon \mathrm{X}$ oder $\tau = \forall x\colon \mathrm{X}$ $X \in \gamma(p)$ gilt, sowie $x \notin \sigma(q, t)$ für alle $q \in \cdot t \backslash \{p\}$ gilt.

- jede Kante $(t, p) \in F \cap T \times P$ mit einer endlichen Menge von Termen konsistent[1] beschriftet ist:

$$\sigma(t,p) = \{r_k \leftarrow R_k(\overrightarrow{z_{R_k}}) \mid k\} \cup \{o_l \leftarrow O_l(\overrightarrow{z_{O_l}}) \mid l\} \cup \{\nu(Dn(r)) := Df(o) \mid r, o\}$$

- und jede Transition $t \in T$ mit einem booleschen Term (Wächter) konsistent[2] beschriftet ist.
- von $\sigma$ gefordert wird, daß das Schaltverhalten einer Transition hinsichtlich der Rollenmarken markentreu ist. D. h. die Anzahl und Identität der abgezogenen Rollenmarken entspricht beim Schalten einer Transition der Anzahl und Identität der abgelegten Rollenmarken. Im Gegensatz dazu dürfen Objektmarken dupliziert und konsumiert werden. Transitionen ohne Nachbereich ziehen ausschließlich Marken ab und können somit auch Rollenmarken konsumieren.

- $A$ ist ein Modell für die Signatur $\Sigma$
- $M_0 \colon P \to 2^{\hat{D}_{\hat{RM}} \cup \hat{D}_{\hat{OM}}}$ ist die *Anfangsmarkierung*, die jeder Phase $p \in P$ eine Menge von Rollen- und Objektmarken aus der Grundmenge der zu $p$ gehörenden Rollen- und Objektmarken $\sigma(p)$ zuweist.

Das Interaktionsnetz ist somit eine auf Pr/T-Netzen aufbauende Petrinetzklasse, welche um das Rollen- und das Objektkonzept erweitert ist.

**Bemerkung 3.14**

Es ist nicht vorgesehen, daß Transitionen Rollenmarken spalten oder verschmelzen. Das Konzept der Nebenläufigkeit wird bereits durch die von der Anfangsmarkierung erzeugten Rollenmarken modelliert. So kann ein Agent, vertreten durch mehrere Rollen, mehrfach an einem Interaktionsverfahren teilnehmen.

Im Gegensatz dazu ist es Transitionen möglich Objektmarken zu dupliziert. Zwar ist das für Agenten beschriebene Konzept der Nebenläufigkeit auch für Ressourcen gleichermaßen verfügbar, jedoch wird dieses den besonderen Entwurfsanforderungen hinsichtlich der Datenabstraktion nur eingeschränkt gerecht. Die Ressource, von der in Form einer

1. Eine Kante $(t, p) \in F \cap T \times P$ heißt genau dann mit einer endlichen Menge von Termen $\sigma(t, p) = \{r_k \leftarrow R_k(\overrightarrow{z_{R_k}}) \mid k\} \cup \{o_l \leftarrow O_l(\overrightarrow{z_{O_l}}) \mid l\} \cup \{\nu(r, o) \mid r, o\}$ *konsistent beschriftet*, wenn alle verwendeten Variablen auch definiert sind und ihnen ein definierter Typ zugewiesen wird, ferner wenn Kontextterme dieser Kante nur auf Marken definiert sind, welche über diese Kante geschaltet werden. Oder formal ausgedrückt, wenn für jeden Substitutionsterm $\tau \in \sigma(t, p), \tau = x \leftarrow X(\overrightarrow{z_X}) \quad X \in \gamma(p)$ gilt, sowie $\exists q \in \cdot t$, so daß $x \in \sigma(q, t)$ und wenn für jeden Kontextterm $\tau \in \sigma(t, p), \tau = \nu(Dn(r)) := Df(o)$ sowohl $r$ als auch $o$ auf der linken Seite eines Substitutionsterms aus $\sigma(t, p)$ vorkommen.
2. Eine Transition $t \in T$ heißt genau dann mit einem booleschen Term $\tau = \sigma(t)$ *konsistent beschriftet*, wenn alle verwendeten Variablen auch definiert sind. Oder formal ausgedrückt, wenn jede freie Variablen in $\tau$ durch genau einen Deklarationsterm $\tau' \in \sigma(p, t)$ mit $p \in \cdot t$ gebunden ist.

Objektmarke abstrahiert wurde, kann entweder ein einzelnes Dokument in einer Interaktion einbringen, oder einen beliebig komplex strukturierten Behälter mit zugeordneten Dokumenten repräsentieren. Ein Interaktionsverfahren kann es nun erfordern, daß die Dokumente einzeln bearbeitet werden, was auf der Objektabstraktion die Abspaltung einer Objektmarke entspricht.

Die *Schaltregel* für Interaktionsnetze entspricht der Schaltregel von Pr/T-Netzen (vgl. Definition 2.3), jedoch ist die Definition der Anwendbarkeit einer Belegung auf eine Transition hinsichtlich des Objektkonzeptes wie folgt zu erweitern:

**Definition 3.22** *Anwendbarkeit einer Belegung*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz, $t \in T$ eine Transition und $\tau_t = \sigma(t)$ die Wächterbedingung von *t*. Eine Belegung $\beta : Term(\Phi) \rightarrow \hat{D}$ heißt in folgenden Fällen auf die Transition *t anwendbar*:

1. Alle in *t* eingehenden Kanten sind ausschließlich mit Deklarationstermen für Rollenmarken beschriftet ($\forall p \in {\cdot}t, \forall \tau \in \sigma(p, t) \quad \tau \in X_{\hat{RM}}$). Eine Belegung $\beta$ heißt dann auf die Transition *t anwendbar*, wenn $\tau_t$ unter der Algebra *A* nach *wahr* auswertet und die Belegung jedes Deklarationsterms der Form $(\forall r)$: *R* bzw. $(\forall o)$: *O maximal* ist (es gibt keine weitere Marke aus der aktuellen Markierung, die zu der Belegung des Deklarationsterms hinzugefügt werden könnte, so daß der Wächterterm weiterhin nach *wahr* auswertet).

2. Sei $\omega$ eine Objektmarke, mit der ein Deklarationsterm einer eingehenden Kante belegt wird ($p \in {\cdot}t, \tau \in \sigma(p, t), \tau \in X_{\hat{OM}}, \beta(\tau) = \omega$). Eine Belegung $\beta$ heißt $\omega$-*blockiert*, wenn es eine nicht am Schaltvorgang beteiligte, aktivierte Rollenmarke gibt ($\exists \rho \in M(p)$, $\rho \notin \{\rho \mid \rho = \beta(\tau), \tau \in \sigma(p, t)\}$ und $ak(r(\rho))$ nicht terminiert), welche Dienste der Objektmarke nutzt ($\exists Dn \in D^{O}_{Ref}(\rho)$, $\exists Df \in D_{Def}(\omega)$: $(Dn, Df) \in \nu(\rho)$). Eine Belegung $\beta$ heißt dann auf die Transition *t anwendbar*, wenn $\tau_t$ unter der Algebra *A* zutrifft und durch keine Objektmarke $\omega$-*blockiert* ist.

Unter dieser Erweiterung wird die Schaltregel für Interaktionsnetze analog zu der von Pr/T-Netzen (vgl. Definition 2.3) definiert. Ist eine Transition $t \in T$ $\beta$-aktiviert, so schaltet sie sofort.

**Bemerkung 3.15**

Führt man bei Objekten einen speziellen Zustand ein, welcher die Menge der Rollenmarken verwaltet, die durch einen Namenskontext an eine Objektmarke gebunden sind, sieht man ferner in jedem Objekt einen Dienst vor, in dem sich Rollenmarken von einer Objektmarke abmelden können und fordert zuletzt von jeder Rollenmarke, daß sie sich vor einer Deaktivierung bei den von ihr benutzten Objektmarken abmeldet, so läßt sich für alle an einer $\beta$-Aktivierung beteiligten Objektmarken die fehlende $\omega$-Blockierung in der Wächterbedingung der Transition zusichern.

Somit stellt die Erweiterung der Schaltregel keine qualitative Erweiterung der verwendeten Petrinetzklasse dar.

Der Unterschied der Interaktionsnetze zu den Pr/T-Netzen besteht einerseits darin, daß durch die vorstrukturierte Signatur $\Sigma$ das Rollen- und Objektkonzept verankert ist. Andererseits können in Interaktionsnetzen die Marken die Aktivierung von Transitionen beeinflussen. Wie für Interaktionsnetze muß nun noch die Semantik des Schaltvorgangs für die Erweiterungen definiert werden:

**Definition 3.23** *Semantik des Schaltvorgangs hinsichtlich Rollen- und Objektmarken*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz und $t \in T$ eine im Schaltmodus $t[\beta]$ aktivierte Transition. Sei ferner $p, \tilde{p} \in P$ eine Phase aus dem Vorbereich der Transition $p, \tilde{p} \in {}^{\cdot}t$ und $(\tau = r\colon R) \in \sigma(p, t)$ bzw. $(\tilde{\tau} = o\colon O) \in \sigma(\tilde{p}, t)$ ein Deklarationsterm aus der Beschriftung der Kante $(p, t) \in F$ bzw. $(\tilde{p}, t) \in F$. Nun schalte $t$ im Modus $t[\beta]$: Sei $\rho = \beta(r)$ eine Rollenmarke und $\omega = \beta(o)$ eine Objektmarke, $\rho \in M(p)$ und $\omega \in M(\tilde{p})$, die am Schaltvorgang beteiligt sind. Der Schaltvorgang läuft in folgenden Stufen ab:

1. Deaktivieren der Rollenmarken:

   Ist die Aktion $\mathrm{ak}(\mathrm{r}(\rho))$ bereits terminiert, so heißt die Rollenmarke *deaktiviert*. Ist die Aktion noch nicht terminiert, so wird sie geordnet abgebrochen.

   Nach Abschluß dieses Schrittes gibt es keine Rollenmarke mehr, welche an Dienste des Objekts $\omega$ gebunden ist.

2. Schalten:

   Sei $\sigma(t, q)$ bzw. $\sigma(t, \tilde{q})$ die Beschriftung einer Kante $(t, q) \in F$ bzw. $(t, \tilde{q}) \in F$. Sei $(\tau' = r \leftarrow R'(\overrightarrow{z_{R'}})) \in \sigma(t, q)$ der Substitutionsterm, welcher den Rollenwechsel für Rollenmarke $r$ spezifiziert und $(\tilde{\tau}' = o \leftarrow O'(\overrightarrow{z_{O'}})) \in \sigma(t, \tilde{q})$ der Substitutionsterm, welcher den Objektwechsel für Objektmarke $o$ spezifiziert. Nun schalte die Transition in dem Schritt $M[t[\beta] > M'$:

- $\rho$ wird aus der Phase $p$ entfernt ( $\rho \in M(p) - M'(p)$ )

  $\omega$ wird aus der Phase $\tilde{p}$ entfernt ( $\omega \in M(\tilde{p}) - M'(\tilde{p})$ )
- $\rho$ wird substituiert ($\rho' = \tau'[\beta]$, vgl. Tabelle 3.10)

  $\omega$ wird substituiert ($\omega' = \tilde{\tau}'[\beta]$)
- Ist $q = \tilde{q}$ und gibt es einen Kontextterm $\tau_v' \in \sigma(t, q)$ $\tau_v = \nu(r, o)$, so wird der Namenskontext von $\rho$ erweitert ($\rho' = \tau_v'[\beta]$)
- $\rho'$ wird in die Phase $q$ abgelegt ($\rho' \in M'(q) - M(q)$)

  $\omega'$ wird in die Phase $\tilde{q}$ abgelegt ($\omega' \in M'(\tilde{q}) - M(\tilde{q})$)

3. Aktivieren der Rollenmarken:

   Die Aktion $\mathrm{ak}(\mathrm{r}(\rho'))$ jeder Marke $\rho' \in \{\rho' | \rho' = \tau'[\beta], \forall \tau' \in \sigma(t, q)), \forall q \in t\cdot\}$ wird gestartet. Eine Rollenmarke $\rho'$ heißt nun *aktiviert*.

Das Terminieren und das Blockieren eines Interaktionsverfahrens ist wie in Definition 3.10 und Definition 3.11 definiert.

### 3.4.4 Anforderungen der Objektkette

Analog zu den Anforderungen den eine Rollenkette an einen Agenten stellt, ergeben sich Anforderungen aus einer Objektkette an eine Ressource. Diese Anforderungen müssen erfüllt sein, bevor die der Kette zugeordnete Teilname an einem Interaktionsverfahren möglich ist.

**Definition 3.24** *Erreichbarkeit von Objekten in einem Schritt*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz, $p \in P$ eine Phase, $\omega \in M(p)$ eine Objektmarke aus der Markierung von $p$ und $O = \mathrm{o}(\omega)$ das Objekt von $\omega$. Sei $t \in p\cdot$ eine Transition im Nachbereich von $p$, für die ein Deklarationsterm $\tau \in \sigma(p, t)$ aus der Beschriftung der Kante von $p$ nach $t$ existiert, der Form $o\colon O$ oder $\forall o\colon O$.

Sei ferner $\beta$ eine Belegung, welche $\tau$ mit der Rollenmarke $\omega$ belegt ($\tau[\beta] = \omega$).

Sei $q \in t\cdot$ eine Phase mit $S \in \gamma|_O(q)$, deren eingehende Kante $(t, q)$ mit einem Substitutionsterm $\tau' \in \sigma(t, q)$ beschriftet ist, so daß $S = \mathrm{o}(\tau'[\beta])$ die Rolle bezeichnet, in die $\omega$ beim Schalten der Transition unter der Belegung $\beta$ wechseln würde.

Dann heißt $S$ *in einem Schritt* von $\omega$, $O = \mathrm{o}(\omega)$ aus *erreichbar*. Die Menge dieser Rollen wird als $E^1(O)$ bezeichnet.

**Definition 3.25** *Erreichbarkeit von Objekten*

Die transitive Hülle von $E^1(O)$ wird mit $E^*(O)$ bezeichnet und gibt die Rolle an, die von $O$ aus erreichbar sind.

**Bemerkung 3.16**

$E^1(O)$ ist über die syntaktische Struktur des Interaktionsnetzes definiert und vernachlässigt laufzeitabhängige Einflüsse, wodurch die Erreichbarkeit berechenbar wird.

Den durch die Relation $E^1(O)$ erzeugten, gerichteten Graphen bezeichnen wir als *Objektkette*. Während einer Teilnahme durchläuft eine Ressource den Graphen auf einem Pfad von einem Start- zu einem Endobjekt. Die Entscheidung, welchen Pfad sie nimmt, wird durch den lokalen Zustand (Markierungen) des Petrinetzes an jedem Verzweigungspunkt bestimmt.

Nun läßt sich die Anforderung definieren, die von einer Ressource erfüllt sein muß, damit es eine durch die Objektkette definierte Teilnahme an einer Interaktion übernehmen kann.

**Definition 3.26** *übernehmbar*

Eine Objektmarke $\omega \in M_o$ aus der Anfangsmarkierung heißt für eine Ressource übernehmbar, falls die Ressource $rs \supseteq \{d | \ d \in \mathrm{D}_{\mathrm{Ref}}(o), o \in E^*(\mathrm{o}(\omega))\}$ jeden referenzierten Dienst eines von $\omega$ aus erreichbaren Objekts implementiert.

### 3.4.5 Erweiterung des Auktionsbeispiels

Zur Veranschaulichung soll nun das Beispiel der Auktion aus Abschnitt 3.3.5 so verallgemeinert werden, daß nun auch eine Menge von Gegenstände, welche in Form eines Kataloges vorliegen, versteigert werden. Das Objektkonzept wird nun dazu genutzt, sowohl den Katalog zu modellieren, in dem die einzelnen Teilnehmer zu Beginn blättern, als auch den einzelnen Versteigerungsgegenstand, welcher dem Katalog entnommen wird. Es fällt schnell auf, daß der Versteigerungsgegenstand als solcher für die Interaktion nicht von Belang ist. Ist der Agent von dem aktuell zur Versteigerung anstehenden Gegenstand informiert, so läuft die Auktion ohne weiteren Zugriff auf den Gegenstand ab. Somit kann der Entwurf aus Abschnitt 3.3.5 für die eigentliche Auktion weitgehend übernommen werden. Das Verfahren terminiert, sobald keine Auktionsgegenstände mehr zur Versteigerung anstehen.

| Dienstmodul | Dienstschnittstelle | Dienstkontext | Dienstprimitive |
|---|---|---|---|
| **Handel** | **Ware** | *Mindestpreis:* Geld<br>*Eigentümer:* Agent<br>*Nr:* Katalognr | void *übereignen(*Geld, Agent*)* |
| | **Aggregat** | | Iterator *erzeugeIterator(*void*)* |
| | **Iterator** | ... | void *start(*void*)*<br>void *weiter(*void*)*<br>bool *istFertig(*void*)*<br>Ware *aktuellesElement(*void*)* |
| | **Verkäufer** | *Gegenstand:* Ware<br>*Mindestpreis:* Geld<br>... | *(*Ware, Geld*) anbieten (*void*)*<br>void *abschließen (*Geld, Agent*)*<br>... |
| | **Käufer** | *Gegenstand:* Ware<br>*Höchstpreis:* Geld<br>... | Geld *bewerten (*Ware*)*<br>bool *bieten (*CountDown, Geld*)*<br>void *scheitern (*Geld*)*<br>void *abschließen (*Geld, Agent*)* |
| **Timer** | **Timer** | *Time-out:* Zeit | void *setzen (*Zeit*)*<br>void *löschen (*void*)* |

Tabelle 3.11: Dienststruktur der um Objekte erweiterten Auktion

Tabelle 3.11 erweitert die Dienststruktur gegenüber Tabelle 3.4 um Ware und Behälter für Waren (Aggregat, Iterator). Für dieses Beispiel ist es nötig die Struktur der Dienstdefinition zu detaillieren: Ein Dienst kann selbst wieder hierarisch aus Moduln und Schnittstellen aufgebaut sein, welche dann durch Dienstkontext und Dienstprimitive definiert sind (vergleiche in diesem Zusammenhang Bemerkung 3.1):

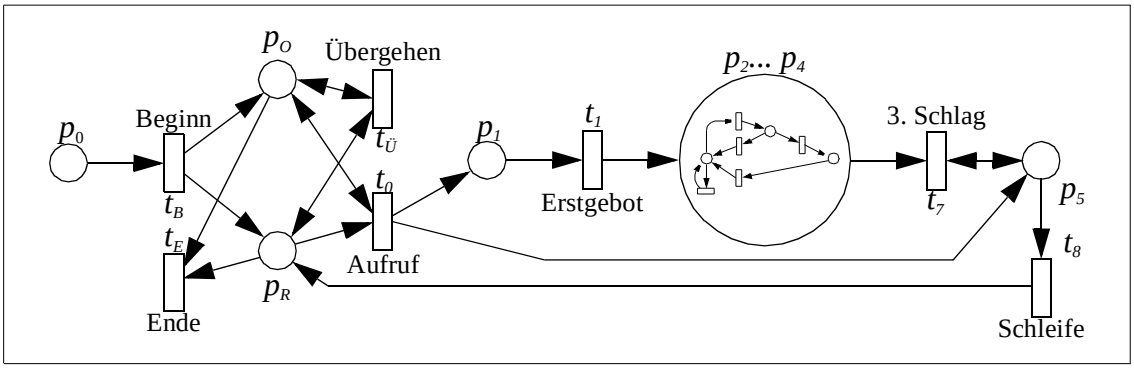


Abbildung 3.9: Netzstruktur des Interaktionsverfahren Auktion

Die Netzstruktur zwischen $t_1$ und $t_7$ entspricht der aus Abbildung 3.7.

Die Auktion sei durch ein Interaktionsnetz $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ beschrieben, wobei die Netzstruktur, wie in Abbildung 3.9 dargestellt, gegeben ist (der Netzausschnitt zwischen der Transition $t_1$ und Transition $t_7$ entspricht dem aus Abschnitt 3.3.5). Die

Beschriftung der Phasen sei wie in Tabelle 3.12 angegeben und die der Kanten und Transitionen wie in Tabelle 3.13. Die Angabe der Signatur entfällt aus Gründen der Übersichtlichkeit. Die $\Sigma$-Algebra sei gegeben durch die übliche Interpretation der Sorten und Funktionssymbole. Die Anfangsmarkierung sei wie folgt

$$M_0 = \begin{cases} \left\{ (I_0(60, \text{false}), ag_{Auktionator}, \nu_{ag}^{I_0}) \right\} \cup \\ \left\{ (A_0(\text{Liste}()), ag_j, \nu_{ag_j}^{A_0}) \;\middle|\; 1 \le j \le N \right\} \cup \\ \left\{ (O_0, rs_{Katalog}, \nu_{rs}^{O_0}) \right\} \cup \\ \{\nu(\text{Handel.Aggregat}(A_0)) := \text{Handel.Aggregat}(O_0)\} & \text{für } p_0, \\ \varnothing & \text{sonst.} \end{cases}$$

Damit ist das Interaktionsverfahren vollständig spezifiziert. Der Ablauf des Verfahrens erfolgt entsprechend der Semantik des Schaltvorgangs hinsichtlich der Rollen- und Objektmarken und der Schaltregel für Pr/T-Netze.

Der eilige Leser sei auf die Organisation der Menge der zur Versteigerung anstehenden Gegenstände in Form eines Katalogs hingewiesen:

1. In dem vorliegenden Entwurf wurde der Katalog als Behälter für die Versteigerungsgegenstände eingeführt. Denkbar wäre auch jeden Gegenstand als eine einzelne Objektmarke zu modellieren, welche in entsprechenden Phasen auf ihre Versteigerung warten würden, wobei deren Anzahl von vornherein nicht feststünde.

   Ausgehend von dem Interaktionsmechanismus Auktion kann man feststellen, daß jeder Auktion eine Phase vorangeht, in der die Auktionsteilnehmer sich eingehend über die Versteigerungsgegenstände informieren können. Diese Phase findet hinsichtlich der Teilnehmer asynchron und zueinander nebenläufig statt.

   Beschrieben durch das Interaktionsmodell, kann eine Rolleninstanz auf eine Objektinstanz zugreifen, wenn diese in ihrem Namenskontext enthalten ist. Dabei wird der Zugriff auf einen Dienstnamen aufgelöst zu einem Dienstaufruf der Objektinstanz. Ein Namenskontext unterstützt jedoch keine Behältnisfunktionalität, so daß man einem Dienstnamen nicht eine Liste von Objektinstanzen zuordnen kann. Folglich kann eine Rolleninstanz gleichzeitig nur auf eine zum Entwurfszeitpunkt feststehende Anzahl von Objektinstanzen zugreifen. Ein Wechsel des Namenskontextes erfolgt immer durch Schalten einer Transition.

| **Phasen** | **Rollen und Objekte mit denen eine Phase $p_i$ beschriftet ist** |
|---|---|
| $p_0$, $\gamma\|_R(p_0) = \{I_0, A_0\}$ $\gamma\|_O(p_0) = \{O_0\}$ | $O_0 = (D_{Def} = \{\text{Handel.Aggregat}\}, D_{Ref} = \{\text{Handel.Aggregat}\}, \vec{z} = (\emptyset), D_{Impl} = \emptyset)$<br>$I_0 = (D^A_{Ref} = \{\text{Timer}\}, D^O_{Ref} = \emptyset, \vec{z} = (\mathit{TimeOut}, \mathit{Start}),$ $ak = <\mathit{Start} = \text{Timer.setzten}(\mathit{TimeOut}) >)$<br>$A_0 = (D^A_{Ref} = \{\text{Handel.Käufer}\}, D^O_{Ref} = \{\text{Handel.Aggregat}\}, \vec{z} = (\mathit{Liste}),$ $ak = <\mathit{cursor} = \text{Handel.Aggregat}.\mathit{erzeugeIterator}()$ $\mathit{cursor.start}()$ **while** $(\neg\ \mathit{cursor.istFertig}())$ **do** **if** $(\text{Handel.Käufer}.\mathit{bewerten}(\ \mathit{cursor.aktuellesElement}\ ()) > 0)$ **then** $\mathit{Liste.add}(\ \mathit{cursor.aktuellesElement}\ ())$ **fi** $\mathit{cursor.next}\ ()$ **done** >) |
| $p_O$, $\gamma\|_R(p_O) = \emptyset$ $\gamma\|_O(p_O) = \{O_O\}$ | $O_O = (D_{Def} = \emptyset, D_{Ref} = \{\text{Handel.Iterator}\}, \vec{z} = (\mathit{Cursor}), D_{Impl} = \emptyset)$ |
| $p_R$, $\gamma\|_R(p_R) = \{I_R, A_R\}$ $\gamma\|_O(p_R) = \emptyset$ | $I_R = (D^A_{Ref} = \emptyset, D^O_{Ref} = \emptyset, \vec{z} = (\emptyset), ak = \emptyset)$<br>$A_R = (D^A_{Ref} = \emptyset, D^O_{Ref} = \emptyset, \vec{z} = (\emptyset), ak = \emptyset)$ |
| $p_1$, $\gamma\|_R(p_1) = \{I_1, A_1\}$ $\gamma\|_O(p_1) = \emptyset$ | $I_1 = (D^A_{Ref} = \emptyset, D^O_{Ref} = \emptyset, \vec{z} = (\mathit{Mindestgebot}), ak = \emptyset)$<br>$A_1 = (D_{Ref} = \{\text{Handel.Käufer}\}, \vec{z} = (\mathit{Liste}, \mathit{Gegenstand}, \mathit{Mindestgebot}, \mathit{Gebot}),$ $ak = <$**if** $(\text{Handel.Käufer}.\mathit{bewerten}(\mathit{Gegenstand}) \geq \mathit{Mindestgebot})$ **then** $\mathit{Gebot} = \text{true}$ **fi** >) |
| $p_{2-4}$ | siehe Tabelle 3.5 (analog zu $p_1$ wird der Zustandvektor von $A_{2-4}$ um *Liste* erweitert) |
| $p_5$, $\gamma\|_R(p_5) = \{I_5, B_5, A_5\}$ $\gamma\|_O(p_5) = \{O_5\}$ | $O_5 = (D_{Def} = \{\text{Handel.Ware}\}, D_{Ref} = \{\text{Handel.Ware}\}, \vec{z} = (\emptyset), D_{Impl} = \emptyset)$<br>$I_5 = (D^A_{Ref} = \{\text{Handel.Verkäufer}\}, D^O_{Ref} = \{\text{Handel.Ware}\}, \vec{z} = (\mathit{Preis}, \mathit{Käufer}),$ $ak = <\text{Handel.Verkäufer.abschließen}(\mathit{Preis}, \mathit{Käufer});$ $\text{Handel.Ware}.\mathit{übereignen}(\mathit{Preis}, \mathit{Käufer}) >)$<br>$B_5 = (D^A_{Ref} = \{\text{Handel.Käufer}\}, D^O_{Ref} = \emptyset, \vec{z} = (\mathit{Liste}, \mathit{Preis}, \mathit{Verkäufer}),$ $ak = <\text{Handel.Käufer.abschließen}(\mathit{Preis}, \mathit{Verkäufer}) >)$<br>$A_5 = (D^A_{Ref} = \{\text{Handel.Käufer}\}, D^O_{Ref} = \emptyset, \vec{z} = (\mathit{Liste}, \mathit{Preis}),$ $ak = <\text{Handel.Käufer.scheitern}(\mathit{Preis}) >)$ |

Tabelle 3.12: Rollendefinitionen für das Interaktionsverfahren Auktion

| T | **Beschriftungen der In-Kanten, des Wächters und der Out-Kanten einer Transition $t_i$; Kanten sind jeweils mit Mengen von Termen beschriftet** |
|---|---|
| $t_B$ | $\sigma(p_0, t_B) = \{o\colon O_0, i\colon I_0, \forall a_j\colon A_0\}$<br>$\sigma(t_B) = i.\,start \wedge \text{Deaktiviert}(a_j)$<br>$\sigma(t_B, p_O) = \{o \leftarrow O_O(\text{Handel.Aggregat.}erzeugeIterator().start())\}$<br>$\sigma(t_B, p_R) = \{i \leftarrow I_R(\varnothing), a_j \leftarrow A_R(a_j.Liste)\}$ |
| $t_{Ü}$ | $\sigma(p_O, t_{Ü}) = \{o\colon O_0\}$<br>$\sigma(p_R, t_{Ü}) = \{i\colon I_R, \forall a_j\colon A_R\}$<br>$\sigma(t_{Ü}) = \neg\, o.cursor.istFertig() \wedge a_j.Liste.enthält(o.cursor.aktuellesElement()) \wedge \text{Card}(\forall a_j) = 0$<br>$\sigma(t_{Ü}, p_5) = \{o \leftarrow O_5(o.cursor.aktuellesElement())\}$<br>$\sigma(t_{Ü}, p_O) = \{o \leftarrow O_O(o.cursor.next())\}$<br>$\sigma(t_{Ü}, p_R) = \{i \leftarrow I_R(), \forall a_j \leftarrow A_R(a_j.Liste)\}$ |
| $t_0$ | $\sigma(p_O, t_0) = \{o\colon O_0\}$<br>$\sigma(p_R, t_0) = \{i\colon I_R, \forall a_j\colon A_R\}$<br>$\sigma(t_0) = \neg\, o.cursor.istFertig() \wedge a_j.Liste.enthält(o.cursor.aktuellesElement()) \wedge \text{Card}(\forall a_j) > 0$<br>$\sigma(t_0, p_5) = \{o \leftarrow O_5(o.cursor.aktuellesElement())\}$<br>$\sigma(t_0, p_O) = \{o \leftarrow O_O(o.cursor.next())\}$<br>$\sigma(t_0, p_1) = \{i \leftarrow I_1(o.cursor.aktuellesElement().Mindestpreis),$<br>$a_j \leftarrow A_1(a_j.Liste,$<br>$o.cursor.aktuellesElement().Gegenstand,$<br>$o.cursor.aktuellesElement().Mindestpreis,$<br>$\text{false})\}$ |
| $t_{1-7}$ | siehe Tabelle 3.5 (analog zu $\sigma(t_0, p_1)$ wird das Element *Liste* des Zustandvektors von $A_{1-4}$ unverändert weitergereicht) |
| $t_7$ | $\sigma(p_4, t_7) = \{i\colon I_4, b\colon B_4, \forall a_j\colon A_4\}$<br>$\sigma(p_5, t_7) = \{o\colon O_5\}$<br>$\sigma(t_7) = i.Hammer$<br>$\sigma(t_7, p_5) = \{i \leftarrow I_5(i.letztesGebot, \text{ag}(b)), \nu(i.\text{Handel-Ware}) := o.\text{Handel-Ware},$<br>$b \leftarrow B_5(b.Liste, i.letztesGebot, \text{ag}(i)),$<br>$a_j \leftarrow A_5(a_j.Liste, i.letztesGebot)\}$ |
| $t_8$ | $\sigma(p_5, t_8) = \{i\colon I_5, b\colon B_5, \forall a_j\colon A_5, o\colon O_5\}$<br>$\sigma(t_8) = \text{Deaktiviert}(i) \wedge \text{Deaktiviert}(b) \wedge \text{Deaktiviert}(\forall a_j)$<br>$\sigma(t_8, p_R) = \{i \leftarrow I_R(), b \leftarrow A_R(b.Liste), a_j \leftarrow A_R(a_j.Liste)\}$ |
| $t_E$ | $\sigma(p_O, t_E) = \{o\colon O_0\}$<br>$\sigma(p_R, t_E) = \{i\colon I_R, \forall a_j\colon A_R\}$<br>$\sigma(t_E) = o.cursor.istFertig()$ |

Tabelle 3.13: Beschriftung $\sigma$ für das Interaktionsverfahren Auktion

Im vorliegenden Entwurf blättert ein Auktionsteilnehmer in der Rolle $A_0$ durch den Katalog. Die in seiner Aktion modellierte Schleife könnte auch durch das Schalten einer Transition realisiert werden, wobei die Rolle dann wissen müßte, über welche Objektinstanzen sie sich bereits informiert hat. Die Transition wäre dann in der Lage an sie nur unbekannte Objektinstanzen zu binden. Sobald keine solche Objektinstanz mehr vorhanden wäre, würde die Rolleninstanz die Phase verlassen.

Diese Alternative ist gut geeignet, solange zwischen den Objektinstanzen keine Ordnung existiert. Andernfalls ist die Nutzung komplexer Datenstukturen, wie in diesem Fall die eines Aggregats, verständlicher und effizienter.

2. Der Katalog wird also in der Phase $p_0$ von den Auktionsteilnehmern studiert, und sobald alle Rolleninstanzen den Katalog durchgearbeitet haben, wird die Auktion durch Schalten der Transition “Beginn” eröffnet. Dabei wird in der Phase $p_O$ der Katalog als Objekt gelagert und die Rollen in die Phase $p_R$ geschaltet.

   Nun werden in einer Schleife solange die Gegenstände versteigert, bis der Katalog abgearbeitet ist. Während der Informationsphase $p_0$ haben sich die Auktionsteilnehmer die Versteigerungsgegenstände gemerkt, für die der ihnen zugeordnete Agent sich interessierte und bereit war den Mindestpreis zu bieten. Falls ein Versteigerungsgegenstand kein Intersse gefunden hat, wird er im Katalog durch Schalten der Transition $t_{\ddot{U}}$ übergangen.

Der Schleifenkörper entspricht weitgehend dem Entwurf der Auktion aus Abschnitt 3.3.5, welcher ausschließlich Rollen verwendete. Der Versteigerungsgegenstand wird nach seinem Aufruf in die Phase $p_5$ geschaltet, wo er auf die Eigentumsüberschreibung wartet.

### 3.4.6 Diskussion

Durch die Einführung der Objektabstraktion in der zweiten Iterationsstufe des Interaktionsmodells ist es nun möglich passive Komponenten eines Multiagentensystems in die Beschreibungen von Interaktionsmechanismen mit aufzunehmen. Somit stehen dem Entwickler neben den Mechanismen der Verhandlungsführung, wie er zwischen Rollen und Agenten üblich ist, auch die Mechanismen der Client/Server-Interaktion wie sie in nichtautonomen, verteilten Anwendungen üblich ist als integrales, durchgängiges Modellierungskonzept zur Verfügung.

1. Die in einer Anwendung auftretenden, passiven Anwendungsobjekte werden durch sogenannte Ressourcen modelliert. Sie sind z. B. in Datenbanken persistent gespeichert und werden durch dieses in Interaktionsverfahren eingebracht. Das Datenbanksystem bildet die auf den Anwendungsobjekten definierten Dienste ab.
2. Die in einer Interaktion verwendeten Ressourcen werden durch Objekte modelliert, welche eine höhere Abstraktion innerhalb der speziellen Interaktion bieten. Objekte können eigene Dienste definieren und implementieren, welche sich schließlich auf die von den Ressourcen bereitgestellten Dienste abstützen.
3. Rolleninstanzen werden durch den Schaltvorgang einer Transition an Objektinstanzen gebunden, so daß sie deren Dienste entsprechend einem Client/Server-Mechanismus nutzen können. Somit entspricht eine Phase für jede Rolleninstanz einem Namenskontext, in dem Dienstreferenzen auf dem der Rolleninstanz fest zugeordneten Agenten und Dienstreferenzen auf der Rolleninstanz temporär zugeordneten Objektinstanzen durch entsprechende Namensbindungen auf Dienstdefinitionen abgebildet werden.
4. Objekte können dazu verwendet werden, spezielle Kommunikationsmechanismen zu kapseln. Somit können proprietäre Kommunikationskanäle direkt genutzt werden. Dies stellt eine entscheidende Verbesserung gegenüber dem Interaktionsmodell aus der ersten Iterationsstufe dar. Soll zwischen zwei Agenten, deren Rollen sich in der selbe Phase befinden ein expliziter Kommunikationskanal geöffnet werden, so findet der Steuerungsteil des Protokolls über ein entsprechendes Objekt zwischen den Rollen statt. Gewähren die Rollen den Agenten nach einem Verbindungsaufbau Zugriff auf die Kommunikationsressource, so kann der Datenfluß dann direkt zwischen den Agenten abgewickelt werden.
5. Objektinstanzen können von einer Belegung nur geschaltet werden, wenn sie nicht $\omega$-blockiert ist, wenn also keine Rolle mehr auf sie zugreift. Somit ist es unmöglich, eine Rolle nachträglich an ein Objekt zu binden. In der nächsten Iterationsstufe werden wir eine Erweiterung einführen, welche das nachträgliche Binden einer Rolle an ein Objekt ermöglicht. Diese Erweiterung könnte bereits in dieser Iterationsstufe erfolgen. Sie wird jedoch auf die nächste Iterationsstufe verschoben, da sie dort unerläßlich ist. Ohne diese Erweiterung wäre eine nachträgliche Teilnahme einer Rolle an einem laufenden Interaktionsverfahren unmöglich.
6. Wie aus dem Beispiel und insbesondere aus der Abbildung 3.9 hervorgeht, wachsen die Petrinetze unter Einbeziehung von Ausnahmebehandlungen schnell an. Es ist also ein Modularisierungskonzept nötig, welches eine Abstraktion auf Interaktionsverfahren bietet.

# 3.5 Interaktionsmodell mit Subinteraktionsverfahren

In Abschnitt 3.3 wurde das grundlegende Interaktionsmodell mit Rollen eingeführt, welches in Abschnitt 3.4 um das Objektkonzept erweitert wurde. In diesem Abschnitt soll nun das Modulkonzept eingeführt werden, auf dem aufbauend Subinteraktionsverfahren beschrieben werden können.

Mit dem Objektkonzept aus Abschnitt 3.4.2 können Datenobjekte spezifiziert werden, auf denen Rollen interagieren. Datenobjekte können eigene gekapselte Interaktionsmechanismen besitzen. Für komplexe Interaktionsmechanismen benötigt man eine geeignete Abstraktion für deren Modellierung und eine geeignete Unterstützung zur Laufzeit.

Durch das Modulkonzept kann die Schnittstelle zu einem Interaktionsverfahren in Form eines Objektes spezifiziert werden und in andere Interaktionsverfahren eingebunden werden. Der Interaktionsmechanismus ist dann im Körper des Objekts gekapselt (vgl. Abbildung 3.10).

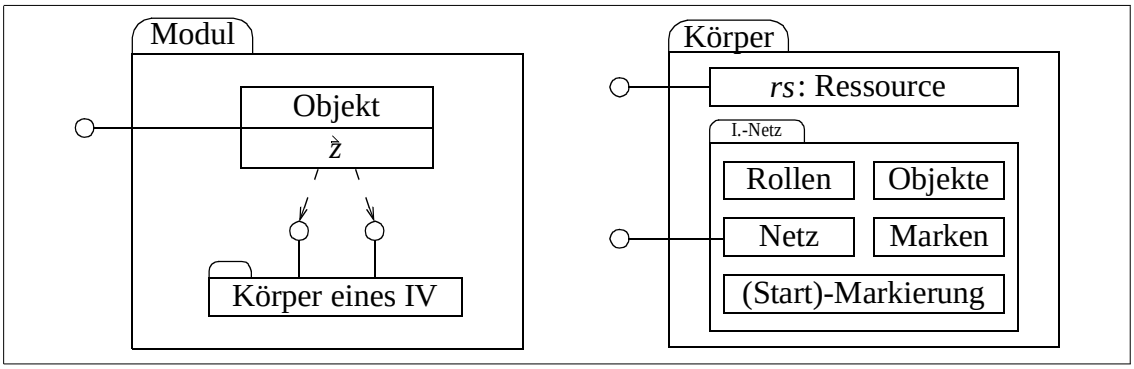


Abbildung 3.10: Struktur eines Moduls und seines Körpers

Im Folgenden werden wir zuerst Anforderungen an das Interaktionsmodell aufstellen, welche die Wiederverwendbarkeit an Interaktionsverfahren stellt. Wir werden darstellen, wie diese Anforderungen durch Nutzung von bestimmten Mustern innerhalb der Petrinetzbeschreibung erfüllt werden können.

Wie in Abschnitt 3.4.3 definiert, besteht ein Interaktionsverfahren aus einer Menge von Rollen, einer Menge von Objekten, einer Pertinetzstruktur welche über den Rollen und den Objekten definiert ist und aus einer Anfangsmarkierung.

Die Anfangsmarkierung hat in Petrinetzen im wesentlichen zwei Aufgaben: Einerseits verteilt sie “Kontrollmarken” im Netz, mit denen einzelne Stellen so initialisiert werden, daß sie zusammen mit den Transitionen im Nachbereich ein definiertes Schaltverhalten des Netzes garantieren. Andererseits definiert die Anfangsmarkierung die Startsituation, von der ausgehend der Petrinetzprozeß entsteht.

Für die Wiederverwendbarkeit eines Interaktionsverfahrens ist es notwendig, die konkrete Anfangsmarkierung zur Initialisierungszeit aus einer *Menge von zulässigen Anfangsmarkierungen* wählen zu können, für welche das Interaktionsverfahren ein korrektes Verhalten zeigt. Beispielsweise kann die Anzahl der Teilnehmer, die an einer Interaktion beteiligt sind in bestimmten Grenzen variieren. Nach der Wahl der Anfangsmarkierung, soll das Petrinetz unter dieser Markierung starten.

Charakteristisch für Agentensysteme als offene Systeme ist, daß während der Laufzeit neue Agenten in das System eintreten, bzw. das System wieder verlassen können. Demnach müssen auch die Teilsysteme und Module diese Offenheit unterstützen. Beim Entwurf von Interaktionsnetzen ist also ein geschlossenes System der Spezialfall und Netze in denen weitere Agenten *asynchron*, während der Laufzeit des Prozesses ein- und wieder austreten können der Standardfall.

### 3.5.1 Muster für Ein- und Ausgänge

Während eines Petrinetzprozesses können Marken in den Prozeß eintreten, wenn in der Netzstruktur z. B. Transitionen existieren, welche keinen Vorbereich haben (oder allgemein, wenn es Transitionen gibt, die mehr Marken erzeugen als sie konsumieren). Analog können Marken aus einem Prozeß z. B. durch Transitionen ausscheiden, welche keinen Nachbereich haben (oder allgemein, wenn es Transitionen gibt, welche mehr Marken konsumieren als erzeugen).

Der Eintritt einer Marke muß mit dem laufenden Prozeß synchronisiert werden. Dies kann auf zwei Arten geschehen: Entweder erzeugt ein Petrinetzprozeß sich die Marke selbst, oder er überprüft an bestimmten Eintrittspunkten, ob Marken in den laufenden Prozeß mit aufgenommen werden wollen. Letzteres bedeutet, daß eine Marke nach ihrem Eintritt in einen Prozeß ersteinmal in einer Wartephase verweilt, bis der Prozeß bereit ist sie aufzunehmen.

Abbildung 3.11 stellt das Muster eines Netzausschnittes für einen asynchronen Eingang[1] dar. Über die Transition “Eingang” können jederzeit Marken das Netz betreten. Abhängig von dem Zustand der Objektmarke in der Wächterstelle kann die Marke unterschiedlich behandelt werden:

- Die Marke verläßt das Netz im nächsten Schritt durch den zugehörigen Ausgang.
- Die Marke verweilt in der Wartephase bis der Wächter entweder das Tor oder den Ausgang freigibt.
- Die Marke betritt das Interaktionsverfahren im nächsten Schritt.

1. Aus technischen Gründen kann ein Eingang nur genau einen Markentyp erzeugen, d. h. der Eingangsstelle ist genau eine Rolle oder ein Objekt zugeordnet (ihre Kapazität ist unbeschränkt).

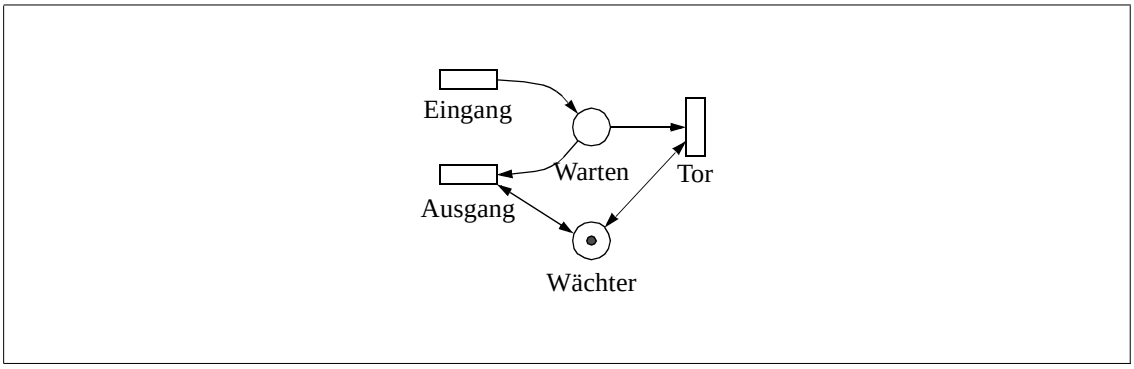


Abbildung 3.11: Ein Muster für einen bewachten, asynchronen Eingang

Die Wächterfunktionalität ist dabei aufgeteilt in drei Bereiche:

- Die Objektmarke definiert die Zustände des Wächters (einlassen, warten lassen oder abweisen). Sie kann entweder passiv sein, also Zustandsübergänge nur aufgrund äußerer Ereignisse vornehmen oder sie hat einen inneren Wecker, der einen Zustandsübergang auslöst.
- Die Tor-Transition löst einen Zustandübergang in der Objektmarke aus, sobald sie eine Marke eingelassen hat.
- Die Ausgang-Transition löst einen Zustandübergang in der Objektmarke aus, sobald sie eine Marke entfernt hat.

Zu dem genannten Muster gibt es zwei Erweiterungen: Häufig ist es notwendig, den Wächter vom Petrinetzprozeß aus beeinflussen zu können, ihn also sperren oder öffnen zu können. Hierfür kann eine Transition ergänzt werden, welche die Wächtermarke konfiguriert. Zweitens kann die Durchlässigkeit des Wächters eine Funktion über den Zustand der Marken in der Wartephase sein, also nur bestimmte Marken durchlassen.

Das Muster für einen asynchronen Eingang wird für Rollenmarken verwendet, welche aus eigenem Antrieb in ein laufendes Interaktionsverfahren eintreten. Gleiches gilt für Objektmarken, welche aufgrund des Schaltens des aufrufenden Interaktionsverfahrens in das laufende eingehängt werden. Beide Markenarten geben bei dem Eintritt in ein Subinteraktionsverfahren die Identität des an sie gebundenen Agenten bzw. der an sie gebundenen Ressource an die Marke weiter, welche im Subinteraktionsverfahren erzeugt wird.

Eingangs- und Ausgangstransitionen bilden die Schnittstelle des Modulkörpers (vgl. Abbildung 3.11). Jede *Eingangstransition* wird auf genau eine *Dienstdefinition* in der Schnittstellenspezifikation abgebildet, welches durch ein Objekt beschrieben wird.

*Eingangstransitionen* sind Transitionen ohne Vorbereich; ihr Nachbereich besteht aus genau einer Phase; die Beschriftung der ausgehenden Kante besteht aus genau einem Subsitutionsterm. Über Eingangstransitionen werden neue Marken im Petrinetz erzeugt. Dabei wird einer Rollenmarke ein Agent oder einer Objektmarke eine Ressource zugeordnet. Diese sind durch den Aufrufer des Dienstes bestimmt, welcher der Eingangstransition zugeordnet ist. In der von der Transition ausgehenden Kante wird der Marke

erstmals eine Rolle oder ein Objekt zugeordnet. (Technisch gesehen definieren Eingangstransitionen eine spezielle Variable "*newToken*", an welche beim Schaltvorgang die neu erzeugte Marke gebunden ist. In dem Substitutionsterm der ausgehenden Kante wird diese Variable verwendet um der Marke eine neue Rolle oder ein neues Objekt zuzuordnen. Dabei können Parameter, die beim Aufruf des Dienstes übergeben wurden, für die Initialisierung des Zustandes verwendet werden.)

Im Gegensatz zum Aufrufparameter kann der Rückgabeparameter des Dienstes nicht direkt aus der konsumierten Marke einer *Ausgangstransition* abgeleitet werden. Eine *Ausgangstransition* ist eine Transition ohne Nachbereich. Eine Marke kann das Petrinetz durch eine Vielzahl von Ausgangstransition verlassen, welche Senken des gerichteten Erreichbarkeitsgraphen sind (vgl. Abschnitt 3.5.5). Die Abbildung der von einer Ausgangstransition konsumierten Marke auf die Rückgabeparameter[1] wird durch die Implementierung der Dienstdefinition geleistet.

Die *Menge der zulässigen Anfangsmarkierungen* kann nun durch Anwendung dieses Musters modelliert werden (vgl. Abbildung 3.12): Jede zulässige Anfangsmarkierung $M_0^i,\ 1 \le i \le n$ wird als eine Tor-Transition modelliert, welche die Markierung beim Schalten erzeugt. Alle Tor-Transitionen sind von einem zentralen Wächter gesteuert, welcher den Beginn des Interaktionsverfahrens verzögert, bis die Bewerbungsfrist für die Interaktion abgelaufen ist. Anschließend schaltet nicht-deterministisch eine der Tor-Transition und erzeugt somit die zugeordnete Anfangsmarkierung. Alle nicht eingelassenen Marken werden durch die Ausgänge wieder aus dem Netz entfernt.

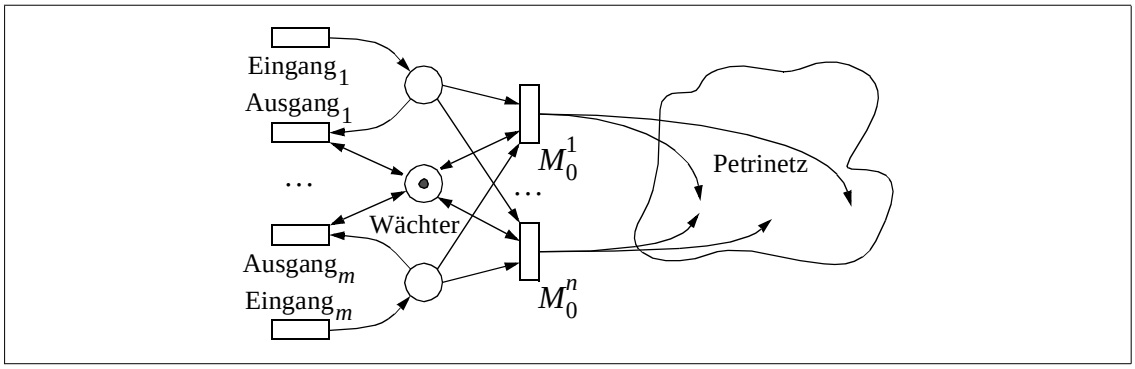


Abbildung 3.12: Ein Muster für die Modellierung einer Menge von zulässigen Anfangsmarkierungen $\{M_0^1, \ldots, M_0^n\}$

Dieses Muster läßt sich zu einer prioritätsgesteuerten Auswahl der Anfangsmarkierung erweitern. Hierfür modelliert der Wächter einen Prioritätszähler. Zu jeder zulässigen Anfangsmarkierung gibt es die Tor-Transition und eine komplementäre Transition, welche den Prioritätszähler des Wächters weiterschaltet, falls die Tor-Transition nach Ablaufen der Bewerbungsfrist nicht aktiviert ist.

1. Beachte die unterschiedliche Behandlung von aufrufenden Rollen und Objekten hinsichtlich der Möglichkeit von Rückgabeparametern (vergleiche Bemerkung 3.15).

Damit ist die Erweiterung der Anfangsmarkierung zu einer Menge zulässiger Anfangsmarkierungen definiert. Neben der Spezifikation einer Startsituation hat die Anfangsmarkierung in Petrinetzen noch die Aufgabe einzelne Stellen mit Kontrollmarken (z.B. Wächter-Marken) zu initialisieren. Diese Initialisierung, welche für alle zulässigen Anfangsmarkierungen gleich ist, wird weiterhin durch die Anfangsmarkierung des Interaktionsnetzes spezifiziert.

### 3.5.2 Zustand eines Interaktionsverfahrens

Ein Interaktionsverfahren ist das Modell der durch ihn beschriebenen Interaktion. Der ablauforientierte Teil des Modells wird durch das Interaktionsnetz beschrieben. Das Modell des Interaktionsziels und des erreichten Interaktionsergebnisses wird durch Objekte beschrieben, welche an eine dem Interaktionsverfahren zugeordneten Ressource gebunden sind. Die Markierung des Interaktionsnetzes und die in der Ressource gespeicherte Information definiert zu einem Zeitpunkt den Zustand des Interaktionsverfahrens (vgl. Abbildung 3.10).

Auf den durch die Ressource modellierten Ausschnitt des Zustandes der Interaktion, kann einerseits von dem die Schnittstellenspezifikation des Moduls definierenden Objekt aus zugegriffen werden. Dieser Mechanismus wird für folgende Aspekte benötigt:

- Konfiguration eines Moduls zum Zeitpunkt seiner Instantiierung (z. B. kann somit die Bewerbungsfrist für die Interaktion von außen vorgegeben werden.)
- Speicherung und Rückgabe des Interaktionsergebnisses an ein aufrufendes Interaktionsverfahren.
- Nach Definition 3.10 terminiert ein Interaktionsverfahren dann, wenn sich keine Rollenmarken mehr im Interaktionsnetz befinden. Eine einmal beendete Instanz eines Interaktionsverfahrens kann also nicht durch einen nachträglichen Eintritt einer Rollenmarke wieder aktiviert werden. Der Zustand der Instanz (laufend oder beendet) wird durch ein Element des Zustandsvektors des Objekts modelliert.

  Wird versucht eine Eingangstransition zu schalten, nachdem die dem Objekt zugeordnete Instanz des Interaktionsnetzes beendet ist, so stößt dies eine entsprechende Ausnahmebehandlung an.

Andererseits kann auf die Ressource von innerhalb des Interaktionsverfahren aus zugegriffen werden. Objektmarken, welche “Kontrollmarken” wie beispielsweise die bereits eingeführten Wächtermarken modellieren, ist die Ressource des Interaktionsverfahrens zugeordnet.

### 3.5.3 Interaktionsverfahren als Objekt (Modulkonzept)

Der Begriff eines Interaktionsobjektes ist die Abstraktion von einem Interaktionsmechanismus, an welchem Rollen und Objekte der selben Phase teilnehmen können. Interaktionsobjekte modellieren die Modulschnittstelle zu einem Interaktionsverfahren. Sie werden von Objekten und Objektmarken modelliert, bieten jedoch gegenüber diesen die Möglichkeit, daß sich außer Rollen auch andere Objektmarken an sie binden können.

Wir erweitern zuerst die Objekte um Dienstnamen, welche auf Dienste von Interaktionsobjekte zugreifen können, sowie einen Mechanismus, der diese Bindung eingeht. Anschließend erweitern wir die Rollen analog und definieren dann das Interaktionsnetz.

**Definition 3.27** *Objekt*

Ein Objekt wird durch das Tupel $o = (D_{Def}, D^{RS}_{Ref}, D^{I}_{Ref}, D_{Impl}, \vec{z})$ beschrieben. Dabei ist $D_{Def} \subseteq D$ die Menge der von dem Objekt *bereitgestellten Dienste*, $D^{RS}_{Ref} \subseteq D$ die Menge der von dem Objekt *referenzierten Ressourcendienste*, $D^{I}_{Ref} \subseteq D$ die Menge der von dem Objekt *referenzierten Dienste eines Interaktionsobjektes*, $D_{Impl} \subseteq D$ die *Implementierung* der Menge der bereitgestellten Dienste und $\vec{z}$ ein Vektor von *beobachteten Zuständen*. Wir bezeichnen mit $\mathrm{D}_{\mathrm{Def}}(o)$ die Menge der bereitgestellten Dienste, mit $\mathrm{D}^{RS}_{\mathrm{Ref}}(o)$ die Menge der referenzierten Ressourcendienste, mit $\mathrm{D}^{I}_{\mathrm{Ref}}(o)$ die Menge der referenzierten Dienste eines Interaktionsobjektes, mit $\mathrm{D}_{\mathrm{Impl}}(o)$ die Menge der Implementierungen und mit $\mathrm{z}(o)$ den Vektor $\vec{z}$ der durch das Objekts $o$ beobachteten Zustände. Mit $O$ bezeichnen wir die Menge aller Objekte.

**Definition 3.28** *Interaktionsobjekt*

Ein Interaktionsobjekt ist ein Objekt mit $D^{I}_{Ref} = \emptyset$. Mit $D_I$ bezeichnen wir die Menge der Interaktionsobjekte.

### 3.5.4 Interaktionsnetz mit Rollen-, Objekt- und Modulkonzept

Das Interaktionsmodell der dritten Iterationsstufe ist gegenüber dem der zweiten Iterationsstufe um die Objektklasse der Interaktionsobjekte erweitert. In der zweiten Iteration wurde die Spezifikationssprache um *Kontextterme* erweitert, welche beschreiben, wie während einem Schaltvorgang die Bindung einer Rollen- an eine Objektinstanz erfolgen soll. Der Aufruf eines Objektdienstes erfolgt nach der Aktivierung der Rolle gesteuert durch ihre Aktion. Mit der Einführung von Interaktionsobjekten in der dritten Iterationsstufe, wird die Spezifikationssprache um *Aufrufterme* erweitert:

Rolleninstanzen werden wie bereits aus der zweiten Iterationsstufe bekannt, durch *Kontextterme* an Interaktionsobjektinstanzen gebunden. Objektinstanzen konnten bisher nicht an andere Objektinstanzen gebunden werden. Dies ist in dieser Allgemeinheit auch weiterhin nicht sinnvoll, jedoch ist die Bindung eines (Nicht-Interaktions-) Objekts an ein Interaktionsobjekt erforderlich um auch Ressourcen geeignet in ein Subinteraktionsverfahren einbringen zu können. In Abwandlung der Kontextterme welche ausschließlich Rollenmarken vorbehalten bleiben, werden *Aufrufterme* eingeführt, durch welche Objektmarken beim Schalten einer Transition an Interaktionsobjektmarken gebunden werden.

**Definition 3.29** *Rolle*

Eine Rolle *r* wird durch das Tripel $r = (D^A_{Ref}, D^O_{Ref}, D^I_{Ref}, \vec{z}, ak)$ beschrieben. Dabei ist $D^A_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Agentendienste,* $D^O_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Objektdienste,* $D^I_{Ref} \subseteq D$ die Menge der von der Rolle *referenzierten Dienste eines Interaktionsobjektes*, $\vec{z}$ ein Vektor von *Zuständen* und *ak* eine *Aktion.* Die Aktion *ak* besteht aus Dienstprimitiven aus $D^A_{Ref} \cup D^O_{Ref} \cup D^I_{Ref}$ und Operationen auf Zustände (Komponenten) aus $\vec{z}$. Wir bezeichnen mit $\mathrm{D}^A_{\mathrm{Ref}}(r)$, $\mathrm{D}^O_{\mathrm{Ref}}(r)$ und $\mathrm{D}^I_{\mathrm{Ref}}(r)$ die Menge der referenzierten Agenten-, Objekt- oder Interaktionsobjektdienste, mit $\mathrm{z}(r)$ den Vektor $\vec{z}$ der Zustände beziehungsweise mit $\mathrm{ak}(r)$ die Aktion der Rolle *r.* Mit $R$ bezeichnen wir die Menge aller Rollen.

**Bemerkung 3.17**

Im Gegensatz zu Objekten, können Rollen die Teilnahme an einem Subinteraktionsverfahren verzögern. Die Entscheidung hierzu wird von der Rolle und, falls sie dieses vorsieht, von dem ihr zugeordneten Agenten getroffen. Objekte haben diese Autonomie nicht.

Technisch gesehen wird eine Rolle durch Schalten einer Transition an eine Marke eines Interaktionsobjekts gebunden. Die Teilnahme erfolgt aus der Aktion der Rolle heraus durch den Aufruf eines Dienstes des Interaktionsobjektes. Die Aktion kann eine geeignete Programmlogik besitzen, welche den Aufruf nur bedingt ausführt, z. B. wenn der Agent zuvor nochmals zugestimmt hat.

Ebenso wie eine Rolle wird ein Objekt durch Schalten einer Transition an eine Marke eines Interaktionsobjekts gebunden. Ein Objekt unterscheidet sich unter anderem von einer Rolle dadurch, daß es keine ihm zugeordnete Aktion besitzt. Somit erfolgt der Aufruf eines Dienstes des Interaktionsobjektes nicht wie bei der Rolle aus der Aktion heraus, sondern durch den Term der Kante, welcher das Objekt in die entsprechende Phase schaltet. Da ein Objekt über keine eigene Autonomie verfügt, ist diese Modellierung schlüssig.

Laut Bemerkung 3.11 ist ein Objekt eine phasenspezifische Sicht auf eine Ressource. Das Objekt selbst ist zustandlos. Im Gegensatz zur Rolle, welche eigene, vom Agenten unabhängige Zustände besitzt, ist beim Aufruf eines Subinteraktionsverfahrens eine Parameterrückgabe an das "aufrufende" Objekt nicht nötig. Zustandsänderungen, welche in einem Subinteraktionsverfahren an einer Ressource vorgenommen werden, sind (entsprechend den in der Ressource verwendeten Mechanismen zur Konsistenzsicherung) direkt in anderen Interaktionsverfahren sichtbar.

Somit kann ein Objekt einen Aufruf eines Interaktionsverfahrens nicht verzögern oder gar ablehnen. Dieses ist nur einer Rolle möglich.

Wir werden später sehen, daß diese Eigenschaft in der Definition von *Aufruftermen*, welche Objekte an ein Interaktionsverfahren binden, genutzt wird.

Der Namenskontext einer Rollen- bzw. Objektmarke wird nun noch um die Bindung an ein Interaktionsobjekt erweitert:

**Definition 3.30** *Rollenmarke*

Eine Rollenmarke ist als ein Tripel $\rho = (r, ag, \nu)$ definiert, wobei $r \in R$ eine *Rolle*, $ag \in A$ einen *Agenten* und $\nu$: $\nu_A \cup \nu_O \cup \nu_I$ den *Namenskontext* definiert:

- $\nu_A$: $D^A_{Ref}(r) \rightarrow ag$ ordnet den von der Rolle referenzierten Agentendiensten die Fähigkeiten des Agenten zu,
- $\nu_O$: $D^O_{Ref}(r) \rightarrow D_{Def}(o)$ ordnet den von der Rolle referenzierten Objektdiensten die von einem Objekt *o* bereitgestellten Dienste zu und
- $\nu_I$: $D^I_{Ref}(r) \rightarrow D_{Def}(oi)$ ordnet den von der Rolle referenzierten Diensten eines Interaktionsobjekts die von einem Interaktionsobjekt *oi* bereitgestellten Dienste zu.

Wir bezeichnen mir $r(\rho)$ die Rolle der Marke.

**Definition 3.31** *Objektmarke*

Eine Objektmarke ist als ein Tripel $\omega = (o, rs, \nu)$ definiert, wobei $o \in O$ ein Objekt, $rs \in RS$ eine Ressource und $\nu$: $\nu_{RS} \cup \nu_I$ den Namenskontext definiert:

- $\nu_{RS}$: $D^{RS}_{Ref}(o) \rightarrow rs$ ordnet den von dem Objekt referenzierten Ressourcendiensten die Funktionalität der Ressource zu und
- $\nu_I$: $D^I_{Ref}(o) \rightarrow D_{Def}(oi)$ ordnet den von dem Objekt referenzierten Diensten eines Interaktionsobjekts die von einem Interaktionsobjekt *oi* bereitgestellten Dienste zu.

Wir bezeichnen mit $o(\omega)$ das Objekt der Marke.

Die Phase ist weiterhin wie in Definition 3.20 definiert.

Die Definition des Interaktionsnetzes wird um drei Aspekte erweitert:

- Bisher waren Kontextterme nur auf Rollen- und Objektmarken definiert, mit denen eine Rolle Zugriff auf entsprechende Dienste eines Objektes erhielt. Der damit verbundene Mechanismus wird nun auch für den Zugriff auf Dienste eines Subinteraktionsverfahrens genutzt. Wir erweitern die Kontextterme zu *Aufruftermen*, welche auf Objekt- und Interaktionsobjektmarken definiert sind. Unter Verwendung des bestehenden Mechanismus ist es nun möglich, eine Objektmarke (initiiert durch einen Aufrufterm an einer Kante) an eine Dienstdefinition eines Interaktionsobjektes zu übergeben und somit über eine Eingangstransition eine Marke in einem Subinteraktionsverfahren zu erzeugen.
- Es werden auf syntaktischer Ebene Referenzvariablen eingeführt, welche in Kontext- und Aufruftermen verwendet werden, welche an Kanten stehen, die in die Stelle zeigen, aus der die Referenzvariable mit einer Marke belegt wurde. Dadurch können Rollenmarken oder Objektmarken an bereits laufende Subinteraktionsverfahren gebunden werden.
- Die Interpretation der Funktionssymbole wird an die erweiterte Definition der Rollen und Objekte angepaßt.
- Die Eingangstransitionen werden so erweitert, daß für sie die Variable `newToken` deklariert ist. Diese Variable kann dann im Substitutionsterm, mit dem die ausgehende Kante beschriftet ist, verwendet werden.

Im Folgenden geben wir die Definition des Interaktionsnetzes vollständig an.

**Bemerkung 3.18**

Im folgenden bezeichnen wiederum $\hat{S}, \hat{R}, \hat{ZR}, \ldots$ die syntaktische Menge von Sorten, Rollen, Zuständen der Rollen etc. Damit unterscheiden wir die Komponenten der Syntax von außerhalb des Interaktionsnetzes definierten Begriffen.

**Definition 3.32** *Interaktionsnetz*

Ein Interaktionsnetz ist ein Pr/T-Netz, welches um das Rollen- und das Objektkonzept erweitert ist:

Kurze Erinnerung an die Definition: $P$ = Menge der Phasen, $\gamma(p) \subseteq R \cup O$, $\gamma|_R(p) \subseteq R$, $\gamma|_O(p) \subseteq O$ Rollen und Objekte der Phase, $z(r)$, $z(o)$ = Zustandsvektor, $\rho$ = Rollenmarke, $r(\rho)$ = Rolle der Marke, $\omega$ = Objektmarke, $o(\omega)$ = Objekt der Marke. Mit $|\vec{z}|$ wird die Länge eines Vektors $\vec{z}$ bezeichnet.

1. Die Erweiterung wird über die Signatur $\Sigma = (\hat{S}, \Phi)$ wie folgt definiert: Wir bisher definieren die Menge der *Sorten* $\hat{S}$ wie in Tabelle 3.14.

| Menge der *Sorten* ... | $\hat{S} = (\hat{R} \cup \hat{ZR} \cup \hat{RM} \cup 2^{\hat{RM}} \cup \hat{DN}) \cup (\hat{O} \cup \hat{ZO} \cup \hat{OM} \cup 2^{\hat{OM}} \cup \hat{DF})$ |
|---|---|
| *der Rollen* | $\hat{R} = \bigcup_{p \in P} \gamma\|_R(p)$ |
| *der Zustände der Rollen* | $\hat{ZR} = \bigcup_{\hat{r} \in \hat{R}} \{\hat{S} \mid \hat{S}$ Sorte von $z(\hat{r})_i, 1 \le i \le \|z(\hat{r})\|\}$ |
| *der Rollenmarken* | $\hat{RM}_{\hat{r}} = \{\rho \mid r(\rho) = \hat{r}\}$, $\hat{RM} = \bigcup_{\hat{r} \in \hat{R}} \hat{RM}_{\hat{r}}$ |
| *der Namen der von den Rollen referenzierten Objektdienste* | $\hat{DN} = \bigcup_{\hat{r} \in \hat{R}} D^O_{Ref}(\hat{r})$ |
| *der Objekte* | $\hat{O} = \bigcup_{p \in P} \gamma\|_O(p)$ |
| *der Interaktionsobjekte* | $\hat{IO} \subseteq \{\hat{o} \in \hat{O} \mid D^I_{Ref} = \emptyset\}$ |
| *der Zustände der Objekte* | $\hat{ZO} = \bigcup_{\hat{o} \in \hat{O}} \{\hat{S} \mid \hat{S}$ Sorte von $z(\hat{o})_i, 1 \le i \le \|z(\hat{o})\|\}$ |
| *der Objektmarken* | $\hat{OM}_{\hat{o}} = \{\omega \mid o(\omega) = \hat{o}\}$, $\hat{OM} = \bigcup_{\hat{o} \in \hat{O}} \hat{OM}_{\hat{o}}$ |
| *der Marken von Interaktionsobjekten* | $\hat{IM} = \bigcup_{\hat{o} \in \hat{IO}} \hat{OM}_{\hat{o}}$ |
| *der von den Objekten bereitgestellten Dienste* | $\hat{DF} = \bigcup_{\hat{o} \in \hat{O}} D_{Def}(\hat{o})$ |

Tabelle 3.14: Definition der Sorten des Interaktionsnetzes

Wir erweitern die Definition der Menge der *Funktionssymbole* $\Phi$ wie in Tabelle 3.15. Dabei bezeichnet wie bisher $\pi_i$ die *Projektion* auf die $i$-te Komponente des Zustandsvektors, mit $\kappa$ bezeichnen wir wie bisher die *Komposition* einer Rolle bzw. eines Objektes und mit $\varsigma$ die *Substitution* der Rolle einer Rollenmarke (auch Rollenwechsel

genannt), respektive des Objekts einer Objektmarke (auch Objektwechsel genannt) und $\varphi$ solche Funktionen, welche durch Methoden der Zustandselemente im Sinne der Objektorientierung gegeben sind.

Mit $\nu$ bezeichnen wir wie bisher den *Namenskontext* zwischen referenzierenden Diensten einer Rolle und definierenden Diensten eines Objektes für ein Paar aus Rollen- und Objektmarke. Neu hinzugekommen ist $\nu$ als *Namenskontext* zwischen den von einem Objekt referenzierten Diensten eines Interaktionsobjektes und den definierenden Diensten eines Interaktionsobjektes.

Weiterhin sei $X \cup XR$ die Menge der *typgebundenen Variablen*, aus der insbesondere *Individuenvariablen* des Typs $\hat{RM}$ bzw. $\hat{OM}$ und *Mengenvariablen* des Typs $2^{\hat{RM}}$ bzw. $2^{\hat{OM}}$ verwendet werden. In *XR* sind die Referenzvariablen zusammengefaßt ($X \cap XR = \emptyset$), Mit $Term(\Phi)$ sei wie üblich die Menge der *Terme* über $\Sigma$ definiert.

| Menge der *Funktionssymbole ...* | $\Phi = \Phi_{\hat{R}} \cup \Phi_{\hat{O}} \cup \Phi_{\hat{RMOM}} \cup \Phi_{\hat{OMIM}} \cup \Phi_{\hat{Z}}$ |
|---|---|
| *auf Rollen und Rollenmarken* | $\Phi_{\hat{R}} = \{\pi_i \mid \pi_i\colon \hat{RM} \to \hat{Z}_{\hat{R}}, i\} \cup \{\varsigma \mid \varsigma\colon \hat{RM} \times \hat{R} \to \hat{RM}\} \cup \bigcup_{\hat{r} \in \hat{R}} \{\kappa_{\hat{r}} \mid \kappa_{\hat{r}}\colon (\hat{S}_1 \times \ldots \times \hat{S}_{\lvert z(\hat{r})\rvert}) \to \{\hat{r}\};\ \hat{S}_i \text{ Sorte von } z(\hat{r})_i, \forall i\}$ |
| *auf Objekten und Objektmarken* | $\Phi_{\hat{O}} = \{\pi_i \mid \pi_i\colon \hat{OM} \to \hat{Z}_{\hat{O}}, i\} \cup \{\varsigma \mid \varsigma\colon \hat{OM} \times \hat{O} \to \hat{OM}\} \cup \bigcup_{o \in \hat{O}} \{\kappa_{\hat{o}} \mid \kappa_{\hat{o}}\colon \hat{S}_1 \times \ldots \times \hat{S}_{\lvert z(\hat{o})\rvert} \to \{\hat{o}\};\ \hat{S}_i \text{ Sorte von } z(\hat{o})_i\}$ |
| *auf Rollen- und Objektmarken* | $\Phi_{\hat{RMOM}} = \{\nu \mid \nu\colon \hat{RM} \times \hat{OM} \to \hat{RM}\}$<br>$\Phi_{\hat{OMIM}} = \{\nu \mid \nu\colon (\hat{OM} - \hat{IM}) \times \hat{IM} \to \hat{OM}\}$ |
| *mit Typisierung in* $\hat{Z}$ | $\Phi_{\hat{Z}} = \{\varphi \mid \varphi\colon \hat{S}_1 \times \ldots \times \hat{S}_k \to \hat{S}_{k+1};\ \hat{S}_i \in \hat{Z}_{\hat{R}} \cup \hat{Z}_{\hat{O}}\}$ |

Tabelle 3.15: Definition der Funktionssymbole des Interaktionsnetzes

2. Die $\Sigma$*-Algebra* $A = (\hat{D}, \hat{F})$ sei ein Modell für die Signatur $\Sigma$, wobei $\hat{D}$ die *Grundmenge* entsprechend Tabelle 3.16 bezeichnet. Die *Interpretation der Funktionssymbole* auf Rollen und Rolleninstanzen ($\Phi_{\hat{R}}$ wird interpretiert von $\hat{F}_{\hat{R}}$), auf Objekten und Objektinstanzen ($\Phi_{\hat{O}}$ wird interpretiert von $\hat{F}_{\hat{O}}$), auf Rollen- und Rolleninstanzen ($\Phi_{\hat{RMOM}}$ wird interpretiert von $\hat{F}_{\hat{RMOM}}$, $\Phi_{\hat{OMIM}}$ wird interpretiert von $\hat{F}_{\hat{OMIM}}$) und auf Zustände der Rollen und der Objekte ($\Phi_{\hat{Z}}$ wird interpretiert von $\hat{F}_{\hat{Z}}$) bezeichnen wir mit $\hat{F} = \hat{F}_{\hat{R}} \cup \hat{F}_{\hat{O}} \cup \hat{F}_{\hat{RMOM}} \cup \hat{F}_{\hat{OMIM}} \cup \hat{F}_{\hat{Z}}$ .

   Die Interpretation $\hat{F}_{\hat{R}}$ und $\hat{F}_{\hat{O}}$ der Funktionssymbole auf Rollen und Objekten ist dabei wie folgt (vgl. Tabelle 3.17): Die Interpretation der *Projektion* legen wir als den Zugriff auf die *i*-te Komponente des Zustandsvektors der Rollen $\rho$ bzw. des Objekts $\omega$ fest. Die Interpretation der *Komposition* legen wir als die Instantiierung der Rolle *R* bzw. des Objekts *O* fest. Die Interpretation der *Substitution* mit einer *Komposition*

| *Grundmenge ...* | $\hat{D} = (\hat{D}_{\hat{R}} \cup \hat{D}_{\hat{ZR}} \cup \hat{D}_{\hat{RM}} \cup \hat{D}_{2^{\hat{RM}}} \cup \hat{D}_{\hat{DN}}) \cup (\hat{D}_{\hat{O}} \cup \hat{D}_{\hat{ZO}} \cup \hat{D}_{\hat{OM}} \cup \hat{D}_{2^{\hat{OM}}} \cup \hat{D}_{\hat{DF}})$ |
|---|---|
| *der Rollen* | $\hat{D}_{\hat{R}} = \bigcup_{\hat{r} \in \hat{R}} D_{\hat{r}}$ |
| *der Zustände der Rollen* | $\hat{D}_{\hat{ZR}} = \bigcup_{\hat{zr} \in \hat{ZR}} D_{\hat{zr}}$ |
| *der Rollenmarken* | $\hat{D}_{\hat{RM}} = \bigcup_{\hat{rm} \in \hat{RM}} D_{\hat{rm}}$ |
| *der Dienstnamen* | $\hat{D}_{\hat{DN}} = \bigcup_{\hat{dn} \in \hat{DN}} D_{\hat{dn}}$ |
| *der Objekte* | $\hat{D}_{\hat{O}} = \bigcup_{\hat{o} \in \hat{O}} D_{\hat{o}}$ |
| *der Zustände der Objekte* | $\hat{D}_{\hat{ZO}} = \bigcup_{\hat{zo} \in \hat{ZO}} D_{\hat{zo}}$ |
| *der Objektmarken* | $\hat{D}_{\hat{OI}} = \bigcup_{\hat{om} \in \hat{OM}} D_{\hat{om}}$ |
| *der Dienste* | $\hat{D}_{\hat{DF}} = \bigcup_{\hat{df} \in \hat{DF}} D_{\hat{df}}$ |

Tabelle 3.16: Grundmenge der $\Sigma$-Algebra

legen wir als die Zuweisung einer Rolle *R* bzw. Objekt *O* an eine existierende Rollenmarke $\rho$ bzw. Objektmarke $\omega$ fest. Die Interpretation des *Kontextes* legen wir für $\hat{RM} \times \hat{OM} \to \hat{RM}$ als die Abbildung (3.1) fest, von der Menge der referenzierenden Dienste der Rolle einer Rollenmarke auf die Menge der definierenden Dienste des Objekts einer Objektmarke

$$\nu\colon (\mathrm{D}^{O}_{\mathrm{Ref}}(\mathrm{r}(\rho)) \cup \mathrm{D}^{I}_{\mathrm{Ref}}(\mathrm{r}(\rho))) \to \mathrm{D}_{\mathrm{Def}}(\mathrm{o}(\omega)) \tag{3.1}$$

und für $(\hat{OM} - \hat{IM}) \times \hat{IM} \to \hat{OM}$ legen wir die Interpretation des *Kontextes* als die Abbildung (3.2) fest, von der Menge der referenzierenden Dienste des Objekts einer Objektmarke auf die Menge der definierenden Dienste eines Objekts einer Interaktionsmarke. Zusätzlich zur Bindung wird bei Ausführung eines *Kontext- und Aufruf-terms* ($\omega' = \nu(\omega, \omega^{I})$) der gebundene Dienst (nicht blockierend) aufgerufen.

$$\nu\colon \mathrm{D}^{I}_{\mathrm{Ref}}(\mathrm{o}(\omega)) \to \mathrm{D}_{\mathrm{Def}}(\mathrm{o}(\omega^{I})) \tag{3.2}$$

Sei weiterhin $\beta$ eine *Belegung* der Variablen, welche über den Aufbau der Terme fortgesetzt wird. Wir interpretieren einen Term $\tau \in Term(\Phi)$ als *booleschen Term*, falls $\beta(\tau) \in \mathbb{B}$. Weiterhin bezeichnen wir einen Term $\rho' = \varsigma(\rho, \kappa_R(\overrightarrow{z_R}))$ bzw.

| **Term** | **vor Ausführung** | **nach Ausführung** |
|---|---|---|
| *Projektion:* $z = \pi_i(\rho)$ $z = \pi_i(\omega)$ | ./. | $z = \mathrm{z}(r(\rho))[i]$ $z = \mathrm{z}(o(\omega))[i]$ |
| *Komposition:* $r = \kappa_R(\overrightarrow{z_R})$ $o = \kappa_O(\overrightarrow{z_O})$ | ./. | $r = (\mathrm{D}^A_{\mathrm{Ref}}(R), \mathrm{D}^O_{\mathrm{Ref}}(R), \mathrm{D}^I_{\mathrm{Ref}}(R), \overrightarrow{z_R}, \mathrm{ak}(R))$ $o = (\mathrm{D}_{\mathrm{Def}}(O), \mathrm{D}^{RS}_{\mathrm{Ref}}(O), \mathrm{D}^I_{\mathrm{Ref}}(O), \mathrm{D}_{\mathrm{Impl}}(O), \overrightarrow{z_O})$ |
| *Substitution* mit einer *Komposition:* $\rho' = \varsigma(\rho, \kappa_R(\overrightarrow{z_R}))$ $\omega' = \varsigma(\omega, \kappa_O(\overrightarrow{z_O}))$ | $\rho = (\mathrm{r}(\rho), ag, \nu_{ag})$, mit $\mathrm{r}(\rho) = (D^A_{Ref}, D^O_{Ref}, \vec{z}, ak)$ | $\rho' = (\mathrm{r}(\rho'), ag, \nu^R_{ag})$, mit $\mathrm{r}(\rho') = (\mathrm{D}^A_{\mathrm{Ref}}(R), \mathrm{D}^O_{\mathrm{Ref}}(R), \overrightarrow{z_R}, \mathrm{ak}(R))$ |
| | $\omega = (\mathrm{o}(\omega), rs, \nu_{rs})$, mit $\mathrm{o}(\omega) = (D_{Def}, D^{RS}_{Ref}, D^I_{Ref}, D_{Impl}, \vec{z})$ | $\omega' = (\mathrm{o}(\omega'), rs, \nu^O_{rs})$, mit $\mathrm{o}(\omega') = (\mathrm{D}_{\mathrm{Def}}(O), \mathrm{D}^{RS}_{\mathrm{Ref}}(O), \mathrm{D}^I_{\mathrm{Ref}}(O), \mathrm{D}_{\mathrm{Impl}}(O), \overrightarrow{z_O})$ |
| *Kontext:* $\rho' = \nu(\rho, \omega)$ | $\rho = (r, ag, \nu_{ag})$ | $\rho' = (r, ag, \nu_{ag} \cup \nu(\rho, \omega))$ |
| *Kontext und Aufruf:* $\omega' = \nu(\omega, \omega^I)$ | $\omega = (\mathrm{o}(\omega), rs, \nu_{rs})$ | $\omega' = (\mathrm{o}(\omega), rs, \nu_{rs} \cup \nu(\omega, \omega^I))$ |

Tabelle 3.17: Interpretation der Funktionssymbole der $\Sigma$-Algebra

$\omega' = \varsigma(\omega, \kappa_O(\overrightarrow{z_O}))$ als *Substitutionsterm* und einen Term der Form $\rho' = \nu(\rho, \omega)$ als *Kontextterm*. Schließlich wird ein Term, welcher ausschließlich aus einer Individuenvariable $r$ der Sorte $\hat{RM}$ bzw. $o$ der Sorte $\hat{OM}$ oder aus einer Mengenvariable der Sorte $2^{\hat{RM}}$ bzw. der Sorte $2^{\hat{OM}}$ besteht, als *Deklarationsterm* bezeichnet (in Zeichen $r$: $R$ bzw. $o$: $O$ oder $\forall r$: $R$ bzw. $\forall o$: $O$). Steht der Variable als "syntaktischer Zukker" das Zeichen '&' voran, so wird durch den Deklarationsterm eine Referenzvariable deklariert.

3. Ein *Interaktionsnetz* ist dann ein Tupel $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ so, daß gilt:

   - $P$ ist eine endliche Menge von *Phasen*,
   - $T$ ist eine endliche Menge von Phasenübergängen (*Transitionen*),
   - $F \subset P \times T \cup T \times P$ ist die *Flußrelation*.

   Mit $T_E$ bezeichnen wir die Eingangstransitionen, welche einen leeren Vorbereich haben. Von der Flußrelation wird gefordert, daß der Nachbereich einer Eingangstransition einelementig ist.

- $\sigma$ ist eine *Beschriftung* von Phasen, Kanten und Transitionen, wobei

  - jede Phase $p \in P$ mit der Menge der Rollen- bzw. Objektmarken beschriftet ist, welche eine Rolle bzw. ein Objekt aus $p$ verkörpern (Typisierung von $p$):
    $\sigma(p) = \bigcup_{\hat{r} \in \gamma|_R(p)} RM_{\hat{r}} \cup \bigcup_{\hat{o} \in \gamma|_O(p)} OM_{\hat{o}}$
  - jede Kante $(p, t) \in F \cap P \times T$ mit einer endlichen Menge von ggf. mit einem Allquantor versehenen Deklarationstermen konsistent[1] beschriftet ist:
    $\sigma(p, t) = \{r_1: R_1, \ldots, \forall r_i: R_i, o_1: O_1, \ldots, \forall o_j: O_j, \ldots\}$
  - jede Kante $(t, p) \in F \cap T_E \times P$ mit einem Term beschriftet ist:
    $\sigma(t, p) \in \{\mathsf{newToken} \leftarrow R(\overrightarrow{z_R}) \mid R \in \sigma(p)\} \cup$
    $\{\mathsf{newToken} \leftarrow O(\overrightarrow{z_O}) \mid O \in \sigma(p)\}$
  - jede Kante $(t, p) \in F \cap (T \setminus T_E) \times P$ mit einer endlichen Menge von Termen konsistent[2] beschriftet ist:
    $\sigma(t, p) = \{r_k \leftarrow R_k(\overrightarrow{z_{R_k}}) \mid k\} \cup \{o_l \leftarrow O_l(\overrightarrow{z_{O_l}}) \mid l\} \cup$
    $\{\nu(Dn(r)) := Df(o) \mid r, o\} \cup$
    $\{\nu(Dn(o)) := Df(oi) \mid o, oi\}$
  - und jede Transition $t \in T$ mit einem booleschen Term (Wächter) konsistent[3] beschriftet ist.
  - von $\sigma$ gefordert wird, daß das Schaltverhalten einer Transition hinsichtlich der Rollenmarken markentreu ist. D. h. die Anzahl und Identität der abgezogenen Rollenmarken entspricht beim Schalten einer Transition der Anzahl und Iden-

1. Eine Kante $(p, t) \in F \cap P \times T$ heißt genau dann mit einer endlichen Menge von *Deklarationstermen* $\sigma(p, t) = \{r_1: R_1, \ldots, \forall r_i: R_i, o_1: O_1, \ldots, \forall o_j: O_j, \ldots\}$ *konsistent beschriftet*, wenn die Deklaration jeder Variablen in der Transition eindeutig ist. Oder formal ausgedrückt, wenn für jeden Term $\tau \in \sigma(p, t), \tau = v: V$ oder $\tau = \forall v: V \;\; V \in \gamma(p)$ gilt, sowie $v \notin \sigma(q, t)$ für alle $q \in \cdot t \setminus \{p\}$ gilt.
2. Eine Kante $(t, p) \in F \cap T \times P$ heißt genau dann mit einer endlichen Menge von Termen $\sigma(t, p) = \{r_k \leftarrow R_k(\overrightarrow{z_{R_k}}) \mid k\} \cup \{o_l \leftarrow O_l(\overrightarrow{z_{O_l}}) \mid l\} \cup \{\nu(r, o) \mid r, o\} \cup \{\nu(o, oi) \mid o, oi\}$ *konsistent beschriftet*, wenn alle verwendeten Variablen auch definiert sind und wenn Kontextterme dieser Kante nur auf Marken definiert sind, welche über diese Kante geschaltet werden. Referenzvariablen dürfen ausschließlich in Kontexttermen verwendet werden, unter der Einschränkung, daß die Kante die sie beschriften wieder in die Phase zurückführen muß, von der die Kante ausging, welche mit dem Deklarationsterm der Referenzvariable beschriftet ist. Oder formal ausgedrückt, wenn für jeden Substitutionsterm $\tau \in \sigma(t, p), \tau = v \leftarrow V(\overrightarrow{z_V}) \;\; V \in \gamma(p) \;\; v \in X$ gilt, sowie $\exists q \in \cdot t$, so daß $v \in \sigma(q, t)$ und wenn für jeden Kontextterm $\tau \in \sigma(t, p), \tau = \nu(Dn(r)) := Df(o)$ sowohl $r$ als auch $o$ auf der linken Seite eines Substitutionsterms aus $\sigma(t, p)$ vorkommen und analog für $\tau = \nu(Dn(o)) := Df(oi)$ sowohl $o$ als auch $oi$ auf der linken Seite eines Substitutionsterms aus $\sigma(t, p)$ vorkommen. Ist $r \in XR$ oder $o \in XR$, mit $r \in \sigma(q, t)$ oder $o \in \sigma(q, t)$, so gilt $q = p$.
3. Eine Transition $t \in T$ heißt genau dann mit einem booleschen Term $\tau = \sigma(t)$ *konsistent beschriftet*, wenn alle verwendeten Variablen auch definiert sind. Oder formal ausgedrückt, wenn jede freie Variablen in $\tau$ durch genau einen Deklarationsterm $\tau' \in \sigma(p, t)$ mit $p \in \cdot t$ gebunden ist.

tität der abgelegten Rollenmarken. Im Gegensatz dazu dürfen Objektmarken dupliziert und konsumiert werden. Transitionen ohne Nachbereich ziehen ausschließlich Marken ab und können somit auch Rollenmarken konsumieren.

- $A$ ist ein Modell für die Signatur $\Sigma$
- $M_0: P \to 2^{\hat{D}_{R\hat{M}} \cup \hat{D}_{O\hat{M}}}$ ist die *Anfangsmarkierung*, die jeder Phase $p \in P$ eine Menge von Rollen- und Objektmarken aus der Grundmenge der zu $p$ gehörenden Rollen- und Objektmarken $\sigma(p)$ zuweist.

Das Interaktionsnetz ist somit eine auf Pr/T-Netzen aufbauende Petrinetzklasse, welche um das Rollen-, das Objekt und das Modulkonzept erweitert ist.

Die *Schaltregel* für Interaktionsnetze ist nun um den Einfluß von Referenzvariablen auf das Schaltverhalten zu erweitern. Sie wird im folgenden vollständig angegeben.

**Definition 3.33** *Schaltregel für Interaktionsnetze*

Sei $N_I = (P, T; F, \Sigma, \sigma, A, M_0)$ ein Interaktionsnetz. Die Schaltregel wird über die Anwendbarkeit einer Belegung, die Aktiviertheit einer Transition und die Folgemarkierung nach dem Schalten der Transition definiert:

1. Sei $t \in T$ eine Transition und $\tau_t = \sigma(t)$ die Wächterbedingung von *t*. Eine Belegung $\beta: Term(\Phi) \to \hat{D}$ heißt *anwendbar*, falls sie alle folgenden Fälle erfüllt:

   - Die Belegung $\beta$ wertet $\tau_t$ unter der Algebra *A* nach *wahr* aus.
   - Die Belegung jedes Deklarationsterms der Form $(\forall r)$: $R$ bzw. $(\forall o)$: $O$ ist *maximal* (es gibt keine weitere Marke aus der aktuellen Markierung, die zu der Belegung des Deklarationsterms hinzugefügt werden könnte, so daß der Wächterterm weiterhin nach *wahr* auswertet).
   - Die Belegung ist durch keine Objektmarke $\omega$ *-blockiert*:

     Sei $\omega$ eine Objektmarke, mit der eine Variable (nicht eine Referenzvariable) belegt wird ( $p \in \cdot t, \tau \in \sigma(p, t), \tau \in X_{O\hat{M}}, \beta(\tau) = \omega$ ).

     Eine Belegung $\beta$ heißt $\omega$*-blockiert*, wenn es eine nicht am Schaltvorgang beteiligte, aktivierte Rollenmarke gibt $(\exists \rho \in M(p), \rho \notin \{\rho \mid \rho = \beta(\tau), \tau \in \sigma(p, t)\}$ und $ak(r(\rho))$ nicht terminiert), welche Dienste der Objektmarke nutzt $(\exists Dn \in D^{O}_{Ref}(r(\rho)) \cup D^{I}_{Ref}(r(\rho)), \exists Df \in D_{Def}(o(\omega))$: $(Dn, Df) \in \nu(\rho)$ ).

2. Sei der Schaltmodus einer Transition $t$ definiert durch die Belegung $\beta$, kurz $t[\beta]$. Eine Transition ist im Schaltmodus $t[\beta]$ genau dann aktiviert ($t$ ist $\beta$-aktiviert), wenn gilt $\tau[\beta] \subseteq M(s)$ für alle $p \in {\cdot}t$ und $\tau \in \sigma(p, t)$.
3. Ist $t$ unter der Markierung $M$ $\beta$-aktiviert und schaltet $t$ im Modus $t[\beta]$, so ergibt sich folgende Markierung.

$$M'(s) = \begin{cases} M(s) & \text{für } s \notin {\cdot}t \cup t{\cdot} \\ M(s) - \beta(\sigma(s,t) - XR) & \text{für } s \in {\cdot}t - t{\cdot} \\ M(s) + \beta(\sigma(t,s)) & \text{für } s \in t{\cdot} - {\cdot}t \\ M(s) - \beta(\sigma(s,t)) + \beta(\sigma(t,s)) & \text{für } s \in {\cdot}t \cap t{\cdot} \end{cases} \tag{3.3}$$

Die Schreibweise $M[t[\beta] > M'$ bezeichne, daß die Transition $t$ im Modus $t[\beta]$ aktiviert ist und durch das Schalten von $t$ die Markierung $M$ in die Markierung $M'$ gemäß (3.3) überführt wird.

Ist eine Transition $t \in T$ $\beta$-aktiviert, so schaltet sie sofort.

**Bemerkung 3.19**

Der Schaltvorgang einer Transition ist eine atomare Aktion. Eine Marke kann mehrere Transitionen aktivieren, jedoch nicht an zwei Schaltvorgängen gleichzeitig beteiligt sein. Die Einführung der Referenzvariablen auf Marken bildet dabei keinen Unterschied: Eine Marke, mit der eine Referenzvariable belegt wurde kann nicht gleichzeitig am Schaltvorgang einer anderen Transition teilnehmen.

Hierdurch ist sichergestellt, daß eine Objektmarke, an welche eine Rollenmarke durch Schalten der Transition $A$ gebunden wird, nicht durch eine gleichzeitig schaltende Transition $B$ aus der Phase entfernt werden kann.

Die Einführung der Referenzvariablen ist also eine Kurzschreibweise für ein Abziehen einer Marke aus einer Phase und ein unverändertes Zurücklegen der Marke in die selbe Phase. Eine laufende Aktion einer Rollenmarke wird dabei nicht unterbrochen, Zugriffe auf eine Objektmarke und laufende Subinteraktionsverfahren werden nicht beeinflußt.

Referenzvariablen werden verwendet, um bei einem Schaltvorgang

- eine Rollenmarke an ein neues Objekt zu binden (Rollenmarke belegt Referenzvariable, Objektmarke wird in die Phase geschaltet) oder
- an eine in der Phase befindlichen Objektmarke eine neu in die Phase geschaltete Rollenmarke zu binden (Objektmarke belegt Referenzvariable, Rollenmarke wird in die Phase geschaltet) oder

- eine Rollenmarke an eine Objektmarke zu binden, wenn sich beide in der selben Phase befinden (Objektmarke und Rollenmarke belegen Referenzvariablen) oder
- analog zu den obigen drei Fällen, eine Objektmarke, welche nicht ein Interaktionsobjekt ist, an ein Interaktionsobjekt zu binden.

Ein Objekt kann in den ersten drei Fällen immer auch ein Interaktionsobjekt sein.

Der Schaltvorgang wurde bereits in Definition 3.23 beschrieben und erfordert keine Erweiterung. Das Terminieren und das Blockieren eines Interaktionsverfahrens ist wie in Definition 3.10 und Definition 3.11 definiert.

### 3.5.5 Anforderungen an Agenten und Ressourcen

Bevor ein Agent oder eine Ressource über einen Eingang ein Interaktionsverfahren (repräsentiert durch eine Transition ohne Vorbereich) betreten darf, muß sichergestellt werden, daß der Agent oder die Ressource die Anforderungen, welche der gewählte Eingang an ihn stellt, auch erfüllen kann.

Die Anforderungen ergeben sich aus dem Erreichbarkeitsgraph (der Rollenkette oder der Objektkette), welchen die durch die Eingangstransition erzeugte Marke während ihres Lebenszyklus innerhalb des Interaktionsverfahrens durchläuft. Jeder Knoten des Erreichbarkeitsgraphen ist durch den Markentyp (Rolle oder Objekt) definiert, welcher der Marke in einer Phase zugeordnet wird. Jeder Markentyp referenziert gewisse Dienste $D^{A}_{Ref}$ des Agenten bzw. $D^{RS}_{Ref}$ der Ressouce, welche diese auch bereitstellen müssen.

Ein Eingang ist dann für einen Agenten oder eine Ressource passierbar, falls die referenzierten Dienste aller von der erzeugten Marke aus erreichbaren Markentypen unterstützt werden (vgl. Definition 3.14 und Definition 3.26).

### 3.5.6 Erweiterung des Auktionsbeispiels

Die Modellierung des Auktionsbeispiels in den vorangegangenen Iterationsstufen des Interaktionsmodells führte zu flachen Netzen, welche nicht modularisierbar waren. In dieser Iterationsstufe führten wir die Modularisierung ein und werden sie in dem Auktionsbeispiel exemplarisch anwenden.

Der Entwurf sieht zwei Interaktionsverfahren vor: eines, welches die Versteigerung eines einzelnen Gegenstands vornimmt und eines, welches unter Verwendung des Ersten die Versteigerung eines Katalogs von Gegenständen steuert (vgl. Abbildung 3.13).

Beide Interaktionsverfahren realisieren das Muster für die Modellierung von einer Menge zulässiger Anfangsmarkierungen, wobei für beide die Menge einelementig ist.

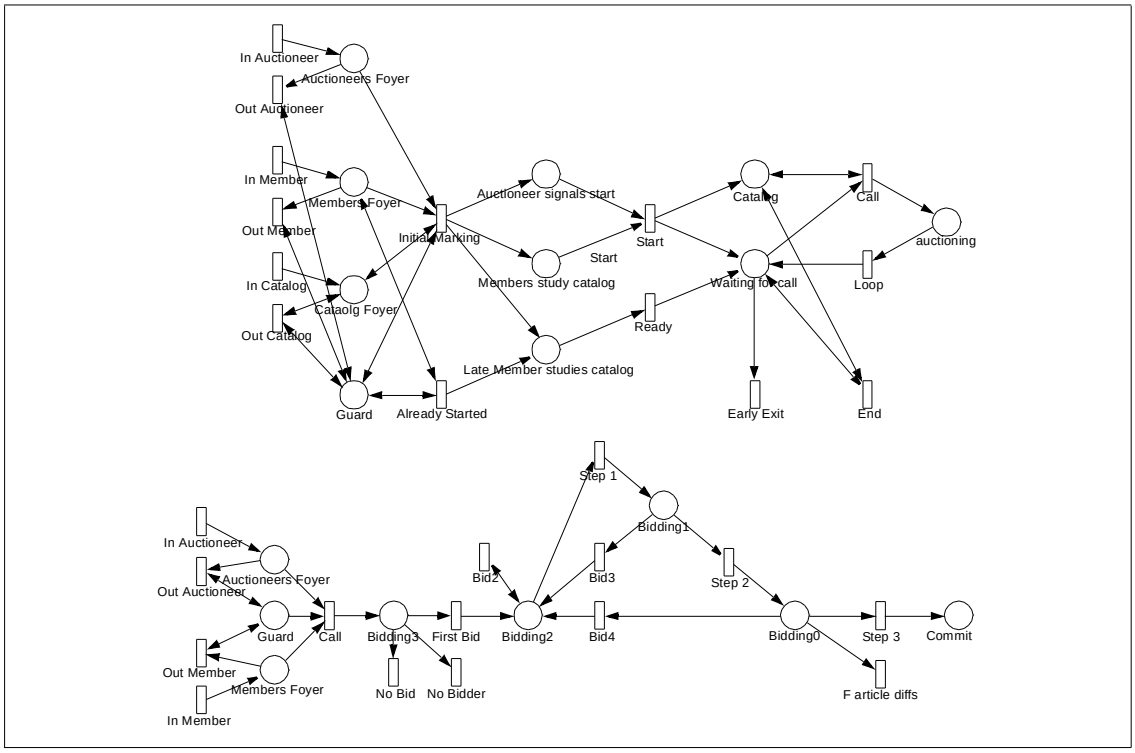


Abbildung 3.13: Netzstruktur des Interaktionsverfahrens Auktion: Das obere Verfahren besitzt in der Phase “auctioning” ein Interaktionsobjekt, welches die Schnittstelle zum unten dargestellten Subinteraktionsverfahren bildet

1. Das die Versteigerung organisierende Interaktionsverfahren:

   Nachdem die Vorbedingung für die Eröffnung des Versteigerung erfüllt ist (der Katalog ist erstellt (Marke in “Catalog foyer”), ein Auktionator ist benannt (Marke in “Auctioneer foyer”) und die Frist vor dem eigentlichen Start der Auktion ist erreicht (definiert durch die Objektmarke in “Guard”)), wird das Interaktionsverfahren “Auktion” durch Schalten der Transition “Initial Marking” eröffnet. Dazu wird allen Auktionsteilnehmern, welche sich für die Teilnahme an dem Interaktionsverfahren registriert haben, die Möglichkeit gegeben eine bestimmte Zeit lang den Katalog zu studieren (“Members study catalog”). Der Termin für den Beginn der Auktion ist allen bekannt; der Auktionator signalisiert nach Erreichen des Termins den Start (“Auctioneer signals start”).

   Auktionsteilnehmer, welche sich zu spät für das Interaktionsverfahren “Auktion” registrieren haben lassen, können bis kurz bevor der letzte Gegenstand zur Versteigerung aufgerufen wird noch verspätet an der Auktion teilnehmen. Hierzu betreten sie das Interaktionsverfahren über den ihnen bekannten Eingang “In member”, jedoch werden sie durch die Transition “Already started” in die Phase “Late member studies catalog” geführt, wo sie den fortlaufend aktualisierten Katalog der noch zur Versteigerung anstehenden Gegenstände einsehen (beim Schalten der Transition

"Initial marking" wurde ein Duplikat der Objektmarke für den Katalog in diese Phase abgelegt). Daraufhin betreten sie die Wartephase vor dem Aufruf des nächsten Versteigerungsgegenstandes.

Die Phase "Waiting for call" dient dem Verweilen, falls ein Gegenstand versteigert wird, an dem ein Teilnehmer nicht interessiert ist. Diese Phase ist darüberhinaus auch für den koordinierten Zutritt von verspäteten Teilnehmern vorgesehen, welche wie bereits beschrieben asynchron auf die laufende Interaktion vorbereitet wurden. Über diese Phase kann ein Teilnehmer auch frühzeitig die Auktion verlassen (Transition "Early exit"). Nach Versteigerung aller Gegenstände wird das Verfahren hier verlassen (Transition "End").

Die eigentliche Versteigerung findet nach Aufruf des zugehörigen Gegenstandes statt, entsprechend der Reihenfolge, in der die Gegenstände im Katalog aufgeführt sind (Schalten der Transition "Call"). Dazu werden alle Teilnehmer geschaltet, welche Interesse an dem Gegenstand bekundet haben. Gemeinsam mit dem Auktionator wird in der Phase "auctioning" die Versteigerung des gewählten Gegenstandes als Subinteraktionsverfahren gestartet. Falls kein Teilnehmer Interesse an dem Gegenstand hat, betritt der Auktionator das Subinteraktionsverfahren allein. Nach Abschluß des Interaktionsverfahrens werden alle Rollen gemeinsam durch die Transition "Loop" in die Phase "Waiting for call" geschaltet.

2. Der Aufruf des Subinteraktionsverfahrens:

   Der Aufruf des Subinteraktionsverfahrens ist ein Zusammenspiel der eingehenden Transitionen, der in der Phase definierten Rollen, der das Subinteraktionsverfahren repräsentierende Interaktionsobjektmarke und der ausgehenden Transitionen.

   Die Interaktionsobjektmarke wurde in diesem Beispiel durch die Anfangsmarkierung des die Versteigerung organisierenden Interaktionsverfahren in der Phase "auctioning" erzeugt. Das dieser Marke zugeordnete Interaktionsnetz wird durch Eintritt der ersten Marke aktiviert und existiert so lange, bis die letzte Rollenmarke das Netz über eine Ausgangstransition verlassen hat (vgl. Definition 3.10 über das Terminieren eines Interaktionsverfahrens).

   Die Transition "Call" entnimmt dem Versteigerungskatalog (Kante zur Phase "Catalog") den nächsten zu versteigernden Gegenstand, aus der Phase "Waiting for call" alle an diesem Gegenstand interessierten Auktionsteilnehmer, sowie den Auktionator, initialisiert die Interaktionsobjektmarke der Phase "auctioning" und bindet die Rollenmarken an die Interaktionsobjektmarke. Anschließend ist die Transition solange $\omega$-blockiert (vgl. Definition 3.33 über die Schaltregel für Interaktionsnetze), bis alle Rollenmarken das Subinteraktionsverfahren verlassen haben.

   Eine Rollenmarke verläßt das Subinteraktionsverfahren, welches eine einzelne Versteigerung modelliert, im Regelfall gemeinsam mit allen beteiligten Rollenmarken durch die Transition "End". Es existieren jedoch noch asynchrone Ausgänge, welche noch zu beschreiben sind, über die eine Rollenmarke frühzeitig das Subinteraktions-

verfahren verlassen kann. In einem solchen Fall wird die Rollenmarke im vorliegenden Entwurf nicht aus der Phase “auctioning” entlassen, sondern wartet dort auf alle anderen Rollenmarken.

Die Transition “Loop” schaltet alle Rollenmarken gemeinsam aus der Phase “auctioning” in die Phase “Waiting for call”. Dies erfolgt jedoch erst, wenn alle Rollenmarken deaktiviert sind. Somit wird sichergestellt, daß alle Rollenmarken das Subinteraktionsverfahren verlassen haben, und die Phase “auctioning” abgeschlossen ist.

3. Das eine einzelne Versteigerung vornehmende Interaktionsverfahren:

   Das Subinteraktionsverfahren entspricht weitgehend dem Beispiel aus Abschnitt 3.3.5. Es ist um das Muster für die Modellierung von einer Menge zulässiger Anfangsmarkierungen erweitert, wobei das einzige Element dieser Menge durch die Transition “Call” beschrieben wird.

   Beim Entwurf der Schaltbedingung dieser Transition muß man folgende Überlegung berücksichtigen: Das Verfahren läßt es nicht zu, daß nach seiner Eröffnung verspätete Teilnehmer noch aufgenommen werden. Da der Eintritt in ein Interaktionsverfahren jedoch asynchron erfolgt, muß also festgelegt werden, unter welcher Bedingung das Verfahren eröffnet wird. Im Allgemeinen wird hierfür eine Zeitschranke verwendet, nach der begonnen wird. Wie immer ist die Frage nach der richtigen Wahl der Zeitschranke jedoch nicht zu beantworten.

   Im vorliegenden Beispiel werden jedoch alle Teilnehmer an dem Verfahren durch das Schalten einer Transition im aufrufenden Interaktionsverfahren an die Interaktionsobjektmarke gebunden. Damit ist die Gruppe der Teilnehmer identifiziert, welche sich an der Interaktion beteiligen. Die Schaltbedingung für die Eröffnung des Verfahrens wird also durch das Erreichen der Gruppengröße aller Interaktionsteilnehmer definiert. Im Regelfall eröffnet damit das Subinteraktionsverfahren schneller als durch die Verwendung einer Zeitschranke. Um jedoch Störungen abzufangen, bei denen es einigen Rollen nicht mehr möglich ist an dem Verfahren teilzunehmen (oder falls es möglich ist, daß der zugehörige Agent eine Teilnahme an dem Subinteraktionsverfahren in letzter Minute verweigert), kann eine zusätzliche, ausreichend groß definierte Zeitschranke die Fehlertoleranz erhöhen.

   Der Regelablauf des Verfahrens ist nun wie folgt: Der Auktionator kennt den zu versteigernden Gegenstand, der durch Schalten der Transition “Call” aufgerufen wird. Hierdurch wird er allen Auktionsteilnehmern bekanntgegeben. In der Phase “Bidding3” sind die Rollenmarken aufgefordert ein Erstgebot abzugeben, welches dem Mindestgebot entspricht. Die Transition “First bid” schaltet all diejenigen Rollenmarken, die zumindest bereit sind das Mindestgebot abzugeben. Die Phasen “Bidding2”, “Bidding1” und “Bidding0” werden in dieser Reihenfolge durchlaufen, wenn kein weiteres Gebot abgegeben wird. Der Auktionator veranlaßt den Phasenübergang nach Ablauf einer durch das Verfahren festgelegten Zeitdauer. Jedes weitere Gebot veranlaßt einen Rücksprung in Phase “Bidding2”. Nach dreimaligem Verstreichen der Zeitdauer ist die Auktion zu Ende und der letzte Bieter erhält den Zuschlag

(Phase “Commiting”). Hier findet der Eigentumsübertrag und der Bezahlvorgang statt. Ist dies erfolgreich, so wird das Verfahren regulär verlassen, ansonsten wird ein Ausnahmefall signalisiert.

Betrachten wir nun mögliche Sonderfälle: Der Entwurf sieht vor, daß in der Phase “Bidding3” ein Teilnehmer, der nicht bereit ist, das Mindestgebot abzugeben, die Auktion vorzeitig durch die Transition “No bid” verläßt. Nun kann dies der einzige Auktionsteilnehmer gewesen sein, so daß das Verfahren durch Schalten der Transition “No bidder” den Auktionator entfernt und somit terminiert. Über diese Transition ist auch der bereits beschriebene Fall abgefangen, in dem bereits das Interaktionsverfahren ohne einen einzigen Auktionsteilnehmer eröffnet wurde.

Eine spezielle Funktion übernimmt die Transition “F article diffs” ein. Sie entspricht in Petrinetztheorie einem Fakt, welcher nie aktiviert wird. In dem vorliegenden Entwurf besitzt der Auktionator und die Auktionsteilnehmer jeweils ein eigenes Model von dem Gegenstand der versteigert wird. Dieses Modell wird beim Schalten der Transition “Call” vom Auktionator zu den Teilnehmern kopiert. Die Transition stellt nun sicher, daß auf dem Pfad durch das Interaktionsnetz die Modelle nicht inkonsistent geworden sind. Dies würde einen Implementierungsfehler im Interaktionsverfahren anzeigen und wird über diese Transition abgefangen und eine entsprechende Ausnahmebehandlung angestoßen.

### 3.5.7 Zusammenfassung

Das in diesem Kapitel vorgestellte Interaktionsmodell ist nun mit der Einführung des Modulskonzepts vollständig spezifiziert. Im Folgenden soll diskutiert werden, welche Modellelemente verantwortlich für die Erfüllung der in Abschnitt 3.1 aufgeführten Anforderungen sind:

1. Abstraktion

   In einem Interaktionsverfahren abstrahiert eine Rolle von Agenten, welche in einem bestimmten Abschnitt des Verfahrens (Phase) die selbe Aufgabe haben. Eine Rollenmarke abstrahiert von einem einzelnen Agenten, welcher die durch die Rolle gestellte Aufgabe individuell löst. Hierzu nutzt er die ihm gegebene Autonomie.

   Alle Interaktionsteilnehmer durchlaufen während einer Interaktion eine Folge von Rollenwechseln. Diese ist durch die Spezifikation des Interaktionsnetzes vorgegeben. Die Schaltsemantik stellt sicher, daß sich alle Teilnehmer entsprechend der Spezifikation verhalten.

2. Adaptivität und Wiederverwendung

   Die Schnittstelle zu einem Subinteraktionsverfahren wird über ein Interaktionsobjekt definiert. Das eigentliche Interaktionsnetz ist als Ressource an die Objektmarke gebunden, welches das Subinteraktionsverfahren repräsentiert. Ein aufrufendes Interaktionsverfahren kann durch diese Entkopplung unabhängig von der konkreten Realisierung des Subinteraktionsverfahren spezifiziert werden.

   Ein aufrufendes Interaktionsnetz kann über mehrere Interaktionsobjektmarken verfügen, unter welchen erst zur Laufzeit eines ausgewählt wird, mit dem die Interaktion fortgesetzt werden soll. Hierdurch ist es möglich, eine Interaktion an eine Situationen adaptieren zu lassen. Eine solche Situation könnte beispielsweise durch Möglichkeit oder der Bereitschaft zur Übernahme gewisser Rollen aus dem Subinteraktionsverfahren bestehen. Falls gewisse Rollen nicht besetzbar wären, müßte versucht werden, daß sich die Teilnehmer auf ein anderes Verfahren einigen.

   Die Wiederverwendbarkeit ist durch das Modulkonzept gewährleistet.

3. Wandelbarkeit

   Sieht man für das Interaktionsobjekt einen asynchronen Eingang vor, so läßt sich das zur Laufzeit aufzurufene Interaktionsverfahren auch nach Starten des aufrufenden Interaktionsverfahren auswählen. Somit könnte eine Überlappung zwischen Ausführung und Entwicklung von aufrufenden und aufzurufenden Interaktionsverfahren realisiert werden. Die Steuerung des Wandels wird dabei jedoch als eigenes Interaktionsverfahren verstanden, welches geeignet mit der zu wandelnden Interaktion zusammenspielen müßte.

4. Informiertheit

   Durch das Schalten einer Transition, werden die Rollen, welche die Transition aktivierten synchronisiert. Somit wird der Abschluß der vorangegangenen Phase und der Beginn der folgenden Phase für diese Rollen allgemein bekannt, wovon die Rollen in der folgenden Phase ausgehen können (vgl. Abschnitt 3.2.5 und Diskussion zum Auktionsbeispiel in Abschnitt 3.3.5).

5. Robustheit

   Das Modell sieht nicht die Existenz einer zentralen Komponente vor: Die Definition des Terminierens eines Interaktionsverfahrens (vgl. Definition 3.10) sieht vor, daß mit Ausscheiden der letzten Rollenmarke das Interaktionsverfahren beendet ist. Aufbauend auf dieser Definition kann die Realisierung der Steuerungskomponente des Interaktionsnetzes auf die teilnehmenden Agenten verteilt werden. Somit führt der Ausfall eines Agenten nicht zu einem Verlust des Interaktionszustandes.

   Im Interaktionsmodell wurde auf Ausnahmebehandlungen nicht konkret eingegangen. Im Allgemeinen wird der Ausfall eines Agenten zum Scheitern eines Interaktionsverfahrens führen. Im Beispiel der Auktion kann der Ausfall eines Agenten, der die Rolle eines Auktionsteilnehmers inne hat, durch das Interaktionsverfahren kompensiert werden, indem die zugehörige Marke aus dem Netz entfernt wird.

Allgemein müßte das Modell um einen speziellen Zustand der Rollenmarke erweitert werden, welche den Ausnahmefall “Agent nicht mehr erreichbar” symbolisiert. Das Schalten einer Transition mit dieser Marke würde dann eine Ausnahmebehandlung in der Transition anstoßen, welche z. B. die Marke aus dem Netz entfernen könnte. Grundsätzlich obliegt es dem Interaktionsverfahren, seine Robustheit hinsichtlich auftretender Ausnahmesituationen zu gewährleisten.

6. Offenheit

   Durch Einführung des Musters für asynchrone Ein- und Ausgänge wird diese Anforderung erfüllt (vgl. Abschnitt 3.5.1). Diese Anforderung wird auch an jedes einzelne Interaktionsverfahren gestellt.

Somit sind die Anforderungen, welche durch das Modell abgedeckt werden können erfüllt. Es bleiben jedoch wie bereits beschrieben, Anforderungen, welche von den zu entwerfenden Interaktionsverfahren zu erfüllen sind.

# Kapitel 4

# Agentenarchitektur in COMROS

Gegenstand des vorliegenden Kapitels ist die Vorstellung der Roboterarchitektur von Comros (Cooperative Mobile Robots Stuttgart). Kennzeichnend für sie sind autonome Funktionseinheiten, die als *Elementaragenten* modelliert sind und die über *Datenflußnetzwerke* arbeitsteilig Aufgaben ausführen. Die Funktionalität der Elementaragenten wird von ihren *Körpern* erbracht. Ihr *Kopf* hat die Aufgabe sich mit anderen Elementaragenten zu koordinieren, zu denen er in einer Abhängigkeitsbeziehung steht. Hierzu nimmt er an sogenannten *Entscheidungsnetzwerken* teil. Im Kapitel 5 wird die Realisierung eines Datenpuffers vorgestellt, welcher als Knoten in einem Datenflußnetzwerk Elementaragenten eines Teams von Robotern verbindet.

Nimmt man die funktionale Zerlegung des Systems in Elementaragenten auf der einen Seite und die Aufgaben, die Querschnittsprozesse über mehrere funktionale Einheiten darstellen auf der anderen Seite, so erkennt man daß die Aufgaben und die Elementaragenten Teil einer *Matrixorganisation* sind, die durch die Entscheidungsnetzwerke aufgespannt wird. Die Entscheidungsnetzwerke werden durch *Interaktionsverfahren* beschrieben, in denen der Kopf eines Elementaragenten *Rollen* annimmt.

Es fällt auf, daß durch die Konzeption der Architektur als Multiagentensystem die Systemgrenzen offen sind. Wie in Abschnitt 3.1 beschrieben, ist es in Multiagentensystemen möglich, daß während der Laufzeit neue Subsysteme hinzukommen. Darunter kann man sich weitere Sensor/Aktor-Module vorstellen, um welche ein Robotersystem für eine bestimmte Aufgabe erweitert wird. Darunter kann man sich aber auch den Zusammenschluß vorher eigenständiger Robotersysteme vorstellen, welche durch eine kooperative Ausführung einer Aufgabe aneinander gebunden werden. Die Struktur und die Funktionsweise dieser sich bildenden und wieder auflösenden *Agenten* ist bisher weitgehend unbekannt.

Dieses Kapitel ist wie folgt aufgebaut: Abschnitt 4.1 beschreibt die Architektur ausgehend von ihren Prinzipien, von ihrer Struktur und von ihren Bausteinen. In den nachfolgenden Abschnitten werden die Bausteine, sowie die sie zusammenhaltenden Netzwerke eingeführt. Das Kapitel endet mit einem Gedankenexperiment, das die Architektur veranschaulicht.

## 4.1 Grobstruktur – Agentenarchitektur

Mobile Systeme für die (teil-) autonome Steuerung von Fahrzeugen (PKW oder LKW) sind inzwischen in der Lage, komplexe Umgebungen (wie z. B. den Innenstadtverkehr [FGG+98]) assistiert zu beherrschen. Demnächst werden sie den Sprung aus den Forschungslabors in die Serienentwicklung vollziehen [Schu 99]. Die verwendeten Softwarearchitekturen sind nur vereinzelt agentenorientiert, statt dessen häufig speziell auf die Aufgabe hin entworfen. Dies bedeutet insbesondere, daß die Anordnung der Systemkomponenten weitgehend vorgegeben und die Beziehungen zwischen den Komponenten fest sind.

Ziel von Comros ist es, eine Softwarearchitektur für Gruppen von interagierenden Robotersystemen zu erstellen, welche sich an verschiedenste Aufgaben anpassen und dynamisch umkonfigurieren kann [BML 97], [LMB 95], [LBLM 98], [OsLe 97], [Par 99].

### 4.1.1 Prinzipien der Architektur

Die Grundsätze, die zu dem Entwurf der Architektur führten, berücksichtigen die Fähigkeit biologischer Systemen sich selbst zu organisieren:

1. Ähnlich der Zellen eines Lebewesens [ABL+ 90] sind die Funktionsmodule von ihrer Fähigkeit zur Autonomie geprägt:

   Die Fähigkeit zur Selbststeuerung hat dabei das Ziel, die Qualität der von den Modulen erbrachten Funktionen sowie die Fehlertoleranz des Gesamtsystems zu erhöhen. Die Umsetzung dieses Prinzips erfordert, die Ausführung jeder Aktion zu überwachen, sowie jede Wahrnehmung auf Plausibilität hin zu überprüfen. Falls entweder die Überwachung oder die Plausibilitätsprüfung scheitert, sind Maßnahmen zu ergreifen, die geeignet sind, die Ursache für das Scheitern zu erkennen und zu beheben.

2. Ein Grundprinzip zum Entwurf von effizienten Problemlösungen ist der Divide-and-Conquer-Ansatz. Übertragen auf die Durchführung von Aufgaben wird dieser Ansatz “Task Decomposition” genannt:

   Die Fähigkeit zur “Task Decomposition” hat das Ziel, die Anpaßbarkeit an sich ändernde Umgebungen zu erreichen. Hierbei übernehmen die autonomen Module die Zerlegung einer Aufgabe in kleinere Teilaufgaben, deren Zuordnung zu spezifischen Modulen, deren Durchführung sowie die Zusammenführung der Ergebnisse.

3. Führt man das Prinzip der Autonomie zu Ende, so bedeutet dies, daß alle Komponenten eines Systems autonom sind und folglich das System durch die Komponenten verteilt gesteuert wird (“Decentralized Control”):

   Die Grundzüge dieses Prinzip sind: das System robust gegenüber den Ausfall einer zentralen Komponente auszulegen, Engpässen während einer Spitzenbelastung vorzubeugen, sowie Entscheidungen dort zu treffen, wo aktuelles Wissen vorhanden ist.

   Die Sicherstellung der Kohärenz des Systems, also der Eigenschaft des Systems sich als eine Einheit zu verhalten, ist daraufhin die herausfordernde Aufgabe der Architektur. Hierzu sind Fragen nach der Höhe der zulässigen Unsicherheit im Wissen, sowie nach der Berücksichtigung des Informiertheitsgrades einer Komponente bei der kooperativen Entscheidungsfindung zu beantworten [LMB 95].

### 4.1.2 Struktur der Architektur

Die Architektur wird von der Vernetzung der autonomen Funktionsmodule (Elementaragenten) geprägt: Sie beruht auf zwei Klassen von Netzwerken, den Datenflußnetzwerken und den Entscheidungsnetzwerken.

Die Klasse der *Datenflußnetzwerke* wird in Anlehnung an eine Pipe-and-filter-Architektur ([SaGa 96] S. 21 f.) durch die Kommunikationsbeziehungen zwischen den arbeitsteilig zusammenarbeitenden Elementaragenten gebildet. Eine konkrete Vernetzung wird aus der Aufgabe heraus gebildet. Werden mehrere Aufgaben nebenläufig bearbeitet, so *verschmelzen* die einzelnen Datenflußnetzwerke zu einem Datenflußnetzwerk. Dabei stehen naturgemäß die Aufgaben in Konkurrenz zueinander.

Die Klasse der *Entscheidungsnetzwerke* verbindet die Entscheidungseinheiten der Elementaragenten. Entscheidungsnetzwerke werden durch ein Interaktionsverfahren beschrieben, welches die Entscheidungen zwischen den beteiligen Agenten abgleicht. Ein konkretes Entscheidungsnetzwerk besitzt ein Restriktionsnetz, welches die Abhängigkeiten zwischen den Entscheidungsfunktionen der einzelnen Agenten modelliert. Aufgabe des Restriktionsnetzes ist es, koordiniert durch das Interaktionsverfahren, die Entscheidungen zwischen den Agenten abzustimmen.

Die Architektur wird in drei Ebenen unterteilt: in die strategische, in die taktische und in die reflexive Ebene.

Die *reflexive Ebene* beinhaltet die Elementaragenten, welche die Sensorik und Aktorik ansteuern. An sie werden hohe Anforderungen hinsichtlich ihrer Reaktionszeit gestellt. Ihre Planungs- und Lernfähigkeit ist auf ein Minimum beschränkt. Die von ihnen ausgeführten Planskelette besitzen Wissen über die von ihnen benötigten und bereits verbrauchten Ressourcen. Als Element einer Pipeline ist die dem Agenten für die Durchführung seiner Funktion zur Verfügung stehende Zeit vorgegeben. Aufgabe der Entscheidungsnetzwerke dieser Ebene ist es, den Ressourcenverbrauch der Pipeline abzustimmen und die Führungsgrößen der einzelnen Planskelette hinsichtlich der Aufgabe untereinander abzustimmen. Das Kommunikationssystem der reflexiven Ebene ist

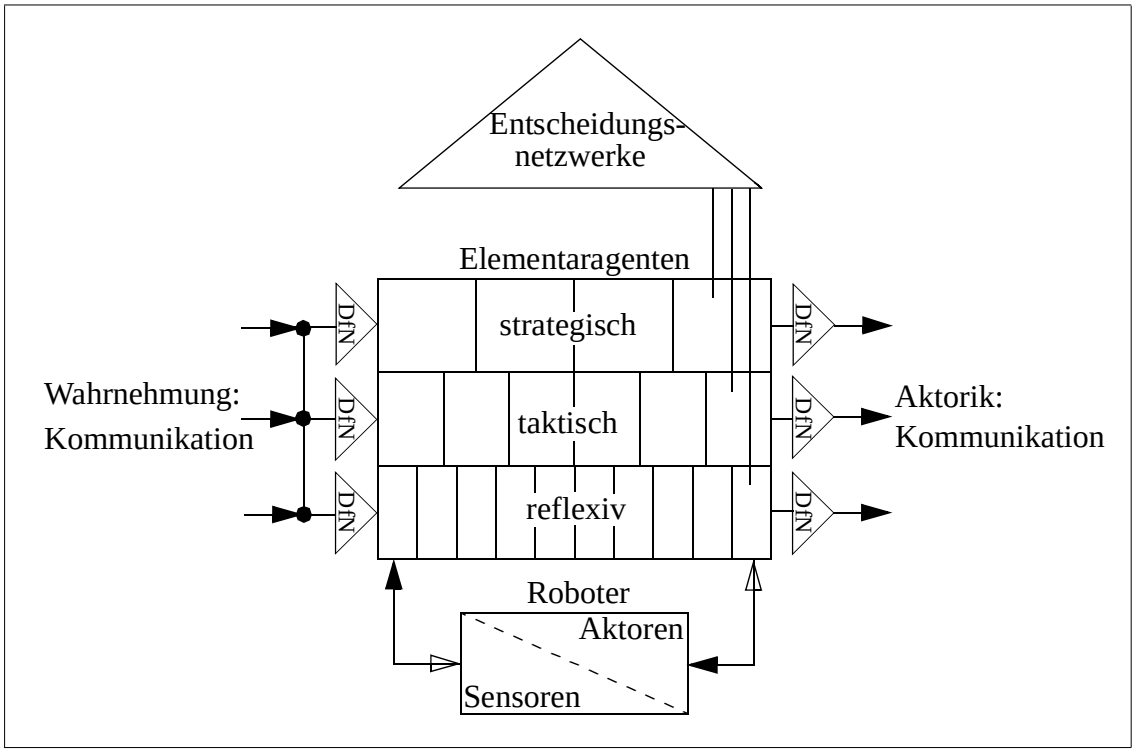


Abbildung 4.1: Grobstruktur der Agentenarchitektur in Comros

DfN ist die Abkürzung für Datenflußnetzwerke; die unterschiedlichen Pfeilspitzen zur Hardware deuten an, daß es eine Hauptrichtung und eine Rückrichtung gibt; die Verbindung zwischen den eingehenden Kommunikationskanälen symbolisiert, daß ein Datenflußnetzwerk auch mehrere Ebenen verbinden kann; die sowohl zur Wahrnehmung als auch zur Aktorik offenen Kommunikationskanäle deuten an daß ein System sich mit anderen Systemen verbinden kann.

hinsichtlich eines hohen Datendurchsatzes und auf minimale Latenzzeiten hin ausgelegt. Jeder Datenkanal bietet jedoch darüber hinaus grundsätzlich die Möglichkeit, von (Elementar-) Agenten höherer Ebenen abgehört zu werden.

Die *taktische Ebene* umfaßt all jene Elementaragenten, welche bei der Interpretation der (Umgebungs-) Situation, in der sich das Robotersystem befindet, und welche bei der Planung des auf die Situation abgestimmten Verhaltens beteiligt sind. Die taktische Ebene ist in der Lage, die Datenflußnetzwerke der reflexiven Ebene zu konfigurieren und Ressourcenzuteilung vorzugeben. Elementaragenten dieser Ebene besitzen ausgedehnte Planungs- und Lernfähigkeiten. Das Kommunikationssystem der taktischen Ebene ist darauf ausgelegt, möglichst plattform- und sprachunabhängig zu sein. Gleichzeitig läßt sich in dieser Ebene das System hinsichtlich der eingebundenen Rechnerkapazität leicht skalieren. Hierbei ist immer sicherzustellen, daß der Ausfall eines beliebigen Rechners stets durch das Restsystem beherrschbar bleibt.

Über die *strategische Ebene* wird der Roboteragent in Multiagentensysteme des Gesamtsystems eingebunden. Sie ist also Teil der *Organisationsschicht* (agent organization

layer) nach [Les 98]. Hauptbestandteil der strategischen Schicht sind operative Führungsaufgaben, wie beispielsweise die Auftragsabwicklung und -durchsetzung in einem wandlungsfähigen Unternehmen [WBW 98]. In [BMS 96] (S. 64) wurde diese Ebene hinsichtlich eines agentenorientierten Auftragsplanungssystems in drei Schichten unterteilt: in eine *Auftragsschicht*, welche die Koordination über einen gesamten Auftragsprozeß übernimmt, in eine *Aktivitätsschicht*, welche die zu Gruppen zusammengefaßten untereinander konkurrierenden Bewerber eines Produktionsschrittes koordiniert und in eine *Bewerberschicht*, welche die einzelnen Agenten enthält, welche ihre Bewerbungen planen und sich mit anderen Bewerbern zu Koalitionen zusammenschließen können.

### 4.1.3 Bausteine der Architektur

Die Systemkomponenten, in die das Robotersystem durch die Softwarearchitektur zerlegt wird, werden als autonome *Agenten* modelliert (im folgenden Elementaragenten genannt). Ein Elementaragent wird aus der Kopplung von Sensorik und zugehöriger Aktorik gebildet, über welche ein erweiterter Regelkreis gebildet wird (Abschnitt 4.2). Ein Agent aggregiert seine Wahrnehmungen von der Umwelt und bildet ein virtuelles *Sensor/Aktor-Paar* (der virtuelle Sensor ist gegeben durch die vom Agenten angebotene Information, der virtuelle Aktor ist gegeben durch die möglichen Beauftragungsmodi des Agenten und deren Parametrisierung). Der Wahrnehmung durch Sensorik ist die Wahrnehmung durch Kommunikation mit anderen Agenten gleichgestellt.

Durch die Kommunikationsbeziehungen werden die Agenten in einem *Datenflußnetzwerk* angeordnet, dessen Struktur durch die zu erledigende Aufgabe bestimmt wird. Ein Datenflußnetzwerk kann entweder fortlaufend in Betrieb sein, oder ereignisgesteuert angestoßen werden. Welche Elementaragenten hierzu berechtigt sind, wird durch entsprechende Rollen modelliert, welche in dem das Datenflußnetzwerk verwaltenden Interaktionsverfahren beschrieben sind.

Datenflußnetzwerke können über Datenkanäle Information aus anderen Datenflußnetzwerken abgreifen und in ihre eigene Problemlösung übernehmen. Diese Nutzung kann entweder “passiv” sein, womit keine Anforderungen an das unterliegende Datenflußnetzwerk gestellt wird. Oder die Nutzung ist “aktiv”, dann werden die Anforderungen über ein sie modellierendes und sie relaxierendes *Entscheidungsnetzwerk* verwaltet.

## 4.2 Elementaragenten

Die Architektur der Elementaragenten geht auf den Autonomiezyklus [Lev 89] und [Lev 92] zurück. Konkrete Realisierungen sind in [Bec 97], [Par 99] und [ROL 95]

beschrieben. Konzeptionelle Weiterentwicklungen finden sich in [LMB 95] und [LBLM 98].

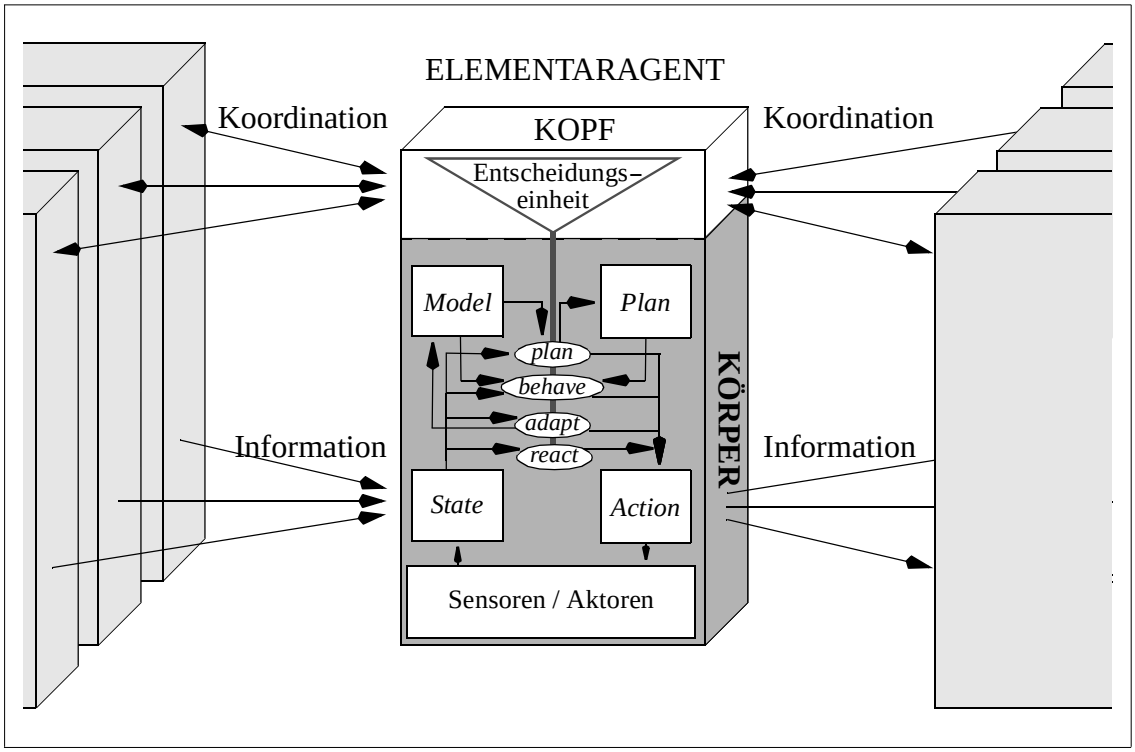


Abbildung 4.2: Grobstruktur eines Elementaragenten

Die Elementaragenten sind die autonomen Funktionsmodule des Systems. Seine Funktionalität erhält ein Elementaragent durch seine Pläne, welche die Kopplung aus Sensorik und zugehöriger Aktorik steuert. Den Körper des Elementaragenten überwacht eine Entscheidungseinheit, welche sich über Entscheidungsnetzwerke mit anderen Elementaragenten koordiniert (Abbildung 4.2).

### 4.2.1 Körper

Der Körper des Elementaragenten basiert auf einer geeigneten *Kopplung* von Sensorik und Aktorik. Als Beispiel kann man sich eine Kamera mit zugehöriger Schnittstellenkarte (sog. Frame Grabber) sowie einem mit mehreren Freiheitsgraden beweglichen Kamerakopf (z. B. Pan-Tilt-Unit) mit zugehörigen Gelenksensoren vorstellen. Ein Elementaragent definiert auf einer abstrakteren Ebene ein *virtuelles Sensor-Aktor-Paar,* welches ausschließlich von einem höheren Elementaragenten gesteuert wird (*Abschottung*), oder welches seine Sensorinformation mittels eines *Datenflußnetzwerkes* mehreren Elementaragenten bereitstellt. Ein Beispiel für eine Abschottung ist der Roboterantrieb, welcher vollständig von einem Piloten gesteuert wird. Ein Beispiel für die mehrfache

Nutzung der Information eines virtuellen Sensors ist die Hinderniserkennung: Deren Information wird sowohl von einer Szeneninterpretation als auch von einem Piloten genutzt.

Das Verhalten eines Elementaragenten wird durch die in seinem Körper ablaufenden Pläne bestimmt. Diese Pläne entsprechen Regelkreisen, welche die Sensoren und Aktoren steuern. [Rau 98] (S. 113) stellt den Autonomiezyklus seiner Realisierung einem adaptiven Regelkreis gegenüber (Abbildung 4.3):

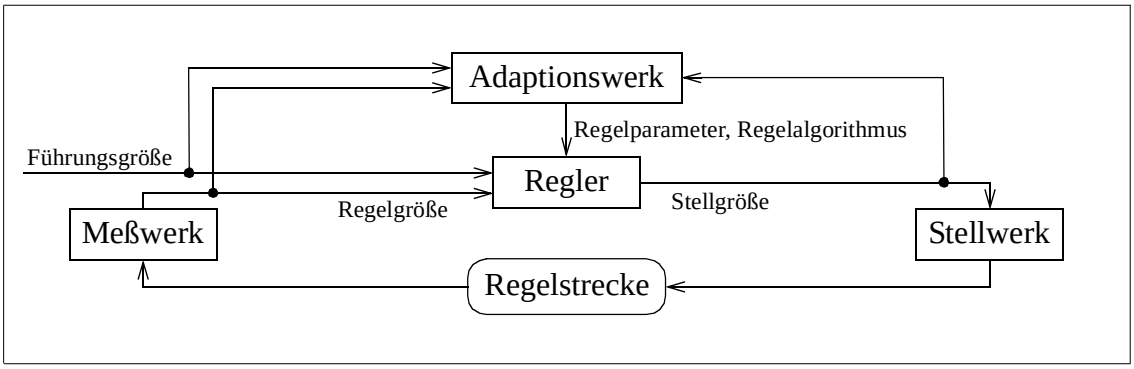


Abbildung 4.3: Blockschaltbild eines adaptive Regelkreises nach [Rau 98]

Nach [LBLM 98] ergeben sich nun folgende Entsprechungen: Der Regler[1] wird durch einen ablaufenden Plan verkörpert, welcher seinen Algorithmus und seine Parameter aus dem Adaptionswerk erhält. Dieses Werk wird durch die Komponenten *Model* bzw. *Plan* gebildet. Das Stellwerk entspricht den Aktoren, das Meßwerk entspricht den Sensoren und die Regelstrecke modelliert die Umwelt, auf dem der Regler einwirkt.

Im Körper des Elementaragenten läuft ein Plan in Form einer Programmschleife ab, welche einer oder mehreren Klassen von Zyklen zugeordnet werden kann:

1. Ein *React*-Zyklus ist die primitivste Form eines Regelkreises. Er besitzt voreingestellte Regelparameter und einen festen Regelalgorithmus. Der Regler berechnet fortlaufend die Stellgröße so, daß die Regelgröße sich der Führungsgröße angleicht.
2. Durch einen *Adapt*-Zyklus ist es dem Elementaragenten möglich zu lernen. Er übernimmt die Regel-, die Führungs- und die Stellgröße und paßt die Regelparameter im Adaptionswerk an, welche durch die Komponente *Model* verwaltet werden.
3. Der *Behave*-Zyklus ist in Lage, sich an ändernde Regelparameter anzupassen, sowie den verwendeten Regelalgorithmus zu wechseln. Hierzu wählt er entsprechende Pläne aus der Komponente *Plan* aus, und ersetzt mit ihnen seinen bisherigen Regelalgorithmus.
4. Der *Plan*-Zyklus ist der Zyklus mit den höchsten kognitiven Fähigkeiten. Er ist in der Lage neue Planskelette zu erstellen, welche den Regler über das Adaptionswerk mit neuen Regelalgorithmen versorgen.

1. [Rau 98] hatte die Entscheidungseinheit als Regler identifiziert.

### 4.2.2 Kopf

Das Verhalten eines Elementaragenten wird durch die teilautonomen Pläne bestimmt, die in seinem Körper ablaufen. Die Entscheidungseinheit als Kopf ist die Steuereinheit des Elementaragenten.

Die Entscheidungseinheit hat die Aufgabe, die im Körper ablaufenden Pläne zu überwachen. Hierzu stellt jeder Plan Information über die von ihm belegten und verbrauchten Ressourcen zur Verfügung. Die Entscheidungseinheit koordiniert die Ressourcennutzung innerhalb des Elementaragenten, kann einzelne Zyklen hemmen oder freigeben und Zuteilungen von verbrauchbaren Ressourcen (z. B. CPU-Zeit) vornehmen. Die Entscheidungseinheit erfüllt damit die Funktion eines Long-term-Scheduler ([SP 88] S. 156 f).

Darüber hinaus ist sie verantwortlich für die Koordination mit anderen Elementaragenten: Erfordert der *Behave*- oder der *Plan*-Zyklus die Koordination mit entsprechenden Zyklen aus anderen Elementaragenten, so fällt dies in den Verantwortungsbereich der Entscheidungseinheit.

Hierzu repräsentiert sie den Elementaragenten entsprechend Definition 3.3 als Menge von Diensten, welche er unterstützt, und welche entsprechend Definition 3.14 Voraussetzung für die Übernahme einer Rolle in einem Interaktionsverfahren sind.

Die Menge der Dienste, welche die Fähigkeiten eines Elementaragenten beschreiben, werden durch die Pläne definiert, die der Elementaragent besitzt. Zur Laufzeit steht eine Rolle über diese Dienste mit den Plänen in Kontakt, welche die den Diensten zugeordnete Funktionalität erbringen. Mit den Diensten ist es einer Rolle möglich, steuernd auf die Ausführung eines Planes einzuwirken.

Durch die Übernahme einer Rolle und durch die Delegierung von Entscheidungsbefugnisse an diese Rolle, ist es der Entscheidungseinheit möglich, sich mit anderen Elementaragenten entsprechend des durch das Interaktionsnetz definierten Mechanismus zu koordinieren. Anders ausgedrückt nimmt die Entscheidungskompetenz durch die übernahme einer Rolle zu. Der Elementaragent wird dadurch fähiger im Sinne der Kohärenz des Gesamtsystems richtige Entscheidungen zu treffen.

Die Entscheidungseinheit dehnt sich also über alle von ihr übernommenen Rollen aus. Sie ist damit inhärent verteilt und kann zur Laufzeit erweitert werden.

## 4.3 Datenflußnetzwerke

Das Datenflußnetzwerk bildet das Rückgrat einer Anwendung. Es beschreibt, wie Elementaragenten zur Lösung von Aufgaben verschaltet werden und wie sie interagieren.

**Definition 4.1** *Datenflußnetzwerk*

Ein Datenflußnetzwerk ist ein bipartiter Graph[1], beschrieben durch das Tupel $N_D = (S, T, F, \sigma)$ so, daß gilt

- $S$ ist eine endliche Menge von Datenpuffern
- $T$ ist eine endliche Menge von Elementaragenten
- $F \subset S \times T \cup T \times S$ ist die Flußrelation
- $\sigma$ attributiert die Datenpuffer hinsichtlich ihres Sichtbarkeitsbereichs: $\sigma$: $S \rightarrow \{$ Schicht-sichtbar, Agent-sichtbar, Team-sichtbar $\}$ .

Die *Elementaragenten* (graphisch durch Rechtecke dargestellt) kommunizieren über *Datenpuffer* (graphisch durch Kreise dargestellt), welche beliebige Datentypen aufnehmen können. *Schnittstellen zwischen Ebenen* oder *zwischen verschiedenen Roboteragenten* werden ebenfalls durch Datenpuffer realisiert. Zur Unterscheidung von gewöhnlichen Datenpuffern werden sie durch andere Symbole (Fünf- oder Dreiecke) dargestellt. Die im Abschnitt 4.2.1 angesprochene *Abschottung* eines Elementaragenten durch einen anderen wird in den Datenflußnetzwerken dadurch ausgedrückt, daß die beide Elementaragenten verbindende Datenpuffer keine Verbindung zu weiteren Elementaragenten besitzen.

Ein Datenflußnetzwerk einer Anwendung ist die Vereinigung von einfacheren Datenflußnetzwerken, die einzelne Teilaufgaben der Anwendung lösen. Verschiedene Teilaufgaben nutzen gemeinsame Autonomiezyklen, welche zwischen den Teilaufgaben datengetrieben wechseln. In Abschnitt 4.5 wird eine Aufgabe mit mehreren Teilaufgaben vorgestellt.

Abbildung 4.4 zeigt eine Folge von Datenflußnetzwerken mit steigender Komplexität der Aufgabenstellung. Netzwerk I zeigt einen Navigator (*N*), der Trajektorien an einen Piloten (*P*) weiterleitet. Dieser wandelt sie in einzelne Fahrbefehle um und schickt sie an einen Roboterantrieb (*R*). In Netzwerk II überwacht *P* zusätzlich die Umgebung mit Ultraschall (*U*), um vor Hindernissen rechtzeitig zu stoppen. Der Ultraschallagent ist in diesem Netzwerk so konfiguriert, daß er fortlaufend feuert. Sobald neue Abstandswerte vorliegen werden sie an *P* weitergeleitet. In Netzwerk III erhält *P* von einer Hinderniserkennung nun Informationen über Art und Position von Objekten innerhalb des Fahrweges, welchen er nun ausweichen kann. In Netzwerk IV greifen sowohl *P* als auch *H* auf *U* zu. Der Ultraschallzyklus ist nun so konfiguriert, daß er nur auf Anforderung (von *P* und *H*) feuert. Neu hinzugekommen ist auch, daß *H* erkannte Objekte den Hinderniserkennungen anderer Roboteragenten mitteilen kann.

1. Die Datenflußnetzwerke können als erweiterte Petri-Netze aufgefaßt werden, was detailliert in [LMB 95] vorgestellt wurde.

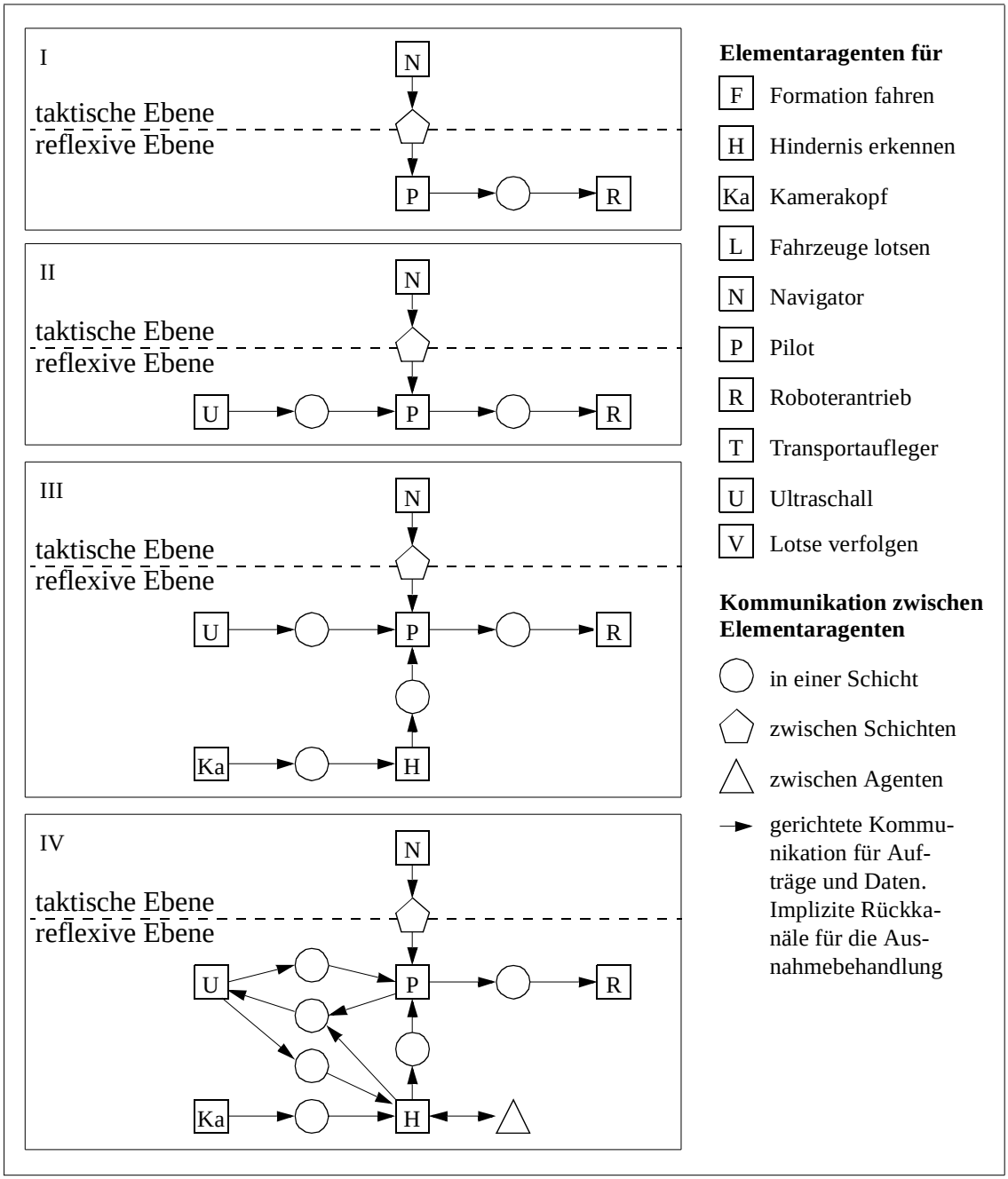


Abbildung 4.4: Beispiele von Datenflußnetzwerken

### 4.3.1 Datenpuffer

Die Attributierung der Datenpuffer bestimmt die Realisierung der vom Datenpuffer angebotenen Schnittstellen:

- Die Attributierung eines Datenpuffers als *Schicht-sichtbar* spezifiziert, daß dieser Datenpuffer keine Sichtbarkeit außerhalb der eigenen Schicht besitzt. In der Implementierung können dann spezielle Optimierungen genutzt werden, wie z. B. die Pufferverwaltung in einem gemeinsamen Speicherbereich der adjazenten Elementaragenten.
- Durch die Attributierung eines Datenpuffers als *Agent-sichtbar* wird die Fähigkeit für eine schichtübergreifende Kommunikation der mit dem Datenpuffer verbundenen Elementaragenten spezifiziert. Hierzu ist der Datenpuffer innerhalb des Agenten bekannt zu machen und der entfernte Zugriff auf seine Schnittstellen zu realisieren.
- Durch die Attributierung eines Datenpuffers als *Team-sichtbar* wird die Fähigkeit der mit dem Datenpuffer verbundenen Elementaragenten zu einer agentenübergreifenden Interaktion spezifiziert. Dies betrifft die Sichtbarkeit des Puffers im Team, sowie die Realisierung eines Rechtemodells hinsichtlich des Schreibens in einen Puffer, sowie des Lesens von Werten aus einem Puffer.

Die Attributierung der Datenpuffer kann weiterhin dazu genutzt werden, die Entscheidung hinsichtlich der Verteilung der Elementaragenten auf unterschiedliche Rechnerknoten zu unterstützen. Ein mit *Agent-sichtbar* attributierter Datenpuffer unterstützt entfernte Zugriffe und kann damit eine Schnittstelle zu anderen Rechnerknoten bilden. Berechnet man für ein Datenflußnetzwerk seine Zusammenhangskomponenten, wobei Datenpuffer mit dem Attribut *Agent-sichtbar* oder *Team-sichtbar* als Trennstellen aufgefaßt werden, so ist jede Zusammenhangskomponente ein Kandidat für die Verteilung auf einen eigenen Rechnerknoten.

Weitere Kriterien, welche die Entscheidung hinsichtlich einer Verteilung auf unterschiedliche Knoten beeinflussen, sind einerseits die benötigte Kommunikationsbandbreite, sowie andererseits die Verzögerung bei der Übertragung, welche die Durchlaufzeit durch die Pipeline erhöht.

### 4.3.2 Bindung von Elementaragenten an Datenpuffer

Ein Elementaragent kann sich auf verschiedene Arten an einen Datenpuffer binden:

- Arbeitet der Elementaragent als Senke im Gleichtakt mit dem Elementaragent, welcher die Quelle der Daten darstellt, so werden neue Daten direkt an ihn ausgeliefert. Innerhalb des Elementaragenten werden die Daten direkt an den entsprechenden Plan weitergeleitet, welcher auf das Eintreffen der Daten wartet.
- Arbeitet der Elementaragent als Senke asynchron zu der Quelle, so kann es auftreten, daß Daten von der Quelle abgeschickt werden, obwohl die vorhergehenden noch nicht bearbeitet wurden.

Soll das Eintreffen neuer Information, ältere, noch nicht zugestellte Information ersetzen, so werden die Daten im Datenpuffer gespeichert und die Senke nur über das Eintreffen neuer Daten benachrichtigt. Diese holt sich zum benötigten Zeitpunkt die aktuellen Daten aus dem Datenpuffer ab.

- Soll jedoch sichergestellt werden, daß alle Daten die Senke erreichen, so wird wie im ersten Fall verfahren, wobei der Elementaragent als Senke für eine geeignete Pufferung verantwortlich ist.

Welche Form der Interaktion zwischen zwei Elementaragenten gewählt wird, bestimmt die Senke durch die Registrierung beim entsprechenden Dienst des Datenpuffers.

### 4.3.3 Sensordaten

Das Prinzip der Autonomie erfordert es, daß jede Wahrnehmung auf Plausibilität hin überprüft werden muß. Für einen Elementaragenten, als Senke von Sensordaten, kann dieser Test fehlschlagen, wobei unter Sensordaten jegliche Art von eingehender Information verstanden werden soll. Geeignete Maßnahmen, welche zur Erkennung und Behebung der Ursache für das Scheitern führen, können häufig nur von der Quelle ergriffen werden. Hierfür ist für jeden Datenkanal ein impliziter Rückkanal für Fehlermeldungen vorgesehen, welcher solange offen gehalten wird, bis die Plausibilitätsprüfung der Senke erfolgreich oder fehlerhaft abgeschlossen ist. Das Spektrum an Fehlerinformation, welche über diesen Kanal zurückfließen kann, wird bei der Deklaration der Datentypen und der zugeordneten Schnittstellen festgelegt.

Gibt es zu einer Informationsquelle genau eine Senke, so ist die Lebensdauer des Fehlerkanals eindeutig definiert und effizient implementierbar: Der Fehlerkanal wird geschlossen, sobald die Prüfung vorgenommen wurde oder die Information von einer nachfolgenden Information überschrieben wurde.

Gibt es hingegen zu einer Informationsquelle mehrere Senken, so kann entweder auf den Fehlerkanal verzichtet werden, der Fehlerkanal nach der Übermittlung des Ergebnisses der ersten Plausibilitätsprüfung geschlossen werden, oder solange bestehen bleiben, bis alle Senken den Erfolg ihrer Plausibilitätsprüfung mitgeteilt haben. Je nach der Anzahl der Senken und der Anzahl der in den Senken zwischengespeicherten Daten, kann der Bedarf an Systemressourcen groß werden.

Die durch die Fehlermeldung angestoßenen Maßnahmen werden jedoch erst mit der Zustellung zukünftiger Sensordaten wirksam.

Grundsätzlich kann am Ende einer Pipeline ein Fehler erkannt werden, welcher nur durch Maßnahmen am Anfang der Pipeline behoben werden kann. Die impliziten Rückkanäle bieten für die Meldung solcher Fehler nicht das geeignete Konzept.

### 4.3.4 Auftragsdaten

Ein Datenflußnetzwerk kann entweder fortlaufend in Betrieb sein, oder ereignisgesteuert angestoßen werden. Auftragsdaten unterstützten ausschließlich den ereignisgesteuerten Modus: Ausgelöst durch einen Auftrag an einen entsprechenden Elementaragenten am Anfang einer Pipeline wird ein neuer Durchlauf durch die Pipeline angestoßen. In dem in Abbildung 4.4 dargestellten Datenflußnetzwerk IV wird der Ultraschall-Agenten sowohl vom Piloten als auch von der Hinderniserkennung aufgefordert erneut Daten zu liefern.

Im Falle des fortlaufenden Betriebs ist es häufig notwendig den Auftrag, welcher dem Betrieb der Pipeline zugrundeliegt, anzupassen. Dabei soll den vorne in der Pipeline angeordneten Elementaragenten dynamisch Information zur Verfügung gestellt, welche weiter hinten in der Pipeline angeordneten Elementaragenten besitzen.

Im Datenflußnetzwerk existiert hierfür kein geeignetes Konzept, da sichergestellt sein muß, daß die genannte Information, in der gesamten Pipeline konsistent ist, also alle Elementaragenten, ihr Verhalten auf die neue Situation anpassen. Hierfür führen wir im folgenden Entscheidungsnetzwerke ein, welche die genannte Information mithilfe von Restiktionstechniken modellieren.

## 4.4 Entscheidungsnetzwerke

Wie im Abschnitt 4.3 beschrieben, werden Aufgaben durch die Vernetzung von spezialisierten Elementaragenten gelöst. Jeder Elementaragent besitzt ein auf seine Funktion abgestimmtes, partielles Weltmodell, dessen Konsistenz über interne Mechanismen sichergestellt wird. Die Strukturierung einer Anwendung in einzelne miteinander vernetzte Elementaragenten gewährleistet eine funktionale Arbeitsteilung, jedoch steht dem entgegen, daß die zugehörigen Weltmodelle der Elementaragenten untereinander nicht disjunkt sind. Eine Hinderniserkennung benötigt z. B. ein Verständnis von Geschwindigkeit als einen den Anhalteweg bestimmenden Parameter, dessen Kontrolle jedoch dem Piloten unterliegt. Über die Modellierung des Datenflusses hinaus ist es also notwendig, die Schnittmenge der Weltmodelle untereinander konsistent zu halten. Um Konsistenz zu erreichen, müssen bei Konflikten Entscheidungen getroffen werden, welches partielle Weltbild bestimmend über andere Weltbilder wird, oder wie verschiedene partielle Weltbilder so kombiniert werden können, daß alle konsistent zueinander sind. Hierfür stellen wir im folgenden Netzwerke aus Entscheidungseinheiten vor.

Ein *Entscheidungsnetzwerk* vernetzt die Entscheidungseinheiten der Elementaragenten, deren Weltmodelle sich überlappen. Hierfür wird ein *Restriktionsnetz* entworfen, welches die Abhängigkeiten zwischen den in der Schnittmenge der Weltmodelle liegenden Parameter modelliert. Unter den *Restriktionsvariablen* kann man sich Parameter – wie z. B. Führungsgrößen und Regelparameter (Abbildung 4.3) – von Plänen vorstellen, welche in den Körpern der einzelnen Elementaragenten ablaufen. Eine Restriktionsvariable ist

dabei eindeutig einem Elementaragenten zugeordnet, und zwar demjenigen, in dessen Zuständigkeitsbereich (Abschnitt 3.2.1) der zugehörige Parameter hauptsächlich liegt.

Dem Entscheidungsnetzwerk liegt ein Interaktionsmechanismus zugrunde, in welchem das Restriktionsnetz konsistent gehalten wird. Dieser Mechanismus ist unabhängig von dem konkreten Restriktionsnetz und von den konkreten Teilnehmern an der Interaktion, sowie von deren Anzahl.

Die Teilnehmer bringen konkrete Verhandlungsstrategien in das Interaktionsverfahren ein, welche über standardisierte Dienstschnittstellen angesprochen werden. Diese Strategien sind in der Entscheidungseinheit angesiedelt.

Im folgenden definieren wir ein Entscheidungsnetzwerk formal:

**Definition 4.2** *Entscheidungsnetzwerk*

Ein Entscheidungsnetzwerk ist ein Tupel $N_E = (A, rs, D, N_I)$ :

- $A \subseteq A$ ist eine endliche Menge von Elementaragenten, welche sich über das Interaktionsverfahren koordinieren. Sie werden von ihren Entscheidungseinheiten repräsentiert.
- $rs \in R$ ist eine Ressource, welche die Abhängigkeiten zwischen den Elementaragenten in Form eines Restriktionsnetzes modelliert.
- $D \subseteq D$ ist eine Teilmenge aller Dienste, und beschreibt die Schnittstelle zwischen $rs \in R$ bzw. $ag \in A$ und $N_I$.
- $N_I$ ist ein Interaktionsverfahren, welches den Verhandlungsmechanismus spezifiziert, welcher Konflikte zwischen $ag \in A$ (ausgedrückt durch Inkonsistenzen in $rs \in R$ ) auflöst.

Es fällt auf, daß Entscheidungsnetzwerke aufgrund der Angabe der Teilnehmer ein geschlossenes System bilden. Dies ist damit zu begründen, daß es sich bei den Entscheidungsnetzwerken um eine sehr enge Kooperation handelt, bei der ein Eintritt von weiteren Partner die Kooperation grundsätzlich verändert. Hierfür ist von allen Partnern eine neuerliche Verpflichtung notwendig. Der Wechsel von einem Entscheidungsnetzwerk zu einem anderen erfolgt dann in einem das Entscheidungsnetzwerk verwaltenden Interaktionsverfahren.

### 4.4.1 Restriktionsnetz

Das Restriktionsnetz, welches den Kern eines Entscheidungsnetzwerks bildet, wird für jede Teilaufgabe entworfen. Es wird eine minimale Menge von Entscheidungseinheiten bestimmt, welche sich durch ihre Restriktionsvariablen gegenseitig beeinflussen. Meist besteht im Datenflußnetzwerk dann eine direkte Nachbarschaft zwischen den beteiligten

Elementaragenten. Abbildung 4.11 im Abschnitt 4.5.2 stellt ein Beispiel eines Restriktionsnetzes dar.

Nach [Win 93] (S. 41) lassen sich Randbedingungen in *imperative* (harte) und *admissive* (Präferenzen modellierende, weiche) *Restriktionen* unterteilen. Imperative Restriktionen modellieren physikalische und physische Abhängigkeiten, welche Konsistenzbedingungen zwischen verschiedenen Weltmodellen zwischen den Elementaragenten darstellen und nicht relaxiert werden können.

Ein Elementaragent besitzt zur Wahrnehmung seiner Autonomie eigene Bewertungsfunktionen, welche sich durch qualitative und quantitative, interne Restriktionen beschreiben lassen. *Qualitative Restriktionen* modellieren Abhängigkeiten zwischen Restriktionsvariablen und weiteren Parametern des Weltmodells, *quantitative Restriktionen* schränken den Wertebereich der Restriktionsvariablen ein. Diese internen Restriktionen sind abhängig von der Planmenge, welche ein Elementaragent entsprechend seiner Funktion bereithält. Die Relaxierung von internen Restriktionen wirkt sich auf die Planmenge dahingehend aus, daß spezialisierte Pläne, welche nur in bestimmten Parameterbereichen betrieben werden können durch weniger differenzierte Pläne verdrängt werden.

Durch die Bekanntgabe der Bewertungsfunktionen innerhalb eines Entscheidungsnetzwerkes ist es möglich, einen Abstimmungsmechanismus zu entwerfen, welcher ausschließlich Lösungen wählt, die Pareto-optimal[1] sind. [ArBö 83] charakterisiert Verfahren zur Durchführung von Gruppenentscheidungen nach Neutralität, Anonymität und Effizienz. Ein Verfahren ist dabei dann *neutral*, wenn das Ergebnis der Gruppenentscheidungen nicht von der Reihenfolge der Alternativen, die zur Entscheidung anstehen abhängt. Ein Verfahren heißt *anonym* oder symmetrisch, wenn das Ergebnis der Gruppenentscheidungen nicht von den “Personen” abhängt, welche die Entscheidung treffen. Hierzu reicht es aus, wenn das Komitee, welches die Entscheidung trifft, durch Los bestimmt wird. Beispielsweise wird bei der “Random-Dictator-Procedure” ein Teilnehmer zufällig gewählt, welcher dann für alle anderen die Entscheidung trifft. Dieses Verfahren ist neutral und anonym. Eine Entscheidung wird von den Teilnehmern dann nicht akzeptiert, wenn es eine Alternative gibt, welche bei einem höheren Nutzen für einige Teilnehmer keinen geringeren Nutzen bei den anderen Teilnehmern bewirken würde. Verfahren die solche Entscheidungen (und damit eine nachfolgende Verhandlung) verhindern heißen *effizient* oder Pareto-optimal. [ArBö 83] diskutiert einige Verfahren, die davon ausgehen, daß die Teilnehmer nicht bereit sind ihre Bewertungsfunktionen offenzulegen. Im Gegensatz zu diesen Verfahren werden in Entscheidungsnetzwerken diese bekannt gegeben. Hierdurch kann das Verfahren so modelliert werden, daß Pareto-Optimalität zugesichert wird.

Andererseits kann das Restriktionsnetz unter Berücksichtigung der modellierten Präferenzen keine Lösung enthalten, wohingegen das Modell ohne Präferenzen lösbar ist. Durch gezieltes, schrittweises relaxieren weicher Restriktionen kann das Restriktionsnetz soweit vereinfacht werden, daß es lösbar wird. Die Auswahl der zu relaxierenden

1. Ein Ergebnis einer Verhandlung heißt Pareto-optimal, wenn es keine Alternative gibt, dessen Nutzen für mindestens einen Teilnehmer besser, für die anderen jedoch nicht schlecht ist ([FuTi 91]).

Restriktionen wird dabei von einer Restriktionshierarchie unterstützt, entsprechend der die Restriktionen zur Relaxierung vorgeschlagen werden.

### 4.4.2 Entscheidungsstrategien

Verfolgt ein Roboteragent gleichzeitig mehrere Teilaufgaben, so können sich Restriktionsnetze überlappen. In der Schnittmenge zweier Restriktionsnetze befindet sich dann eine Untermenge der Restriktionsvariablen, welche ein oder mehrere Elementaragenten in das Entscheidungsnetzwerk einbringen. Aufgabe der zugehörigen Entscheidungseinheiten ist es sicherzustellen, daß die Werte ihrer Variablen in allen Restriktionsnetzen konsistent sind. Hierzu kann eine Entscheidungseinheit verschiedene Strategien verfolgen:

- Sie kann versuchen, diese Parameter nicht zu verändern und stattdessen in den Verhandlungen solche Lösungen des Restriktionsnetzes vorzuschlagen, die diese Parameter konstant halten.
- Sie kann versuchen, Werte für diese Parameter zu finden, welche in angrenzenden Entscheidungsnetzwerken keine Verhandlung erfordern und dort von ihr lokal kompensiert werden können.
- Sie kann den Entscheidungsspielraum der angrenzenden Entscheidungsnetzwerke berücksichtigen und versuchen, diesen zwischen den Restriktionsnetzen verschiedener Entscheidungsnetzwerke auszugleichen.
- Sie kann versuchen, den Bedarf an kurzfristigem Entscheidungsspielraum zu berücksichtigen, um somit ein Fortpflanzen von Parameterfestlegungen über eine Vielzahl von Entscheidungsnetzwerken zu verhindern.

Die ersten beiden Strategien sind *Deeskalationsstrategien*, welche das Fortpflanzen über Entscheidungsnetzgrenzen vermeiden. Die letzten beiden Strategien sind *Eskalationsstrategien*, welche gezielt die Verhandlung in einem angrenzenden Entscheidungsnetzwerk anstoßen, um die Wahrscheinlichkeit von zukünftigen Verhandlungen zu reduzieren.

### 4.4.3 Interaktionsverfahren der Entscheidungsnetzwerke

Das Interaktionsverfahren der Entscheidungsnetzwerke, welches nachfolgend beschrieben wird, ist selbst Bestandteil eines übergeordneten, den Einsatz des Entscheidungsnetzwerkes koordinierenden Verfahrens. Der zugehörige Koordinationsmechanismus des übergeordneten Interaktionsverfahrens besteht im Wesentlichen aus zwei Phasen: Einer *Simulationsphase*, in welcher das Zusammenspiel der Restriktionsvariablen getestet wird

und aus einer *Betriebsphase*, in der die auf das Restriktionsnetz wirkenden Werteveränderungen durch die dynamische Situation, in der sich das Robotersystems befindet, bestimmt werden. In der *Simulationsphase* gehen die aktuellen, lokalen Modelle der einzelnen Elementaragenten bereits ein, welche aufgrund von Lernvorgängen gegenüber den zum Entwurfzeitpunkt des Restriktionsnetzes bestehenden Modellen verändert sein können.

Im folgenden wird das Interaktionsverfahren des in der *Betriebsphase* verwendeten Subinteraktionsverfahrens beschrieben. Es ist ein symmetrisches Verfahren, in welchem kein Teilnehmer eine herausgehobene Rolle hat. (In [BML 97] wurde u. a. ein Interaktionsverfahrens angegeben, welches eine ausgezeichnete Rolle zur Lösung des Restriktionsnetzes besitzt.)

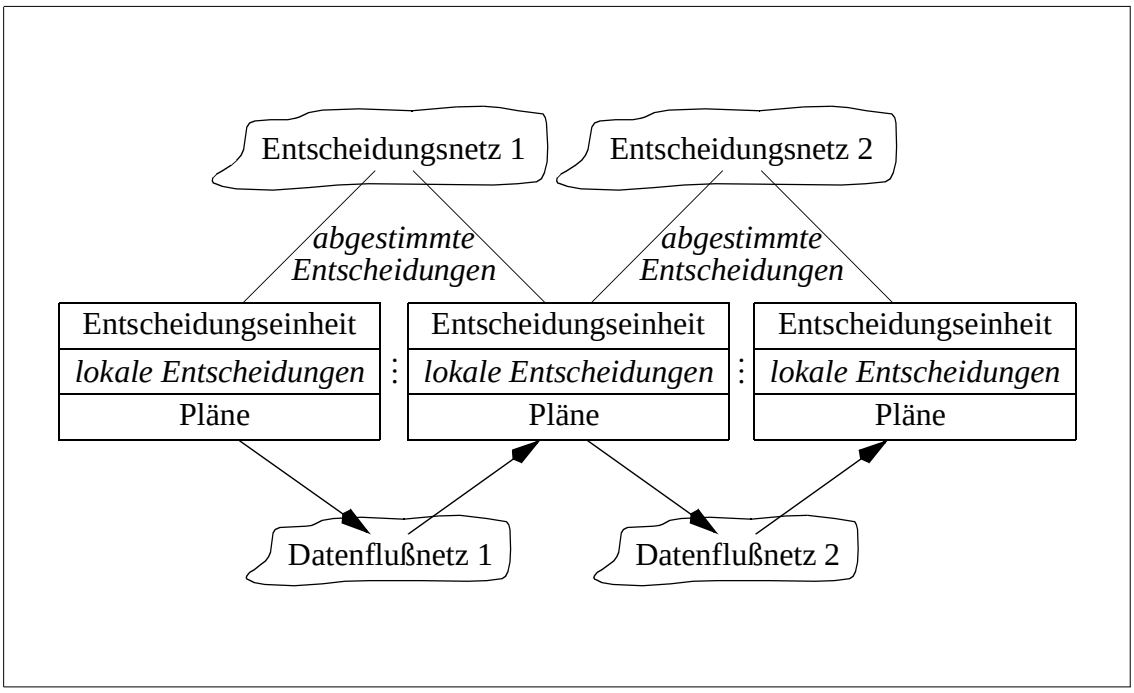


Abbildung 4.5: Zusammenwirken der Elementaragenten bei der Entscheidungsfindung

Bevor wir das Interaktionsverfahren im Detail beschreiben, soll noch kurz die Idee des Zusammenwirkens der Elementaragenten mit dem Entscheidungsnetzwerk beschrieben werden (Abbildung 4.5): Die Elementaragenten sind über das Datenflußnetzwerk miteinander verbunden und erledigen arbeitsteilig eine Aufgabe. Unter Nutzung ihrer Autonomie treffen sie *lokale Entscheidungen*, die den Zustand des Gesamtsystems beeinflussen (z. B. ein Abbremsen, welches durch den Pilot veranlaßt wird). Das System ist also in gewissen Grenzen fähig, sich autonom zu verhalten. Die Entscheidungsnetzwerke setzen auf den Elementaragenten auf und befähigen sie *abgestimmte Entscheidungen* zu treffen und damit Kohärenz zu erreichen. Während des Abstimmungsprozesses im Entscheidungsnetzwerk ist das System weiterhin in Betrieb und wird durch die Autonomie der einzelnen Elementaragenten gesteuert. Ist die Abstimmung erfolgreich, so werden die

dort *abgestimmten Entscheidungen* in Form von Wertebereichen der einzelnen Restriktionsvariablen an die Pläne der Elementaragenten übergeben, welche daraufhin ihr autonomes Verhalten an die Vorgaben anpassen und ihre *lokalen Entscheidungen* forthin innerhalb dieser Wertebereiche treffen. Reicht der hiermit festgelegte *Entscheidungsspielraum* für einen Elementaragenten nicht mehr aus, so veranlaßt er über das Entscheidungsnetzwerk eine erneute Abstimmung. Dies kann dadurch erreicht werden, daß der Entscheidungsspielraum seiner eigenen Restriktionsvariablen verschoben wird, ohne Restriktionsvariablen anderer Elementaragenten zu verändern (*lokale Lösung*) oder falls dies nicht erfolgreich ist, muß die Konsistenz des Entscheidungsnetzwerkes durch Veränderung der Restriktionsvariablen mehrerer oder aller Elementaragenten erreicht werden (*globale Lösung*).

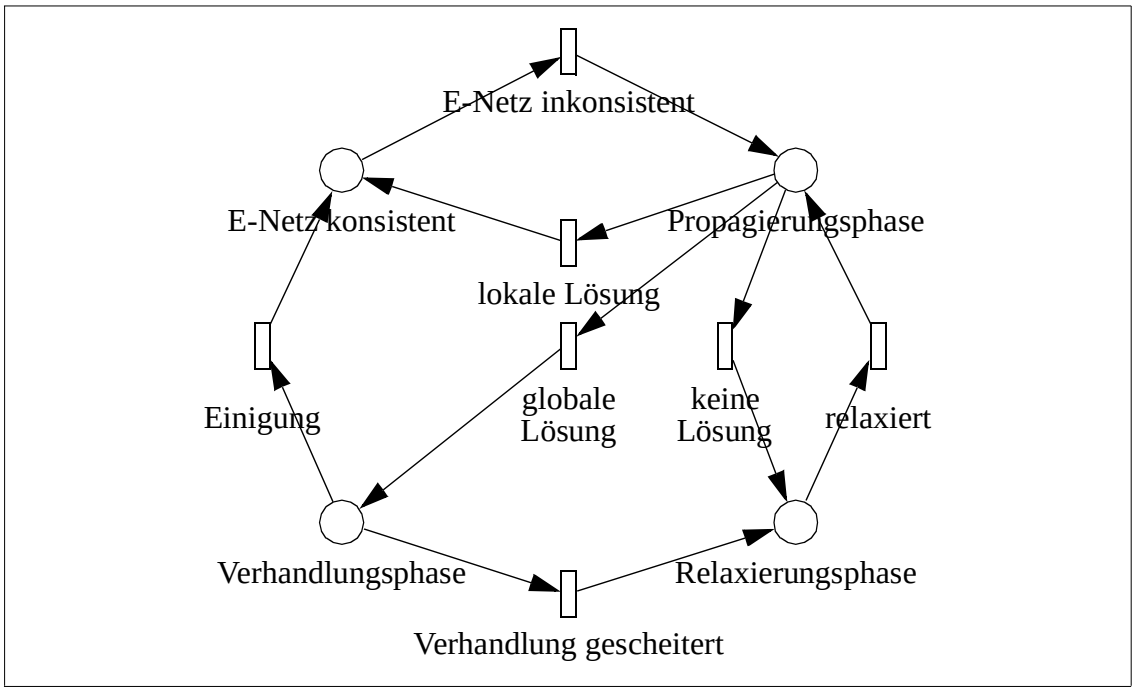


Abbildung 4.6: Interaktionsverfahren der Entscheidungsnetzwerke

In Abbildung 4.6 ist das Entscheidungsnetzwerk[1] dargestellt. Alle Teilnehmer und das Restriktionsnetz befinden sich anfänglich in der Phase "*E-Netz konsistent*". Die Restriktionsvariablen sind mit den Werten belegt, welche in der *Simulationsphase* des übergeordneten Interaktionsverfahrens als Ausgangswerte berechnet wurden.

1. Die Netzstruktur, welche die Menge der möglichen Anfangsmarkierungen durch asynchrone Ein- und Ausgänge modelliert, ist der Übersichtlichkeit halber weggelassen.

1. Phase “*E-Netz konsistent*”:

   Durch Eintritt in diese Phase werden die in dem Restriktionsnetz vorliegenden, abgestimmten Entscheidungsspielräume, welche in einer vorgelagerten Phase an die Elementaragenten übergeben wurden, gültig. In dieser Phase verweilen nun die Teilnehmer des Entscheidungsnetzwerkes, bis der Entscheidungsspielraum für *einen* Elementaragenten nicht mehr ausreicht und eine erneute Abstimmung des Restriktionsnetzes notwendig wird. Dieser Teilnehmer trägt seinen neuen Entscheidungsspielraum – durch Änderung der, seine Präferenzen modellierenden quantitativen Restriktionen – in das Restriktionsnetz ein und verursacht damit die Inkonsistenz des Restriktionsnetzes. Daraufhin schaltet die Transition “*E-Netz inkonsistent*” alle Marken in die “*Propagierungsphase*”.

2. Phase “*Propagierungsphase*”:

   Beim Wechseln in die Propagierungsphase wird dem die Inkonsistenz verursachenden Teilnehmer die Rolle “*Problemlöser*” zugeteilt, wohingegen die anderen Teilnehmer jeweils eine passive Rolle zugewiesen bekommen. Der *Problemlöser* propagiert die geänderten Restriktionen und ermittelt so, ob das Entscheidungsnetzwerk noch Lösungen besitzt. Hat das Restriktionsnetz keine Lösung mehr, so schaltet die Transition “*keine Lösung*” alle Marken in die *Relaxierungsphase*. Besitzt das Restriktionsnetz noch Lösungen, so berechnet der Problemlöser eine neue Lösung. Dabei wird versucht eine Lösung zu berechnen, welche ausschließlich die Restriktionsvariablen des Problemlösers verändern. Wird eine solche Lösung gefunden, so schaltet die Transition “*lokale Lösung*” alle Marken in die Phase *E-Netz konsistent*. Dabei werden die betroffenen Restriktionsvariablen an den Elementaragenten übergeben. Wird keine lokale Lösung gefunden, so schaltet die Transition “*globale Lösung*” alle Marken in die *Verhandlungsphase*.

3. Phase “*Verhandlungsphase*”:

   In der Verhandlungsphase wird allen Teilnehmern die Rolle “*kooperativer Problemlöser*” zugewiesen. Ihnen wird durch Schalten der Transition die Lösung der Propagierungsphase vorgeschlagen, über welche sie nun abstimmen. Dabei wählt jeder Teilnehmer entsprechend seiner Zugehörigkeit zu weiteren Entscheidungsnetzwerken eine Strategie aus Abschnitt 4.4.2.

   Daraufhin generieren die Teilnehmer Lösungen und stimmen über sie ab[1]. Begrenzt man die Zeit für die Dauer der Abstimmung und bestimmt den letzten Vorschlag als Ergebnis der Verhandlung, so wird im Allgemeinen nicht eine Pareto-optimale Lösung gefunden, jedoch kann die Reaktionsfähigkeit des Entscheidungsnetzwerkes garantiert werden.

1. [ArBö 83] (S. 1396) stellt das Abstimmungsverfahren “Voting by Successive Proposal and Veto” vor, von dem er beweist, daß das Verfahren effizient ist. Für Teilnehmerzahlen bis drei wird ein einfacheres Verfahren untersucht (“Voting by Repeated Veto”), welches neutral, anonym und effizient ist. Dieses Verfahren basiert auf dem Streichen der jeweils schlechtesten Alternative. Grundsätzlich setzen die Verfahren voraus, daß jeder Teilnehmer alle Alternativen kennt und bewertet hat.

Am Ende der Abstimmungsphase werden die Restriktionsvariablen entsprechend der gefundenen Lösung an die Elementaragenten übergeben und durch Schalten der Transition "*Einigung*" aktiviert. Die Marken werden in die Phase "*E-Netz konsistent*" geschaltet.

Die Verhandlung kann scheitern, wenn alle Lösungen des Restriktionsnetzes Einschränkungen in dem Handlungsspielraum eines Elementaragenten voraussetzen würden, der jedoch aufgrund seiner Einbindung in andere Entscheidungsnetzwerke und aufgrund seiner gewählten Strategie diese nicht hinnehmen will oder kann. Dann werden alle Marken durch die Transition "*Verhandlung gescheitert*" in die *Relaxierungsphase* geschaltet.

4. Phase "*Relaxierungsphase*":

   Besitzt das Restriktionsnetz keine Lösung, oder konnten sich die Teilnehmer in der Verhandlungsphase auf keine Lösungen einigen, so muß der Lösungsraum durch Relaxieren von Restriktionen erweitert werden. Das Restriktionsnetz enthält die Präferenzen der Elementaragenten in Form einer Hierarchie aus weichen Restriktionen, wobei jede Hierarchiestufe gleich priorisierte Präferenzen enthält. Die Anzahl der Stufen, sowie die Zuordnung von Präferenzen zu einer Stufe werden durch den Entwickler vorgenommen. In der Relaxierungsphase werden nun aus der obersten Hierarchiestufe, welche noch Präferenzen enthält, eine Restriktion auf Vorschlag, durch Mehrheitsentscheid oder wenn keine Einigung erfolgt durch Los bestimmt, welche als nächstes relaxiert wird. Dem Elementaragent, dem die Präferenz zugeordnet ist, wird beim Schalten der Transition "*relaxiert*" in die *Propagierungsphase* die Rolle *Problemlöser* zugeordnet. Allen anderen Teilnehmern wird eine passive Rolle zugeordnet.

   Falls keine weichen Restriktionen relaxiert werden können, so scheitert in dieser Phase das Interaktionsverfahren, was im aufrufenden Interaktionsverfahren behandelt wird, und dort z. B. zum Abbruch der Aufgabe führen kann.

## 4.5 Lotsenbasierter, kooperativer Transport

Im folgenden wird anhand eines umfangreicheren Gedankenexperiments demonstriert, wie Entscheidungsnetzwerke mit Elementaragenten zusammenarbeiten, welche über Datenflußnetzwerke aufgabenbezogen interagieren.

In einem Produktionsunternehmen soll der Transport von sperrigen Material wie z. B. von Trägern zukünftig von fahrerlosen Transportsystemen übernommen werden. Hierzu wurden zwei Roboterfahrzeuge mit einem Transportaufleger ausgerüstet und ein Fahrzeug als Lotse vorgesehen. Dieses fährt dem Transportkonvoi voraus und überprüft, ob die Korridore für den nachfolgenden Konvoi passierbar sind (Abbildung 4.7).

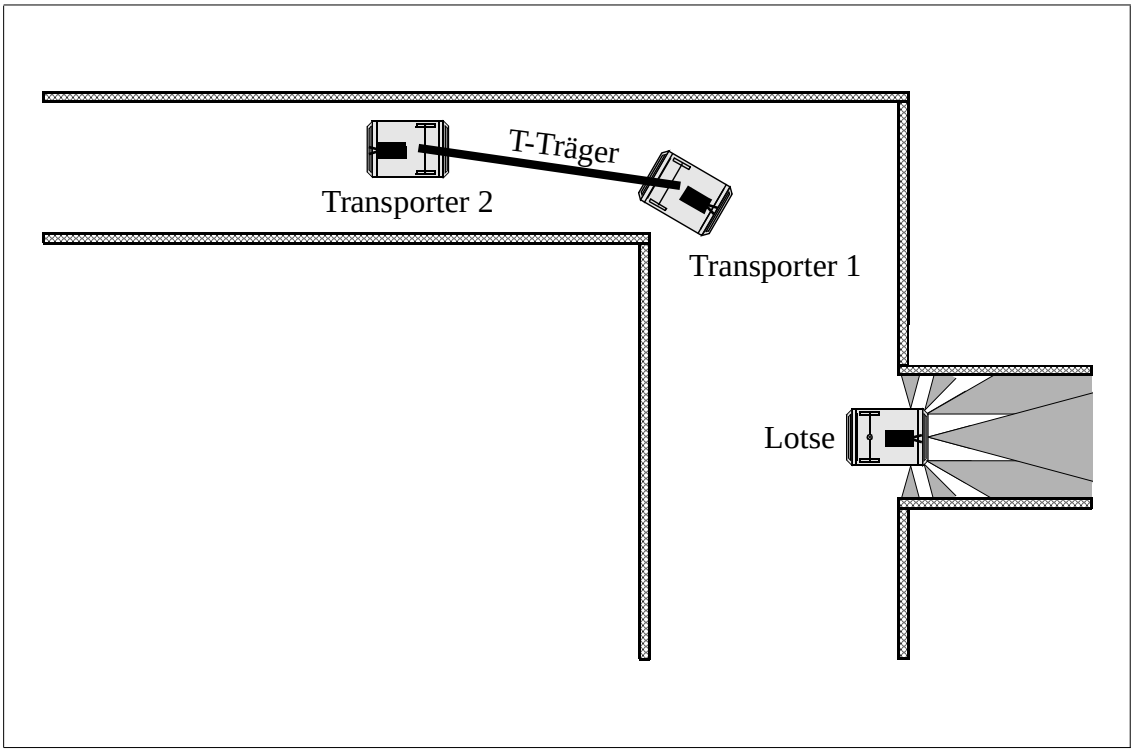


Abbildung 4.7: Szenario lotsenbasierter, kooperativer Transport

Für den Lotsen, Transporter 1 und Transporter 2 lassen sich folgende Teilaufgaben identifizieren, die nebenläufig durchgeführt werden:

- Untersuchung des Fahrwegs auf statische Hindernisse: Diese Teilaufgabe gehört zum Grundrepertoire eines autonomen Fahrzeugs. Sie wird um die Anforderung erweitert, klassifizierte Objekte und deren Position anderen Fahrzeugen mitzuteilen. Als Sensor wird ein in einer Achse schwenkbarer Kamerakopf eingesetzt.

- Manövrieren in engen Durchfahrten: Die Fahrzeuge sollen in der Lage sein, den Raum in abknickenden Korridoren vollständig auszunutzen. Hierfür wird ein Ultraschallgürtel von 24 Sensoren je Fahrzeug eingesetzt, welcher ein vollständiges Abtasten der Fahrzeugumgebung ermöglicht.

- Beförderung des Trägers durch Transporter 1 und Transporter 2: Dieser liegt auf je einem Transportaufleger auf, dessen eingebaute Sensoren die rotatorische und translatorische Relativbewegung der beiden Fahrzeuge innerhalb gewisser Grenzen messen und ausgleichen kann.

- Lotsen: Der Lotse gibt in Zeitabständen eine Route frei, die für die Transportfahrzeuge passierbar ist. Die Route wurde den Fahrzeugen vorab von ihren Navigatoren bekanntgegeben. Die Strecke muß bis zu einem vorgegebenen Termin zurückgelegt werden. Sobald abzusehen ist, daß der Termin nicht gehalten werden kann, wird dies den Navigatoren durch eine Fehlermeldung mitgeteilt.

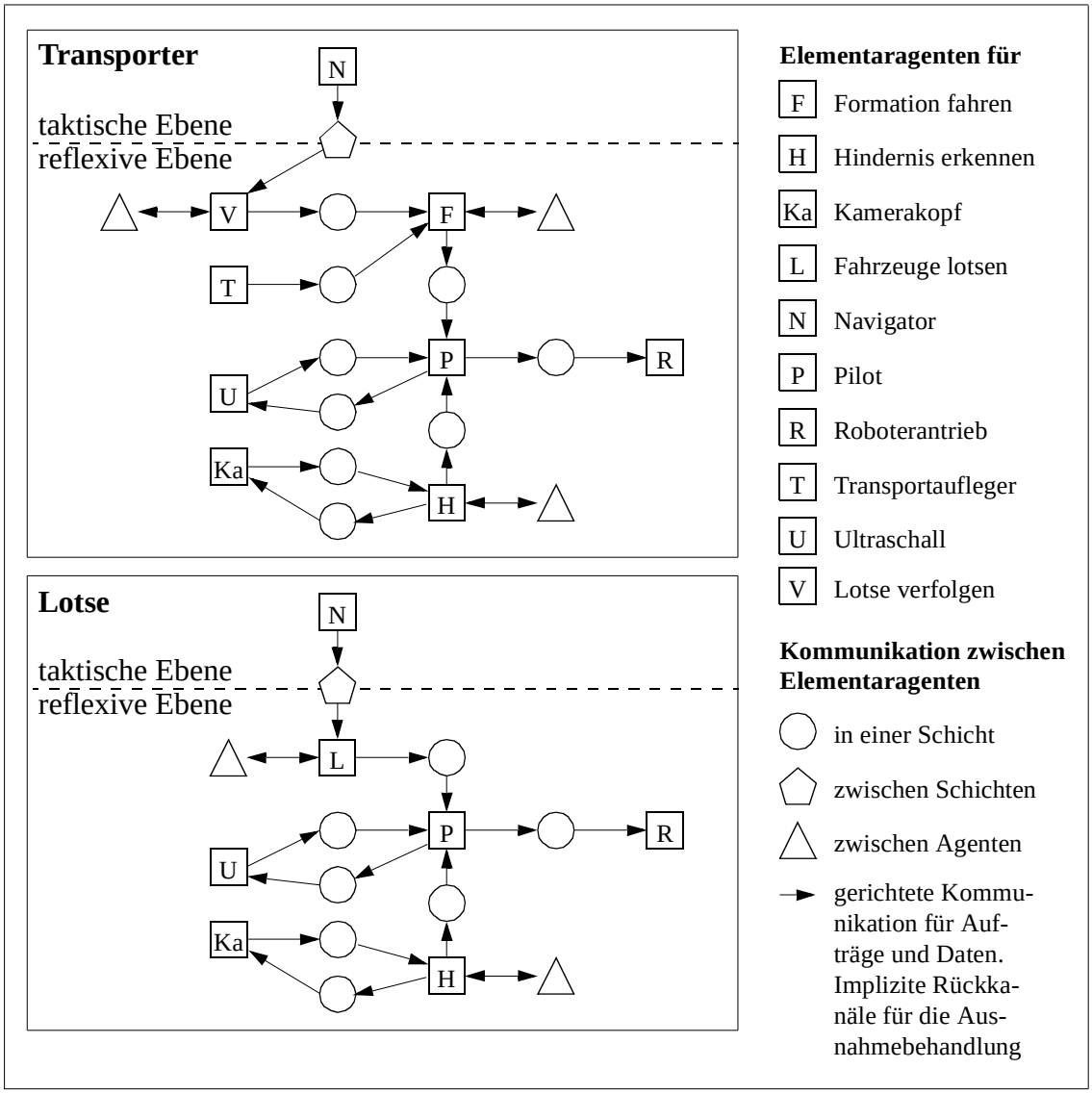


Abbildung 4.8: Datenflußnetzwerke beim lotsenbasierten, kooperativen Transport

### 4.5.1 Datenflußnetzwerke

Der Entwurf für die konkrete Agentenarchitektur der Fahrzeuge sieht folgendermaßen aus: Alle Fahrzeuge sind jeweils mit einem beweglichen Kamerakopf – gesteuert durch den Elementaragenten *Ka* – und Ultraschallsensoren – gesteuert durch den Elementaragenten *U* – ausgerüstet. Die Hinderniserkennung (*H*) jedes Fahrzeuges wertet Bilder der Kamera aus und stellt dem Piloten (*P*) eine Hinderniskarte zur Verfügung. Die zu fahrende Trajektorie wird von dem Lotsenfahrzeug grob vorgegeben (durch *L*). Die Transportfahrzeuge fahren diese ab (durch *V*), wobei die Elementaragenten

„Formationsfahren“ (*F*) den momentanen rotatorischen und translatorischen Versatz des zu transportierenden Trägers berücksichtigen. Der Pilot (*P*) erhält die korrigierte Trajektorie von *F* und kommandiert die Fahrbefehle an den Roboterantrieb, welcher durch den Elementaragenten *R* gesteuert wird.

Die Datenflußnetzwerke des Lotsen und der Transportfahrzeuge sind in Abbildung 4.8, eine Übersicht über die Entscheidungsnetzwerke in Abbildung 4.9 dargestellt.

Die Hinderniserkennung *H* verwaltet eine Karte, in der sie fortlaufend erkannte Hindernisse einträgt. Sie überwacht den Fahrbereich, der vom Pilot *P* spezifiziert wird und meldet erkannte Hindernisse an ihn weiter. Der Fahrbereich wird von *P* durch zwei Radien (nach links bzw. nach rechts) angegeben, innerhalb dieser Grenzen generiert *P* Fahr- und Ausweichbewegungen. Würde *P* eine engere Kurve fahren, so garantiert *H* keine rechtzeitige Hindernismeldung mehr. Die Restriktionen zwischen dem Fahrbereich, der gewählten Geschwindigkeit und dem von der Kamera zu überwachenden Bereich werden über ein *Ka-H-P*-Restriktionsnetz (Abbildung 4.11) abgestimmt.

Der Pilot *P* nutzt den Ultraschall *U* für die Kontrolle des Seitenabstandes während des Rangierens.

Das Formationsfahren wird von den Elementaragenten *F* (Formation fahren) und *T* (Transportaufleger) erbracht. Hierbei plant *F* ausgehend von der Sensorinformation von *T* in kurzen Abständen Trajektorien für *P.* Die für *P* zugängliche Umweltinformation in Form von zulässigen Fahrbereichen bringt *P* in ein *F-P*-Entscheidungsnetzwerk ein. Die beiden *F* der Transportfahrzeuge tauschen die Trajektorien untereinander aus, wodurch ein schneller Informationsfluß ermöglicht wird.

Das gelotste Fahren wird in den Transportfahrzeugen vom Elementaragent *V* (Lotse verfolgen) koordiniert. *V* erhält Trajektorien vom Autonomiezyklus *L* des lotsenden Fahrzeugs und reicht sie an *F* weiter. Notwendige Abstimmungen werden über ein *F-V-L*-Entscheidungsnetzwerk vorgenommen.

Den einzelnen Teilaufgaben werden Datenflußnetzwerke zugeordnet, welche jeweils durch ein Entscheidungsnetzwerk koordiniert wird (Abbildung 4.9):

- *Ka*, *H* und *P* hängen hinsichtlich der Teilaufgabe Hindernisvermeidung eng voneinander ab. Z. B. wird der Anhalteweg von der Geschwindigkeit, der maximalen Verzögerung und der Reaktionszeit bestimmt. In die Reaktionszeit geht jedoch wesentlich die zu verarbeitende Bildgröße und damit die Auflösung ein. Je schneller das Fahrzeug ist, desto weiter muß die Hinderniserkennung die Umgebung erkundigen, desto höher sollte die Auflösung sein. Dies verlängert die Reaktionszeit und somit den Anhalteweg. Hier gilt es den optimalen Betriebspunkt zu finden, der sich abhängig von der Umgebung dynamisch verschieben kann.

- Die Entscheidungseinheiten der Elementaragenten *P* und *F* der beiden Fahrzeuge sind verantwortlich für die Regelung des über den Träger gekoppelten Transportsystems. Den passiven Verfahrweg der Transportaufleger nutzen die Piloten, um Hindernissen auszuweichen oder zu rangieren. Die beiden Fahrzeuge unterteilen den Verfahrweg in drei Bereiche, den Bereich, welcher der Autonomie jedes Fahr-

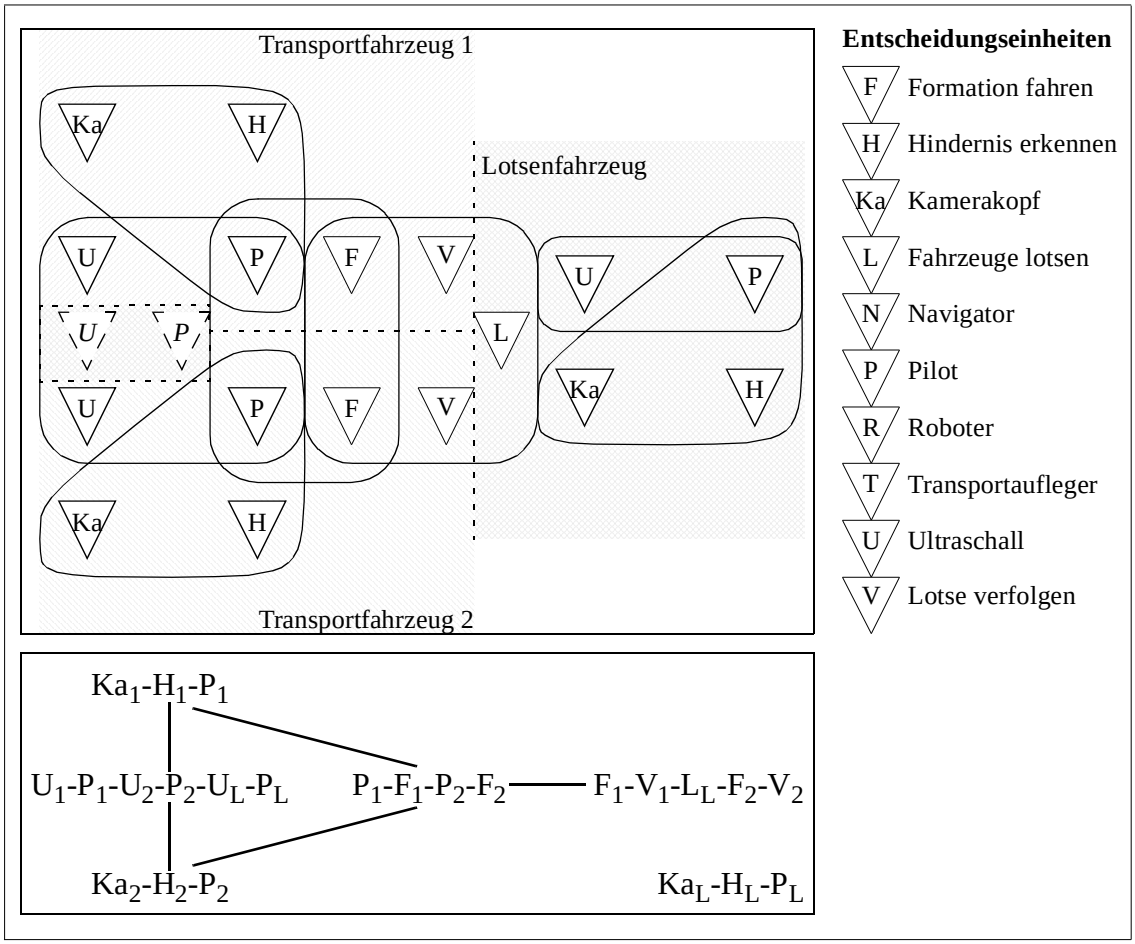


Abbildung 4.9: Entscheidungsnetzwerke und ihre Überlappungen

In der oberen Darstellung sind die Entscheidungseinheiten der Elementaragenten auf den einzelnen Fahrzeugen so angeordnet, daß die Entscheidungsnetzwerke dargestellt durch geschlossene Kurven, die an ihnen beteiligten Entscheidungseinheiten beinhalten. In der unteren Darstellung sind die Entscheidungsnetzwerke benannt durch die an ihnen beteiligten Entscheidungseinheiten. Zwei Entscheidungsnetzwerke sind durch eine Kante verbunden, wenn mindestens eine Entscheidungseinheit Teil beider Netzwerke ist.

zeuges unterliegt, den gemeinsamen Bereich, den die Fahrzeuge in Abstimmung gemeinsam nutzen, und den Bereich, der nicht verletzt werden darf und eine sofortige Gegenreaktion beider Fahrzeuge erfordert.

- Die Entscheidungseinheiten der Elementaragenten *F* und *V* der Transportfahrzeuge und *L* des Lotsen sprechen die zu fahrende Trajektorie ab. Dabei bringt *L* Restriktionen der Mission (z. B. Termine, welche für einzelne Streckenabschnitte von *L* vorgegeben werden), *V* die statischen Restriktionen des Transportsystems und *F* die operative Umsetzung der von *L* vorgegebenen Restriktionen ein.

- Die Entscheidungseinheiten der Elementaragenten *U* und *P* der drei Fahrzeuge koordinieren die Zündreihenfolge der Ultraschallsensoren, um ein minimales Übersprechen zu erreichen. Die *P* nutzen die Ultraschallsensoren zum rangieren und sind hierfür als Auftraggeber an der Koordination beteiligt.

## 4.5.2 *Ka-H-P*-Entscheidungsnetzwerk

Die Abhängigkeiten zwischen Kamerakopf, Hinderniserkennung und Piloten werden durch den Algorithmus festgelegt, mit dem der Elementaragent *H* den zu überwachenden Bereich bestimmt. Dieser Bereich wird durch je einen Blickwinkel (nach links bzw. nach rechts) sowie durch einen Mindestabstand (Anhalteweg) und einem maximalen Abstand eingegrenzt. Unterhalb des Mindestabstandes ist ein rechtzeitiges Erkennen eines Hindernisses unmöglich, so daß durch andere Verfahren (z.B. durch Eintragen vorausgehender Auswertungen in eine Karte), Hindernisfreiheit zugesichert werden kann. Für Objekte jenseits des maximalen Abstands ist es ausreichend, wenn sie in einem späteren Zyklus klassifiziert werden.

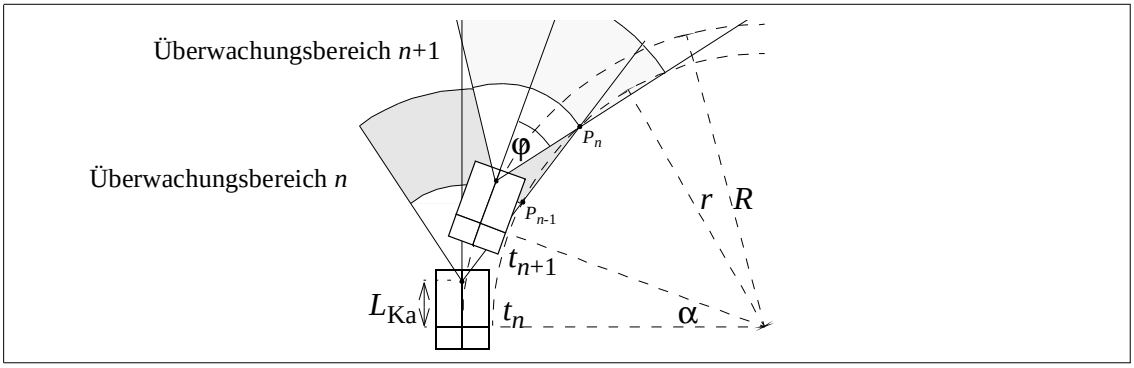


Abbildung 4.10: Konstruktionszeichnung für den Überwachungsbereich der Hindernisvermeidung bei Zusicherung des Piloten keinen Bogen kleiner als *R* zu fahren

Gegeben ist Radius $R$. Die Bildanalysezeit sei $\Delta t = t_{n+1} - t_n$. Das Fahrzeug fährt während der Bildauswertung bei konstanter Geschwindigkeit $v$ einen Bogen der Länge $l$. Der zurückgelegte Winkel des Kreises beträgt $\alpha = l/R$. Ausgehend von der Position der Kamera zum Zeitpunkt $t_n$ und $t_{n+1}$ ziehen wir zwei Geraden, die sich und den Kreis mit Radius $r$ in einem Punkt $P_n$ schneiden. Sei $\varphi$ als der Winkel definiert, der die Steigung beider Geraden (bezogen auf das Roboterkoordinatensystem zum Zeitpunkt $t_i$) festlegt. Anschaulich bestimmt der Winkel $\varphi$ eine Grenze des Gesichtsfeldes der Hinderniserkennung, welches unter der Annahme überwacht wird, daß nur statische Hindernisse existieren. Gegeben der Radius $R$, muß um Kollisionsfreiheit zusichern zu können, ein um die Breite des Fahrzeugs verringerter Radius $r$ eingesehen werden. Dabei sind alle Objekte die sich in der Kreisfläche um die Kamera zwischen $P_{n-1}$ und $P_n$ befinden, zu klassifizieren.

Abbildung 4.10 zeigt, wie der Blickwinkel $\varphi$ definiert wird. Sei $t_\Sigma$ die Reaktionszeit der Pipeline *Ka-H-P-R* und $T_H$ definiere die Periode, nach der die Kamera ein Bild unter dem gleichen Blickwinkel aufnimmt. Dann berechnet sich der Anhalteweg $l$ bei einer konstanten Geschwindigkeit $v$ und einer Bremsverzögerung $a$, der auf einem Kreisbogen mir Radius $R$ zurückgelegt wird, zu

$$f_l(v, a, t_\Sigma, T_H): \quad l = v^2/(2a) + (t_\Sigma + T_H)v. \tag{4.1}$$

Der Blickwinkel $\varphi$ ergibt sich mit

$$f_\varphi(R, l): \quad \varphi = \operatorname{acot}\frac{L_{Ka}RA - r\sqrt{-r^2 + (L_{Ka}^2 + R^2)A}}{r^2 - R^2A}. \tag{4.2}$$

Dabei sei $A = (2 - 2\cos\alpha)/(\sin\alpha)^2$ und $r$ der um die Breite des Fahrzeuges verringerte Radius $R$. Die zu klassifizierenden Objekte liegen dann in einer Entfernung zur Kamera von

$$f_d(l, R): \quad (d_{\min}, d_{\max}) = (d(P_{\mathrm{Ka}}, P_{n-1}),\ d(P_{\mathrm{Ka}}, P_n)). \tag{4.3}$$

Ist der Blickwinkel größer als das Gesichtsfeld der Kamera (bestimmt durch $R_f$), so muß der Kamerakopf geschwenkt werden. Die Periodendauer wird mittels eines Planungsmoduls $f_T$ berechnet.

Die Algorithmen der Bildverarbeitung benötigen für das kleinste in *H* modellierte Hindernis eine Mindestanzahl von Pixeln. Nun legt die Brennweite der Kamera, zusammen mit der konfigurierten Auflösung des Schnittstellenkarte (Frame Grabber) zum Einlesen der Bilder, eine maximale Entfernung fest, jenseits der das Objekt nicht mehr als Hindernis erkannt werden kann. Insbesondere muß $d_{\max}$ kleiner als diese Entfernung sein. $d_{\max}$ erhöht sich proportional zu dem Anhalteweg, also z. B. durch höhere Geschwindigkeiten oder durch eine längere Periodendauern. Die Periodendauer wird maßgeblich durch die Kamerakopfbewegungen bestimmt, um Bilder aus unterschiedlichen Sektoren, links und rechts der Fahrtrichtung aufzunehmen. Wird $\varphi$ erhöht, so steigt die für die Bewegung des Kamerakopfes benötigte Zeit in diskreten Schritten, abhängig von der Brennweite des Objektivs. Somit steigt die Periodendauer und die Entfernung des zu überwachenden Bereichs, der aber durch die Bildauflösung beschränkt ist. Höhere Auflösungen führen auch zu höheren Bildaufnahmezeiten.

Die Restriktionsvariablen der Elementaragenten und das Restriktionsnetz des Restriktionsnetzes ist in Abbildung 4.11 dargestellt.

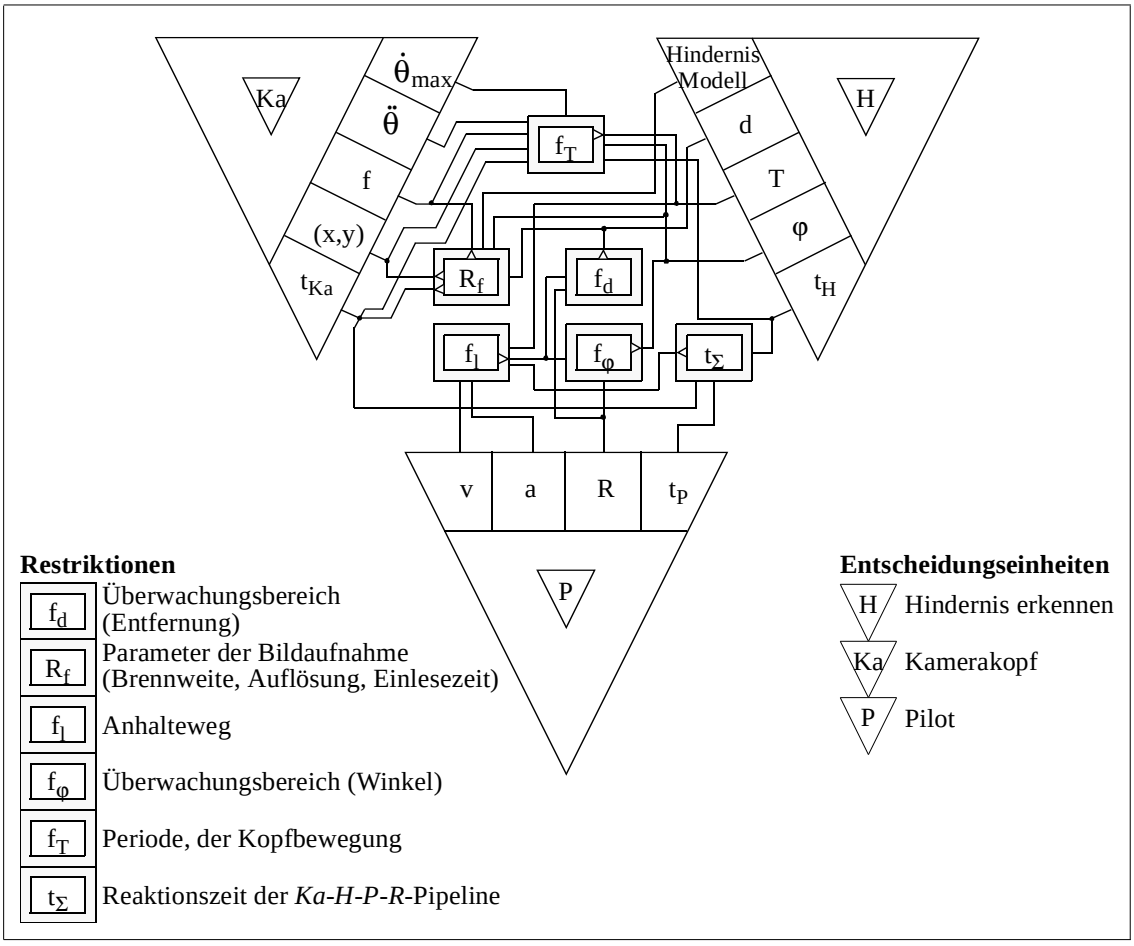


Abbildung 4.11: *Ka-H-P*-Restriktionsnetz

Die Restriktion des Anhalteweges ist durch Gleichung (4.1) beschrieben. Die Restriktionen für den Überwachungsbereich wird durch Gleichung (4.2) hinsichtlich ihres Winkels und durch Gleichung (4.3) hinsichtlich der Entfernung angegeben. Der Zusammenhang zwischen den Parametern der Bildaufnahme ist eine Relation über die Kenndaten der Schnittstellenkarte zum Einlesen des Bildes in Verbindung mit den Abbildungseigenschaften des Objektivs und der Kamera. Sie ist das Ergebnis einer Kalibrierung und kann nicht allgemein angegeben werden. Die Restriktion der Reaktionszeit der *Ka-H-P-R*-Pipeline ist durch die Summe der Einzelzeiten gegeben.

### 4.5.3 Weitere Entscheidungsnetzwerke

Das Entscheidungsnetzwerk zwischen *P* (Piloten) und *F* (Formation fahren) dient der Abstimmung der Trajektorie, die *P* abfahren darf. *P* kann die Trajektorie in vorgegebenen Grenzen autonom anpassen. Muß der Pilot des vorausfahrenden Fahrzeugs ($P_1$) z. B. vor einem Hindernis ausweichen, und überschreitet er dabei die ihm vorgegebenen Grenzen,

so stimmt $P_1$ die Trajektorie über das $F_1$-$P_1$-$F_2$-$P_2$-Entscheidungsnetzwerk der Fahrzeuge ab. Somit wird sichergestellt, daß die Fahrzeuge sich nicht zu weit voneinander entfernen und der Träger nicht verlorengeht.

Das Entscheidungsnetzwerk zwischen $L$ (Fahrzeuge losten), $V_{1,2}$ (Lotsen verfolgen) und $F_{1,2}$ (Formation fahren) wird benötigt, um die Trajektorie abzustimmen, nach der die Formation die Korridore abfährt. Die Entscheidungseinheiten der Elementaragenten $F_{1,2}$ bringen ihre Parameter über physische Beschränkungen der Formation ein. Die Entscheidungseinheiten der Elementaragenten $V_{1,2}$ koordinieren die Anordnungen beim Verfolgen des Lotsen (hintereinanderfahren, nebeneinanderfahren, Festlegen des vorderen bzw. linken Fahrzeugs). $L$ stimmt die zulässige Entfernung ab, die der Lotse den folgenden Fahrzeugen voraus sein darf und verwaltet Termine, die während der Fahrt eingehalten werden müssen.

Das Entscheidungsnetzwerk zwischen den Elementaragenten $U_{1,2,3}$ (Ultraschall) und $P_{1,2,3}$ (Pilot) der Fahrzeuge wird benötigt, um Störungen zwischen den Ultraschallsensoren der Fahrzeuge zu minimieren. Es wäre bereits ausreichend nur $U_{1,2,3}$ untereinander abzustimmen, jedoch können durch die Hinzunahme von $P_{1,2,3}$ die Sensorabfragen aneinander angepaßt werden. Die Piloten stimmen über das $U_{1,2,3}$-$P_{1,2,3}$-Entscheidungsnetzwerk ihre Abfragen so ab, daß ihnen Antwortzeiten für ihre Abfragen garantiert werden. Dies ist für Formationsfahrten wichtig, da das Ausbleiben eines Sensorergebnisses sich auf den Konvoi auswirkt, indem er z. B. zum Halten gezwungen wird.

### 4.5.4 Dynamisches Verhalten

Nachdem die Entscheidungsnetzwerke der konkreten Roboterarchitektur eingeführt wurden, wird nun untersucht, wie sich das System verhält, wenn es aufgrund eines äußeren Ereignisses aus einem Gleichgewichtszustand gebracht wird. Das Ereignis sei durch ein sich auf der Fahrroute befindendes Objekt gegeben, das von der Hinderniserkennung des vorderen Transportfahrzeugs nicht rechtzeitig klassifiziert werden konnte. Die Hinderniserkennung wird ein weiteres Bild auswerten und feststellen, daß es sich um eine Bodenunreinheit handelt und nicht um ein Hindernis. Die Aufgabe der Entscheidungsnetzwerke ist es, die Geschwindigkeit zu reduzieren, um Zeit für eine zweite Bildauswertung zu schaffen. Hierzu muß der Konvoi abbremsen und der Lotse über die Verzögerung informiert werden. Es wird angenommen, daß keine Termine einer Verzögerung entgegenstehen (andernfalls würde die Bodenunreinheit als Hindernis angesehen werden und vom Konvoi umfahren werden müssen).

Da das genannte Objekt nicht im ersten Bild klassifiziert werden konnte, benötigt $H$ (Hinderniserkennung) ein weiteres Bild aus der selben Blickrichtung. Während dieser Zeit würde sich das vordere Fahrzeug bei gleichbleibender Geschwindigkeit so nah an das Objekt annähern, daß ein sicheres Ausweichen nicht mehr möglich wäre. $H$ erzeugt also eine interne Restriktion für das Intervall $(d_{\min}, d_{\max})$ des Beobachtungsbereiches, so daß

bei der zweiten Aufnahme das Objekt darin enthalten ist. Daraufhin propagiert H das Intervall durch das Restriktionsnetz und berechnet eine Lösung für die verringerte Geschwindigkeit (*Propagierungsphase*). In der folgenden *Verhandlung* verfolgt *H* die *Strategie*, wenn nötig $\varphi$ zu reduzieren, um durch eine sich somit verkürzende Periode $T_H$ Zeit für die Klassifikation zu erhalten.

*H* unterbreitet nun *P* den Vorschlag, eine geringere Geschwindigkeit zu fahren. *P* kann nur in Absprache mit dem nachfolgenden Fahrzeug die Geschwindigkeit ändern, was durch interne Restriktionen für *P* (Parametereinschränkungen aus dem *F-P*-Entscheidungsnetzwerk) festgelegt wird. *P* hat nun die Wahl, die Parametereinschränkungen zu relaxieren oder einen Gegenvorschlag unter Berücksichtigung der Einschränkungen zu machen. In diesem Beispiel versucht *P,* die Geschwindigkeit im *F-P*-Entscheidungsnetzwerk zu verringern.

*P* ($P_1$) *propagiert* die von *H* vorgeschlagene Geschwindigkeit und unterbreitet den Verhandlungspartnern im *F-P*-Netzwerk ($F_1$, $P_2$, $F_2$) die berechnete Lösung: Die Trajektorie soll von den beiden Transportfahrzeugen mit geringerer Geschwindigkeit abgefahren werden. *P* verfolgt die Strategie, die Verhandlung dann abzubrechen, wenn eine Einigung nicht direkt möglich ist.

Die Verhandlungspartnern von *P* überprüfen daraufhin ihre internen Restriktionen und stimmen zu, wenn sie nicht verletzt sind. Ansonsten berechnen sie einen Gegenvorschlag. $F_1$ ermittelt nun, ob im *F-V-L*-Netzwerk Restriktionen hinsichtlich Verzögerungen vorliegen. Dies sei in unserem Beispiel nicht der Fall, so daß $F_1$ dem Vorschlag von *P* zustimmt. Nachdem auch $P_2$ und $F_2$ zugestimmt haben, kann *P* an *H* seine Zustimmung melden.

*H* bestätigt das Verhandlungsergebnis im *Ka-H-P*-Netzwerk, (*P* im *P-F*-Netzwerk) und daraufhin verringern *P* und $P_2$ die Geschwindigkeit.

Nun kann *H* das benötigte zweite Bild analysieren, und das Objekt erfolgreich als kein Hindernis klassifizieren. Die Einwirkung durch das äußere Ereignis ist damit abgeschlossen und *H* kann die interne Restriktion für $(d_{\min}, d_{\max})$ zurücknehmen. Dies führt zu einer erneuten *Propagierung* durch das *Ka-H-P*-Netzwerk und über den beschriebenen Mechanismus wird die Geschwindigkeit wieder angehoben.

# Kapitel 5

# Anwendung

Im vorliegenden Kapitel wird die Realisierung eines Interaktionsverfahrens diskutiert, welches eine fehlertolerante Kommunikation zwischen Agenten verwaltet. Die Kommunikation wurde in Stockholm bei der Weltmeisterschaft für Roboterfußball (Robo Cup) 1999 im Team CoPS (*Co-operative soccer playing robots*), Stuttgart eingesetzt.

Dieses Kapitel stellt ein Beispiel vor, welches bis auf das Modulkonzept alle Elemente des Interaktionsmodells enthält. In Abschnitt 5.4 wird auf die Definitionen verwiesen, welche zum Verständnis des Beispiels benötigt werden.

## 5.1 Aufgabenstellung

In Abschnitt 4.3.1 wurden verschiedene Ausprägungen von Datenpuffern in Datenflußnetzwerken eingeführt und in Abschnitt 4.3.2 verschiedene Interaktionsformen zwischen Datenpuffern und Elementaragenten besprochen.

Die Aufgabenstellung bestand darin, für die CoPS-Datenflußnetzwerke einen Datenpuffer zu realisieren, welcher die Sichtbarkeit *Team* besitzt. Aufgrund der Vereinfachungen bei der Architektur der Elementaragenten, die aufgrund des engen Zeitplans bis zur Weltmeisterschaft vorgenommen wurden, war die Verwendung des Fehlerkanals wie in Abschnitt 4.3.3 beschrieben, nicht vorgesehen. Somit wurde die Anforderung an die Interaktion zwischen Elementaragenten und Datenpuffer wie folgt festgelegt:

Die Elementaragenten, die mit dem Datenpuffer verbunden sind, schreiben regelmäßig Information in den Datenpuffer und lesen diesen bei Bedarf asynchron aus. Durch Eintreffen neuer Information sollen noch nicht ausgelesene Daten überschrieben werden. Das Ergebnis der Plausibilitätsprüfung in den lesenden Elementaragenten sollte nicht an den schreibenden Elementaragenten zurückgemeldet werden.

Aufgrund der technischen Rahmenbedingungen – Einsatz eines Funk-Ethernet-Netzwerkes – war die wichtigste Anforderung, eine fehlertolerante Kommunikation zu realisie-

ren, welche bei Auftrennung des Netzes die Kommunikation in den entstandenen Teilnetzen aufrecht erhält und bei Wiederherstellung des Netzes wieder zu einer Einheit verschmilzt. Desweiteren wurde gefordert, daß während des Betriebs neue Teammitglieder beitreten können und einzelne Teammitglieder ausscheiden können. Ein ungeordnetes Ausscheiden – etwa durch Systemausfall – sollte von der Realisierung toleriert werden.

## 5.2 Interaktionsnetz

Die Lösungsidee besteht darin, den Datenpuffer als eine (Kommunikations-) Ressource zu modellieren, welche die geforderte Fehlertoleranz bereitstellt.

Das Verfahren zur Initialisierung der Kommunikation, die Aufnahme weiterer Teammitglieder, das geordnete Verlassen einzelner Teammitglieder sowie das Beenden der Kommunikation werden durch ein Interaktionsnetz spezifiziert, in welchem die Teammitglieder durch Rollen und die Ressourcen durch Objekte abstrahiert werden. Dieses Interaktionsnetz kann damit unverändert für andere Realisierungen von Datenpuffern mit Sichtbarkeit *Team* wiederverwendet werden.

Die Elementaragenten und die Kommunikationsressource durchlaufen das Interaktionsverfahren als Marken, wobei sie phasenspezifisch Rollen bzw. Objekte annehmen (Abschnitt 3.2.5).

Weiterhin wurde die Rolle eines (menschlichen) Administrators vorgesehen, welcher den Status der Kommunikation überwachen und Maßnahmen zum Wiederanlauf auslösen kann.

Das Interaktionsnetz ist in Abbildung 5.1 abgebildet. Der Ablauf des Interaktionsverfahrens wird anhand verschiedener Szenarien beschrieben, wobei wir mit dem Standardszenario beginnen:

### 5.2.1 Standardszenario

Ein Team von $n$ Elementaragenten benötigt einen Datenpuffer, der durch die Kommunikationsressource realisiert ist. Im Standardszenario sind die Teammitglieder von Beginn an bekannt und beteiligen sich an dem Interaktionsverfahren. Weiterhin ist ein Administrator vorhanden. In Abbildung 5.2 ist das Interaktionsverfahren abermals dargestellt, wobei die nicht relevanten Netzstrukturen grau erscheinen.

Bei der Beschreibung des Ablaufs folgen wir den Phasen des Standardszenarios, wobei wir die Initialisierung unter einem Aufzählungspunkt zusammenfassen. Die Beschreibung möchte nur einen Überblick über den Ablauf des Verfahrens geben, für die detail-

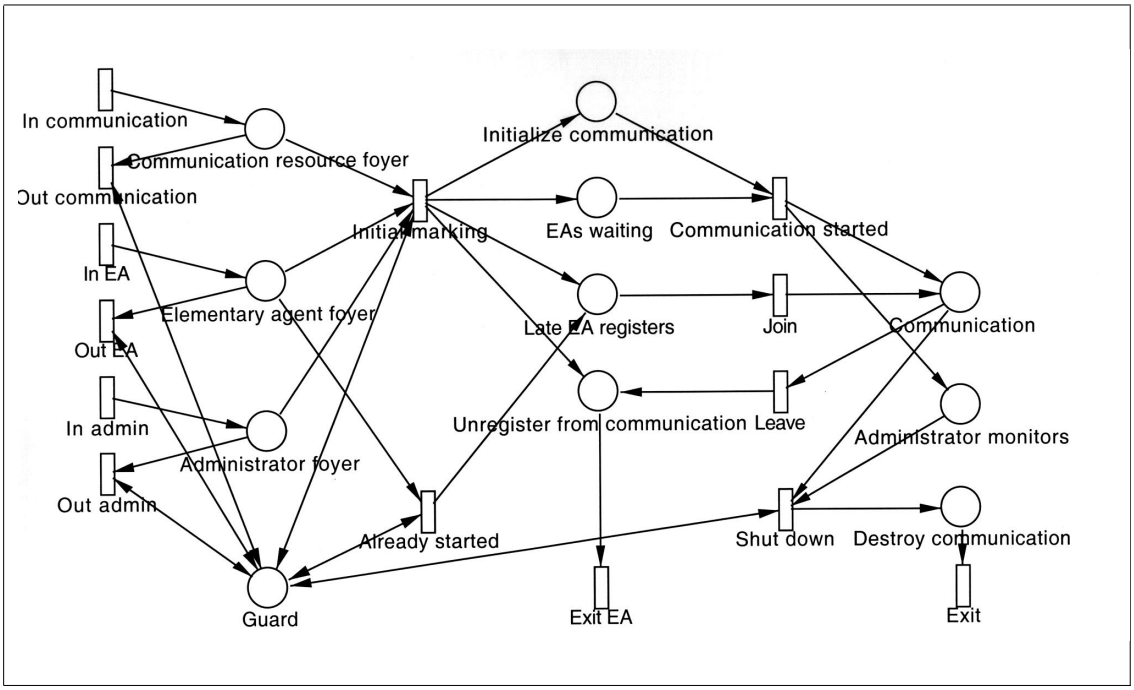


Abbildung 5.1: Netzstruktur des Interaktionsverfahren zur Administration eines Datenpuffers mit Team-Sichtbarkeit (Definition 3.32 auf Seite 88)

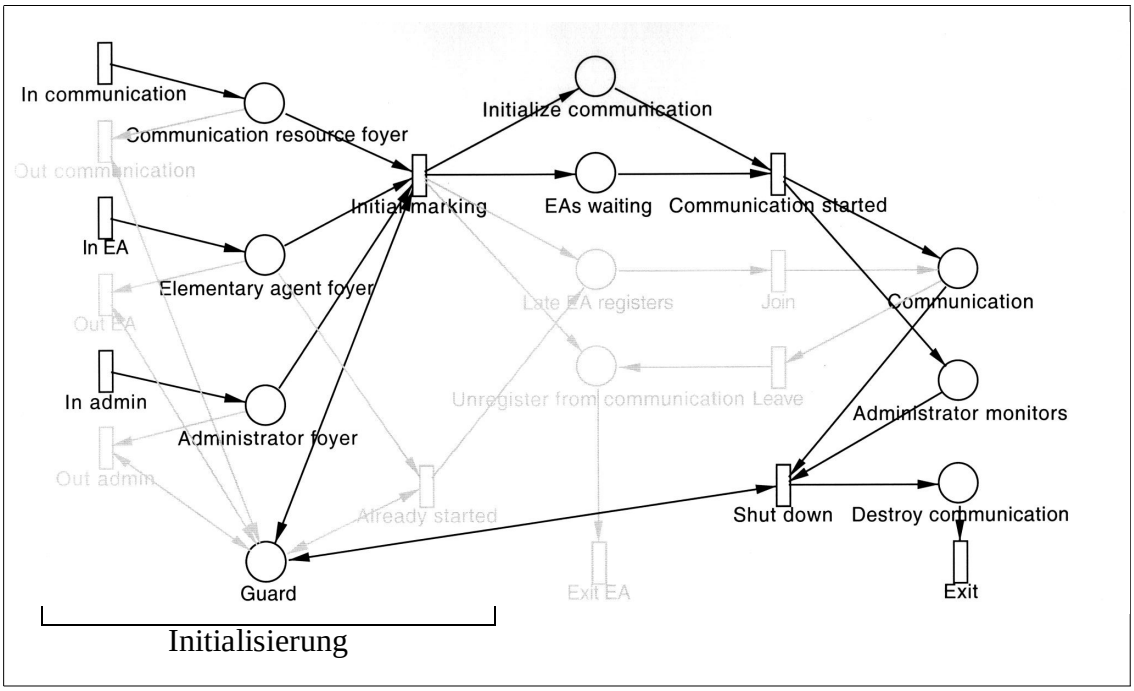


Abbildung 5.2: Standardszenario des Interaktionsverfahrens

lierte Spezifikation der Rollen, der Objekte, der Phasen und der Transitionen sei auf Abschnitt 5.4 hingewiesen. Der Leser möge zwischen den Detaillierungsebenen wechseln, um ein bestmöglichstes Verständnis zu erhalten.

1. Initialisierung:

   Entsprechend dem Muster für asynchrone Ein- und Ausgänge (Abschnitt 3.5.1) betritt die *Kommunikationsressource* ihr Foyer über die Transition *In communication*, die *Elementaragenten* betreten ihr Foyer über die Transition *In EA* und der *Administrator* betritt sein Foyer über die Transition *In admin*.

   Die Marken verweilen in ihrem jeweiligen Foyer bis das Interaktionsverfahren durch Schalten der Transition *Initial marking* eröffnet wird. Diese Transition wird aktiviert, sobald alle Teilnehmer in ihren Foyers eingetroffen sind. Die erforderliche Anzahl der Marken in den einzelnen Phasen wurde dem Interaktionsverfahren bei seine Instantiierung mitgegeben und ist in dem Objekt *State* der Phase *Guard* gespeichert.

   Schaltet die Transition *Initial marking*, so werden die Marken der Teilnehmer in folgenden Phasen bewegt:

   - Der Administrator wird zusammen mit der Kommunikationsressource in die Phase *Initialize communication* geschaltet.
   - Die Elementaragenten verweilen in der Phase *EAs waiting*.

   (Die grauen Kanten werden in den nachfolgenden Szenarien besprochen.)

2. Phase *Initialize communication*:

   Bevor der Datenpuffer genutzt werden kann, muß die Kommunikationsressource allen Elementaragenten bereitgestellt werden. In dieser Phase führt der Administrator die Initialisierung der Kommunikationsressource durch. Hierzu kennt er die am Interaktionsverfahren beteiligten Elementaragenten (welche ihm beim Schalten der Transition *Initial marking* mitgegeben wurden).

   Der Administrator meldet sich bei der Ressource mit der Adresse seines Bedienfeldes an, über welche er in folgenden Phasen Kontrollausgaben des Datenpuffers empfängt.

3. Phase *Communication*:

   Durch Schalten der Transition *Communication started* wird den Elementaragenten der Zugriff auf den Datenpuffer gewährt. Jeder Elementaragent erhält die Rolle *Communicator* zugewiesen, welche auf den durch das Objekt *Channel* repräsentierte Datenpuffer zugreifen kann. Statt den Informationsfluß über die Rollen und das Objekt ablaufen zu lassen, besitzt jeder Agent einen individuellen Anschluß an den Datenpuffer, welcher ihm zu Beginn der Phase durch seine Rolle übergeben wird. Über diesen Anschluß verschickt er seine Nutzdaten (dies sind Sensordaten über Eigenposition und Ballposition). Steuerdaten, wie den Antrag auf Ausscheiden aus der Kommunikation, werden von seiner Rolle bearbeitet.

4. Phase *Administrator monitors*:

   Nach der Initialisierung der Kommunikation ist es die Aufgabe des Administrators den Betrieb des Datenpuffers zu überwachen, wozu er in der Phase *Administrator monitors* die Rolle *Monitor* übernimmt. In der vorliegenden Realisierung besitzt der Administrator nur die Möglichkeit, die Kommunikation geordnet herunterzufahren.

5. Phase *Destroy communication*:

   Nachdem die Rolle *Monitor* der Phase *Administrator monitors* signalisierte, die Kommunikation herunterzufahren, schaltet die Transition *Shut Down*, welches unmittelbar den Eingang in das Interaktionsverfahren verschließt. Sie deaktiviert die Rollenmarken der Phase *Communication* und weist ihnen die Rolle *Disconnector* in der Phase *Destroy communication* zu. Die Marke des Objekts *Channel* wechselt zum Objekt *Closing Channel*. Der Administrator wechselt von der Rolle *Monitor* zur Rolle *Terminator* und schließt den Datenpuffer.

   Ist die Kommunikation abgebaut, so verlassen alle Rollenmarken gemeinsam das Interaktionsverfahren durch die Transition *Exit*. Das Interaktionsverfahren terminiert daraufhin (Definition 3.10 auf Seite 52).

Charakteristisch für Agentensysteme als offene System ist, daß während der Laufzeit neue Agenten in das System eintreten, bzw. das System wieder verlassen können. In den folgenden beiden Szenarien werden wir den geordneten Eintritt (Abschnitt 5.2.2) und den geordneten Austritt (Abschnitt 5.2.3) behandeln.

### 5.2.2 Szenario “neues Teammitglied”

In Abbildung 5.3 ist das Interaktionsverfahren abermals dargestellt, wobei die nicht am Szenario beteiligten Netzstrukturen grau erscheinen.

Nach der Eröffnung des Interaktionsverfahrens, können neue Elementaragenten in das Interaktionsverfahren aufgenommen werden. Sie betreten das Interaktionsverfahren über die Transition *In EA* und werden von der Transition *Already Started* in die Phase *Late EA registers* geschaltet.

Bei der Eröffnung des Interaktionsverfahrens durch die Transition *Initial marking* wurde die Phase *Late EA registers* bereits vorbereitet, neue Teammitglieder zu registrieren: Die Transition erzeugte eine Marke des Objekts *Check In*, welche der Kommunikationsressource zugeordnet ist. Über dieses Objekt meldet sich das neue Teammitglied in der Rolle *Late EA* bei der Kommunikationsressource an.

Sobald die Anmeldung erfolgreich abgeschlossen ist, kann der Teilnehmer der Phase *Communication* in der Rolle eines *Communicator* beitreten (Schalten der Transition *Join*).

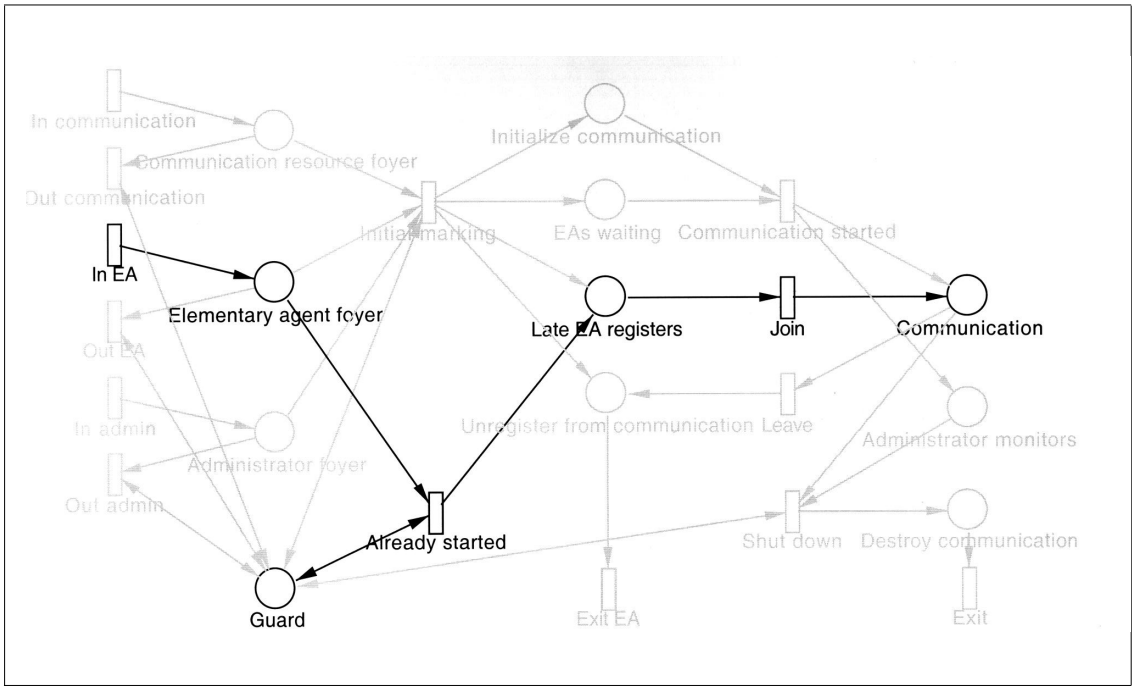


Abbildung 5.3: Szenario “neues Teammitglied”

### 5.2.3 Szenario “Frühzeitiges Verlassen des Teams”

In Abbildung 5.4 ist das Interaktionsverfahren abermals dargestellt, wobei die nicht am Szenario beteiligten Netzstrukturen grau erscheinen.

Ein Elementaragent in der Rolle eines *Communicator* kann auf Anfrage das Team verlassen. Hierzu schaltet ihn die Transition *Leave* aus der Phase *Communication* in die Phase *Unregister from communication*, wo er die Rolle *Leaving EA* annimmt.

Bei der Eröffnung des Interaktionsverfahrens durch die Transition *Initial marking* wurde die Phase *Unregister from communication* bereits vorbereitet, Teammitglieder abmelden zu können: Die Transition erzeugte eine Marke des Objekts *Check Out*, welche der Kommunikationsressource zugeordnet ist. Über dieses Objekt meldet sich das Teammitglied in der Rolle *Leaving EA* bei der Kommunikationsressource ab.

Sobald die Abmeldung erfolgreich abgeschlossen ist, kann der Teilnehmer das Interaktionsverfahren verlassen (Schalten der Transition *Join*).

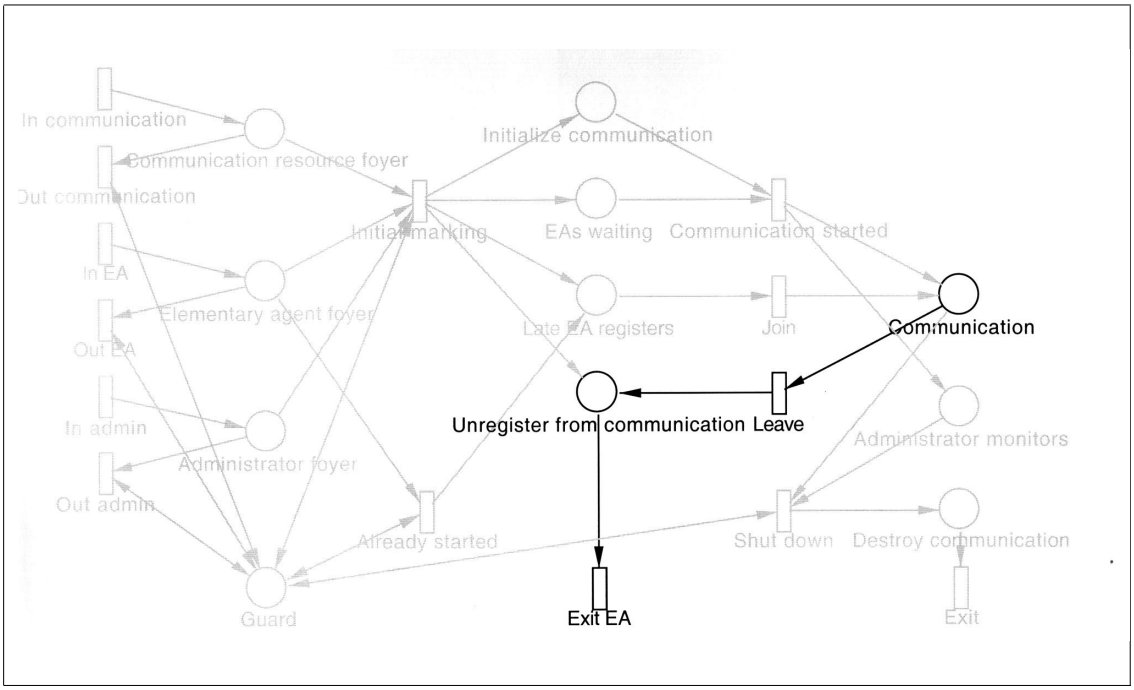


Abbildung 5.4: Szenario “Frühzeitiges Verlassen des Teams”

## 5.3 Im Interaktionsverfahren verwendete Dienste

Die Dienste definieren die Schnittstelle zwischen dem Interaktionsmodell und den Agenten und Ressourcen. Sie ist statisch. Änderungen an den Diensten, welche nicht abwärtskompatibel sind, erfordern Änderungen an allen Agenten und Ressourcen und an allen Interaktionsverfahren, welche auf die geänderten Teile der Dienste aufbauen.

Die Dienste dieser Anwendung wurden nicht unter dem Gesichtspunk entworfen, ohne Änderungen der genannten Art langfristig Bestand zu haben (insbesondere ist die Schnittstelle *Communication* von der zwischen den Elementaragenten ausgetauschten Information abhängig. Hier wären Mechanismen zu nutzen, bei denen Datentypen sich selbst beschreiben [Red 96], S. 179–195).

Die einzelnen Dienste werden im folgenden gemäß der statischen Programmstruktur (Paketstruktur) eingeführt, zu denen die einzelnen Schnittstellen zusammengefaßt wurden.

### 5.3.1 Paket network.dataflow.team.buffer

In dem Paket *comros.robSA.network.dataflow.team.buffer* sind die Dienste zusammengefaßt, welche von dem Datenpuffer mit Sichtbarkeit *Team* unterstützt werden (Abbildung 5.5). Jeder Dienst ist durch eine Schnittstellenklasse beschrieben, welche genau eine Methode besitzt. Die Klasse *State* definiert einen Aufzählungstyp, der den Zustand der Kommunikation beschreibt.

Der Datenpuffer wird mit einer Liste von Elemenaragenten gestartet und bietet eine Schnittstelle zu einem Administrator. Über diese kann der Administrator unabhängig von dem Interaktionsverfahren die ablaufende Kommunikation überwachen. Über den Dienst *Synchronize* kann der Datenpuffer gezwungen werden, den internen Zustand zu ermitteln und sich zu synchronisieren. Mit dem Dienst *Shut Down* wird der Datenpuffer geordnet angehalten. Über die Dienste *Add* und *Remove* können nachträglich Agenten beim Datenpuffer angemeldet bzw. frühzeitig abgemeldet werden.

Über den Dienst *Get Connector* kann für einen angemeldeten Agenten der ihm zugeordnete Zugang ausgelesen werden. Dieser Dienst übergibt eine Referenz auf den Zugang, welche sich wie die Schnittstellenklasse *Communication* verhält. Nachdem der Agent diese Referenz übergeben bekommen hat, kann er Daten in den Puffer schreiben und aus dem Puffer lesen.

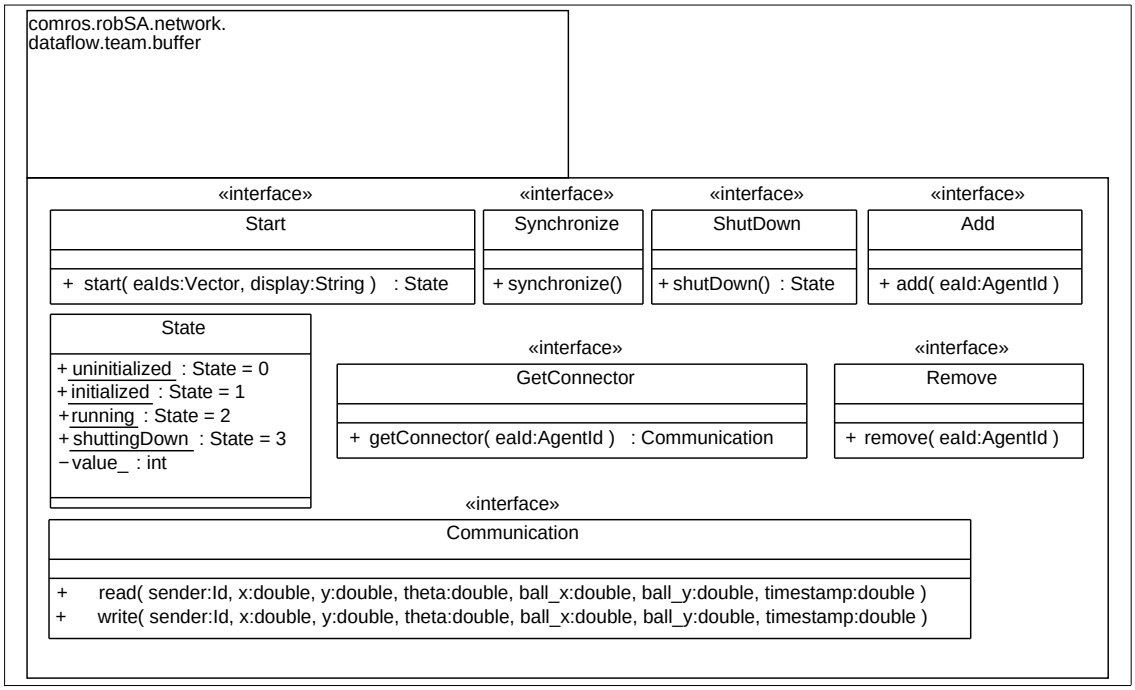


Abbildung 5.5: Dienste eines Datenpuffers mit Sichtbarkeit “Team” in UML-Notation

### 5.3.2 Paket network.dataflow.team.ea

In dem Paket *comros.robSA.network.dataflow.team.ea* sind die Dienste zusammengefaßt, welche von den Elementaragenten als Knoten im Datenflußnetzwerk unterstützt werden (Abbildung 5.6).

Ein Elementaragent wird durch den Aufruf des Dienstes *Connect* mit dem Datenpuffer verbunden und durch den Aufruf des Dienstes *Disconnect* wieder vom Datenpuffer gelöst.

Eine Besonderheit stellt der Dienst *Wait For Exit Req* dar: Ein Aufruf dieses Dienstes kehrt erst zurück, sobald der Elementaragent die Absicht hat, aus dem Datenverbund auszutreten. Dieser Dienst ist “revocable”, was bedeutet, das der Aufrufer sein Warten abbrechen kann und dem Elementaragenten damit die Möglichkeit entzieht, seine Absicht mitzuteilen. Der Aufruf des Dienstes ist also widerrufbar. Da der Widerruf eines solchen Dienstes zu einem beliebigen Zeitpunkt vom Aufrufer erfolgen kann, muß die Dienstimplementierung *rollback*-fähig sein.

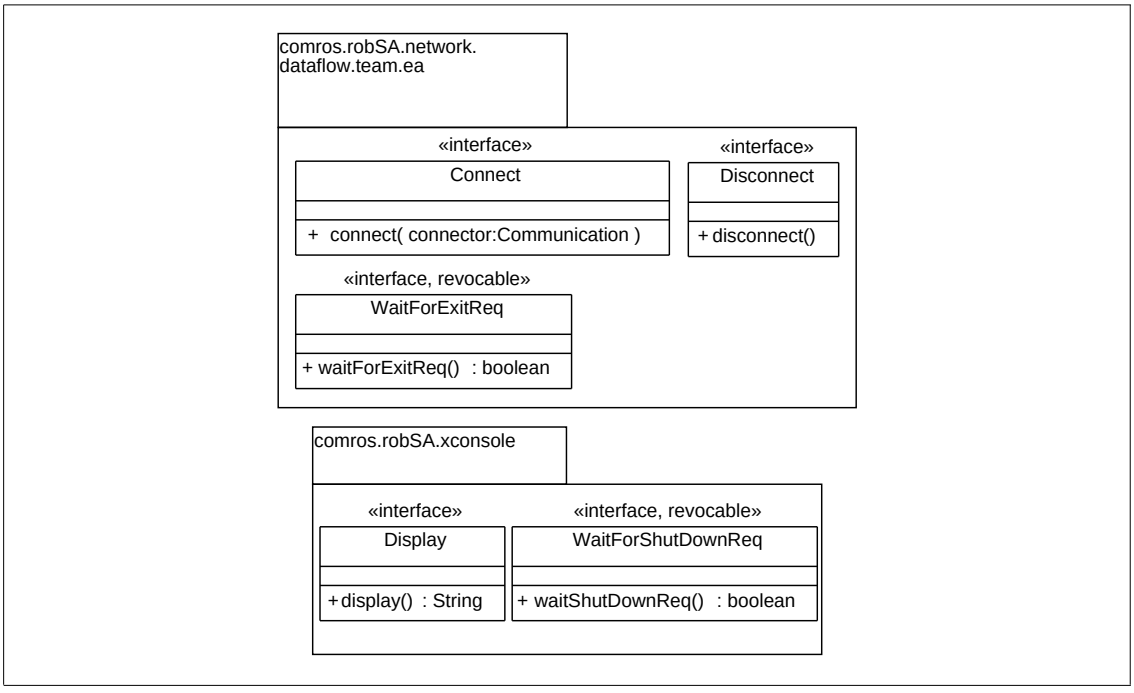


Abbildung 5.6: Dienste der Elementaragenten und des Administrators bezogen auf die Verwendung des Datenpuffers in UML-Notation

### 5.3.3 Paket xconsole

In dem Paket *comros.robSA.xconsole* sind die Dienste zusammengefaßt, welche von einem Administrator des Datenflußnetzwerkes unterstützt werden (Abbildung 5.6).

Mit dem Dienst Display erhält der Aufrufer einen Verweis auf das Bediengerät des Administrators. Mit Kenntnis dieser Referenz ist es möglich, beliebige graphische Bedienschnittstellen anzeigen zu lassen. Der Aufrufer des Dienstes *Wait For Shut Down Req* wartet solange, bis der Administrator die Entscheidung auf das Herunterfahren des Datenpuffers bekannt gibt.

### 5.3.4 Diskussion der Struktur der Dienste

In den Paketen *comros.robSA.network.dataflow.team.buffer* und *comros.robSA.network.dataflow.team.ea* wurden die Dienste zusammengefaßt, welche von dem Datenpuffer mit Sichtbarkeit *Team* bzw. von den Elementaragenten unterstützt werden. Dieses Vorgehen ist für diese Anwendung gerechtfertigt, sollte jedoch bei der Definition von Diensten für allgemeine Agentenanwendungen nur mit großer Vorsicht eingesetzt werden. Hierdurch wird eine Paketstruktur festgelegt, welche statisch ist und bei der Wiederverwendung in anderen Interaktionsverfahren keine adäquate Gruppierung bietet.

Die Dienste sollten vielmehr an typischen Handlungen ausgerichtet und entsprechend zu Paketen zusammengefaßt werden, wie es in den Dienstdefinitionen der Auktionsbeispiele aus dem Kapitel 3 umgesetzt wurde. Ist man in der Lage, über diese Dienste “Qualifikationsprofile” zu definieren, welche Agenten typischerweise besitzen, so sollte dieses unabhängig von der statischen Namensstrukur erfolgen, welche durch die Pakete definiert werden. Diese Profile können durch sogenannte “Organisatorische Rollen” beschrieben werden, die für bestimmte Positionen in einer Organisation gewisse “Qualifikationen” vordefinieren. Davon könnte hinsichtlich gewisser Individuen abgewichen werden, da durch das Interaktionsmodell sichergestellt ist, daß Individuen nur entsprechend ihrer Qualifikation und nicht entsprechend ihrer Position in der Organisation in Interaktionsverfahren eingesetzt werden.

## 5.4 Rollen, Objekte, Phasen und Transitionen

Im Abschnitt 5.2 wurde das Interaktionsverfahren informell beschrieben. In diesem Abschnitt werden nun die Rollen, Objekte, die Phasen und die Transitionen in Form von Tabellen definiert. Der die Tabelle referenzierende Text ist dabei als eine Konkretisierung der verbalen Beschreibung aus Abschnitt 5.2 zu verstehen und wird auf Besonderheiten

der einzelnen Einträge der Tabellen eingehen. Der Aufbau der Tabellen entspricht den Definitionen aus dem Interaktionsmodell der dritten Iterationsstufe (Tabelle 5.1).

| Komponente des Modells | Referenz |
|---|---|
| **Rolle** | *Definition 3.29 auf Seite 85* |
| **Objekt** | *Definition 3.27 auf Seite 84* |
| **Phase** | *Definition 3.20 auf Seite 64* |
| **Transition, (Interaktions-netz)** | *Definition 3.32 auf Seite 88* |

Tabelle 5.1: Im Abschnitt 5.4 verwendete Definitionen

Da in diesem Interaktionsverfahren keine Subinteraktionsverfahren verwendet werden, ist die Komponente $D^{I}_{Ref}$, welche die referenzierten Dienste eines Interaktionsobjekts angibt, die von einer Rolle oder einem Objekt verwendet werden, stets leer. Aus Gründen der Übersichtlichkeit entfällt diese Komponente in den die Rollen und Objekte definierenden Tabellen.

### 5.4.1 Anwendung des Musters für Ein- und Ausgänge

Wie in Abschnitt 3.5.1 beschrieben, wird in Interaktionsverfahren das Muster für einen bewachten, asynchronen Eingang verwendet, um Marken für Agenten oder Ressourcen im Interaktionsverfahren erstmals zu erzeugen und ihren Zutritt zu steuern. Dieses Muster wird für jede Quelle einer Rollen- oder Objektkette benötigt, und ist genau einem Markentyp – also einer Rolle oder einem Objekt – zugeordnet. Bestandteil des Musters ist eine Wartephase (in Tabelle 5.2 Foyer genannt), welche ausschließlich Marken des dem Muster zugeordneten Markentyps aufnehmen kann:

| **Phasen** | ***Communication resource foyer*** | ***Elementary agent foyer*** | ***Administrator foyer*** |
|---|---|---|---|
| $\gamma\|_R$ | $\varnothing$ | *Elementary Agent* | *Administrator* |
| $\gamma\|_O$ | *Team Buffer* | $\varnothing$ | $\varnothing$ |

Tabelle 5.2: Markentypen der Foyer-Phasen

Die Rollen in den Foyers besitzen keine Aktionen und verhalten sich somit passiv (Tabelle 5.3). Analog besitzt das Objekt aus Tabelle 5.4 keine definierten oder implementierten Dienste.

Die Eingangstransition, welche die Objektmarke der Kommunikationsressource erzeugt, ist in Tabelle 5.5 definiert. Die Eingangstransition, welche die Rollenmarken erzeugt, ist in Tabelle 5.6 definiert.

| **Rollen** | **Elementary agent foyer.Elementary Agent** | **Administrator foyer.Administrator** |
|---|---|---|
| $D_{Ref}^{A}$, **referenzierte Agentendienste** | ∅ | ∅ |
| $D_{Ref}^{O}$, **referenzierte Objektdienste** | ∅ | ∅ |
| $\grave{z}$, **Zustandsvektor** | ∅ | Vector *eaIds* |
| $ak$, **Aktion** | ∅ | ∅ |

Tabelle 5.3: Rolle *Administrator* der Phase *Administrator foyer* und Rolle *Elementary Agent* der Phase *Elementary agent foyer*

| **Objekt** | **Communication resource foyer.Team Buffer** |
|---|---|
| $D_{Def}$, **bereitgestellte Dienste** | ∅ |
| $D_{Ref}^{RS}$, **referenzierte Ressourcendienste** | ∅ |
| $\grave{z}$, **Zustandsvektor** | ∅ |
| $D_{Impl}$, **Dienstimplementierungen** | ∅ |

Tabelle 5.4: Objekt *Team Buffer* der Phase *Communication resource foyer*

| **Transition** | | ***In communication*** |
|---|---|---|
| $\cdot t$ | ∅ | |
| **Wächterbedingung von** $t$ | | ./. |
| $t\cdot$ | *Communication resource foyer* | newToken <- Team Buffer( ) |

Tabelle 5.5: Beschriftung der Transition *In communication* und ihrer Kanten

Das Muster für Ein- und Ausgänge (Abschnitt 3.5.1) sieht vor, daß eine Wächtermarke die Durchlässigkeit des Tors steuert. Ist das Tor dauerhaft verschlossen, so existiert zu jedem Eingang ein zugehöriger Ausgang, welcher die Marken aus der Wartephase entfernt. Der Wächter besitzt eine Komponente "*I_state*" in seinem Zustandsvektor $\grave{z}$, welche den Zustand des Interaktionsverfahrens speichert (Tabelle 5.9). Abbildung 5.7 zeigt das zugehörige Zustandsdiagramm. Durch Instantiieren des Interaktionsverfahrens wird die Wächterzustandskomponente "*I_state*" auf "*starting*" gesetzt. In diesem Zustand verweilen alle Marken in ihren Foyers. Schaltet die Transition *Initial marking*, so wird die Wächterzustandskomponente "*I_state*" auf "*running*" gesetzt. In diesem

| **Transition** | | ***In EA*** | ***In admin*** |
|---|---|---|---|
| $\cdot t$ | $\emptyset$ | | |
| **Wächterbedingung von** $t$ | | ./. | ./. |
| $t\cdot$ | *Elementary agent foyer* | newToken <- Elementary Agent( ) | |
| | *Administrator foyer* | | newToken <- Administrator( ) |

Tabelle 5.6: Beschriftung der Transition *In EA* und *In admin*, sowie ihrer Kanten

Zustand werden Marken, welche über die Eingänge *In communication* oder *In admin* das Interaktionsverfahren betreten, sofort durch die zugehörige Ausgangstransition entfernt. Elementaragenten können über die Transition *In EA* das Verfahren jederzeit betreten. Schaltet die Transition *Shut Down*, so wird die Wächterzustandskomponente "*I_state*" auf "*shutting down*" gesetzt. In diesem Zustand werden alle Marken, welche jetzt noch über die Eingänge *In communication*, *In admin* oder *In EA* das Interaktionsverfahren betreten, sofort durch die zugehörige Ausgangstransition entfernt. Die Instanz des Interaktionsverfahrens hört mit dem Verlassen der letzten Rollenmarke auf zu existieren.

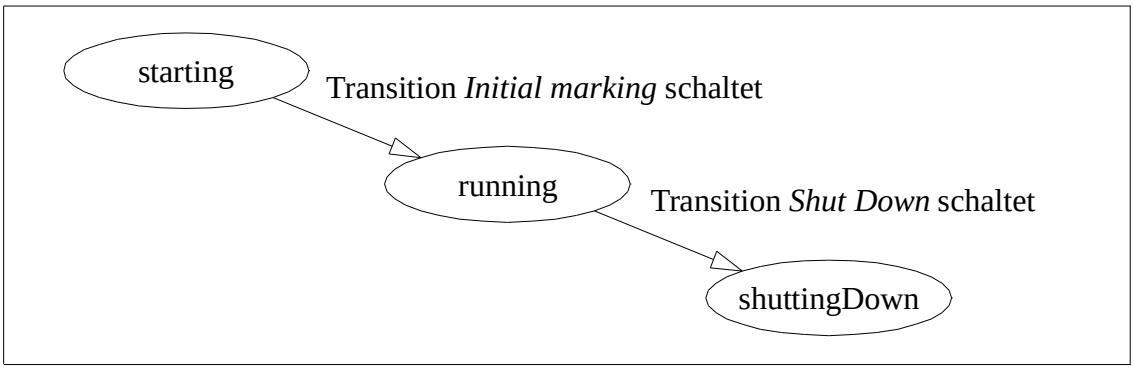


Abbildung 5.7: Zustandsdiagramm des Wächters

Somit ergeben sich die Ausgangstransitionen, welche den Eingängen zugeordnet sind, entsprechend der Tabelle 5.7.

## 5.4.2 Anfangsmarkierung

Die Menge der möglichen Anfangsmarkierungen ist durch die erforderliche Teilnahme einer Kommunikationsressource, eines Administrators und endlich vieler Elementaragenten bestimmt. Für diesen Spezialfall ist es ausreichend, genau eine Transition für die Modellierung der möglichen Anfangsmarkierungen vorzusehen, nämlich die Transition *Initial marking*. Die Bedingung für ihre Aktivierung verwendet Informationen,

| **Transition** | | ***Out communication*** | ***Out EA*** | ***Out admin*** |
|---|---|---|---|---|
| $\cdot t$ | *Communication resource foyer* | *o*: Team Buffer | | |
| | *Elementary agent foyer* | | *r*: Elementary Agent | |
| | *Administrator foyer* | | | *r*: Administrator |
| | *Guard* | *g*: State | *g*: State | *g*: State |
| **Wächterbedingung von** $t$ | | *g.state* == I_state.running \|\| *g.state* == I_state.shuttingDown | *g.state* == I_state.shuttingDown | *g.state* == I_state.running \|\| *g.state* == I_state.shuttingDown |
| $t\cdot$ | *Guard* | *g* | *g* | *g* |

Tabelle 5.7: Beschriftung der Transition *Out communication*, *Out EA* und *Out admin*, sowie ihrer Kanten

welche zum Zeitpunkt der Instantiierung des Interaktionsverfahrens vorliegen und dem Interaktionsverfahren mitgegeben wurden: die Anzahl der teilnehmenden Elementaragenten (*noEA*).

Erst wenn alle Teilnehmer in ihren Foyers durch Marken repräsentiert sind, startet das Interaktionsverfahren durch Schalten der Transition *Initial marking*. Den Zugriff auf diese Information ermöglicht das Objekt *State* der Phase *Guard* (Tabelle 5.8 und Tabelle 5.9).

| **Phase** | ***Guard*** |
|---|---|
| $\gamma\|_R$ | $\varnothing$ |
| $\gamma\|_O$ | *State* |

Tabelle 5.8: Markentypen der Phase *Guard*

| **Objekt** | **Guard.State** |
|---|---|
| $D_{Def}$, **bereitgestellte Dienste** | $\varnothing$ |
| $D_{Ref}^{RS}$, **referenzierte Ressourcendienste** | $\varnothing$ |
| $\vec{z}$, **Zustandsvektor** | I_state *state*, int *noEA* |
| $D_{Impl}$, **Dienstimplementierungen** | $\varnothing$ |

Tabelle 5.9: Objekt *State* der Phase *Guard*

Die Transition *Initial marking* (Tabelle 5.10) erzeugt also die Markierung des Interaktionsverfahrens, welche von den Teilnehmern abhängt. Sie schaltet den Administrator und die Kommunikationsressource in die Phase *Initialize communication*, alle Elementaragenten in die Phase *EAs waiting* und ändert den in der Wächtermarke gespeicherten Zustand des Interaktionsverfahrens auf "*running*".

Die Transition *Initial marking* erzeugt jedoch nicht nur die Markierung für das in Abschnitt 5.2.1 beschriebene Standardszenario, sondern auch für die Szenarien "Neues Teammitglied" (Abschnitt 5.2.2) und "Frühzeitiges Verlassen des Teams" (Abschnitt 5.2.3). Hierfür erzeugt die Transition für die Kommunikationsressource eine Marke vom Objekt *Check In* in der Phase *Late EA registers* und eine Marke vom Objekt *Check Out* in der Phase *Leaving EA*.

| **Transition** | | ***Initial marking*** |
|---|---|---|
| ·***t*** | *Communication resource foyer* | c: Team Buffer |
| | *Elementary agent foyer* | *ea*[*]: Elementary Agent |
| | *Administrator foyer* | *a*: Administrator |
| | *Guard* | *g*: State |
| **Wächterbedingung von *t*** | | *g.state* == I_state.starting && *ea*[*].*size* == *g.noEA* |
| ***t*·** | *Guard* | *g* <- State(*g.state* = I_state.running) |
| | *Initialize communication* | *a* <- Starter( eaIds = *ea*[*].Id(), *finished* = false ) (5.1)<br>*c* <- Communication Engine( )<br>nameService.bind(*a*, "Start", *c*);<br>nameService.bind(*a*, "Synchronize", *c*); (5.2) |
| | *EAs waiting* | *ea* <- Elementary Agent( ) |
| | *Late EA registers* | *c* <- Check In( ) |
| | *Unregister from communication* | *c* <- Check Out( ) |

Tabelle 5.10: Beschriftung der Transition *Initial marking* und ihrer Kanten

Die Variable *ea*[*] ist als Mengenvariable der Rolle *Elementary Agent* definiert.

## 5.4.3 Initialisierung und Betrieb der Kommunikation

Tabelle 5.11 definiert die Rollen und Objekte für die in diesem Abschnitt besprochenen Phasen.

| **Phasen** | ***Initialize communication*** | ***EAs waiting*** | ***Communication*** | ***Administrator monitors*** |
|---|---|---|---|---|
| $\gamma\|_R$ | *Starter* | *Elementary Agent* | *Communicator* | *Monitor* |
| $\gamma\|_O$ | *Communication Engine* | ∅ | *Channel* | ∅ |

Tabelle 5.11: Markentypen der Phasen *Initialize communication*, *EAs waiting*, *Communication* und *Administrator monitors*

In der Phase *Initialize communication* ist es die Aufgabe des Administrators in seiner Rolle als *Starter* den Datenpuffer zu initialisieren. Hierfür wurde ihm beim Schalten der Transition *Initial marking* (siehe (5.1), Tabelle 5.10) die Identität der Elementaragenten bekannt gegeben und die von ihm referenzierten Dienste an die Marke des Datenpuffers gebunden (siehe (5.2), Tabelle 5.10).

Definiert durch die Aktion seiner Rolle initialisiert der Administrator den Datenpuffer in (5.3), Tabelle 5.12. Der *Starter* übergibt der *Communication Engine* eine Referenz auf das Bedienfeld seines Agenten, welchem daraufhin Kontrollausgaben über den Zustand des Datenpuffers durch die Ressource angezeigt werden. Der durch die Kontrollausgaben stattfindende Informationsfluß wird nicht durch das Interaktionsverfahren gesteuert.

| **Rolle** | **Initialize communication.Starter** |
|---|---|
| $D^A_{Ref}$**, referenzierte Agentendienste** | import xconsole.Display |
| $D^O_{Ref}$**, referenzierte Objektdienste** | import network.dataflow.team.buffer.Start;<br>import network.dataflow.team.buffer.Synchronize; |
| $\hat{z}$**, Zustandsvektor** | import java.util.Vector;<br>Vector *eaIds*; boolean *finished* |
| $ak$**, Aktion** | { nameService.resolve("Start").<br>start( *eaIds*, *((*Display*) agent*).display( ) );<br>nameService.resolve("Synchronize").synchronize( );<br>*finished* = true; } (5.3) |

Tabelle 5.12: Rolle *Starter* der Phase *Initialize communication*

| **Objekt** | **Initialize communication.Communication Engine** |
|---|---|
| $D_{Def}$**, bereitgestellte Dienste** | implements network.dataflow.team.buffer.Start,<br>network.dataflow.team.buffer.Synchronize |
| $D^{RS}_{Ref}$**, referenzierte Ressourcendienste** | import network.dataflow.team.buffer.Start;<br>import network.dataflow.team.buffer.Synchronize; |

Tabelle 5.13: Objekt *Communication Engine* der Phase *Initialize communication*

| **Objekt** | **Initialize communication.Communication Engine** |
|---|---|
| $\dot{z}$ **, Zustandsvektor** | import network.dataflow.team.buffer.State;<br>State *state* |
| $D_{Impl}$ **, Dienstimplementierungen** | import java.util.Vector;<br>public State *start*( Vector *eaIds*, String *display* )<br>{ return *state* = ((Start) resource).start( *eaIds*, *display* ); }<br>public void *synchronize*( )<br>{ ((Synchronize) resource).synchronize( ); } |

Tabelle 5.13: Objekt *Communication Engine* der Phase *Initialize communication*

Das Ziel der Phase *Initialize communication* ist erreicht, sobald die Aktion des *Starter* beendet ist und als Zustand von *Communication Engine* “initialized” anzeigt wird (siehe (5.4), Tabelle 5.14). Daraufhin schaltet die Transition *Communication started*.

| **Transition** | | ***Communication started*** |
|---|---|---|
| $\cdot t$ | *Initialize communication* | *a*: Starter<br>*c*: Communication Engine |
| | *EAs waiting* | *ea*[*]: Elementary Agent |
| **Wächterbedingung von** $t$ | | *c.state* == State.initialized && *a.finished* (5.4) |
| $t \cdot$ | *Communication* | *c* <- Channel( )<br>*ea* <- Communicator( *exitReq* = false )<br>nameService.bind(*ea*, "GetConnector", *c*); |
| | *Administrator monitors* | *a* <- Monitor( *exitReq* = false ) (5.5) |

Tabelle 5.14: Beschriftung der Transition *Communication started* und ihrer Kante

Die Variable *ea*[*] ist als Mengenvariable der Rolle *Elementary Agent* definiert.

In der Phase *Communication* tauschen die Elementaragenten, denen die Rolle *Communicator* zugewiesen wurde, untereinander über den Datenpuffer Information aus. Statt jedes Informationspaket über die Rolle zum Objekt *Channel* und von dort zur Ressource zu leiten, erhält jeder Elementaragent seinen eigenen Anschluß an den Datenpuffer: Beim Eintritt in die Phase fragt jeder *Communicator* den *Channel* nach dem Anschluß seines Agenten (siehe (5.6), Tabelle 5.15). Diesen reicht der *Communicator* weiter an seinen Agenten (siehe (5.7), Tabelle 5.15), welcher daraufhin den Datenpuffer nutzen kann. (5.8), Tabelle 5.15 setzt den *Communicator* wartend, bis entweder sein Agent den Wunsch auf Austritt aus dem Team signalisiert, oder die Rollenmarke durch Schalten der Transition *Shut Down* aus der Phase geschaltet wird.

Der Administrator wurde beim Schalten der Transition *Communication started* in die Phase *Administrator monitors* geschaltet, in welcher er die Rolle *Monitor* annimmt (siehe

| **Rolle** | **Communication.Communicator** |
|---|---|
| $D^{A}_{Ref}$, **referenzierte Agentendienste** | import network.dataflow.team.ea.Connect<br>import network.dataflow.team.ea.WaitForExitReq |
| $D^{O}_{Ref}$, **referenzierte Objektdienste** | import network.dataflow.team.buffer.GetConnector; |
| $\vec{z}$, **Zustandsvektor** | boolean *exitReq* |
| $ak$, **Aktion** | { Connector *port* = nameService.resolve("GetConnector").<br>getConnector( *agent*.Id*()* ); (5.6)<br>*((*Connect*) agent*).connect( *port* ) ; (5.7)<br>*exitReq* = *((*WaitForExitReq*) agent*).waitForExitReq( ); } (5.8) |

Tabelle 5.15: Rolle *Communicator* der Phase *Communication*

| **Objekt** | **Communication.Channel** |
|---|---|
| $D_{Def}$, **bereitgestellte Dienste** | implements network.dataflow.team.buffer.GetConnector |
| $D^{RS}_{Ref}$, **referenzierte Ressourcendienste** | import network.dataflow.team.buffer.GetConnector; |
| $\vec{z}$, **Zustandsvektor** | $\varnothing$ |
| $D_{Impl}$, **Dienstimplementierungen** | public Communication *getConnector*( AgentId *eaId* )<br>{ return ((GetConnector) resource).getConnector( *eaId* ); } |

Tabelle 5.16: Objekt *Channel* der Phase *Communication*

(5.5), Tabelle 5.14). Die Ausführung der Überwachungsaufgabe obliegt der Autonomie des Administrators. Seine Rolle verweilt so lange in dieser Phase, bis der Administrator das Betriebsende des Datenpuffers anzeigt (siehe (5.9), Tabelle 5.17), wodurch die Transition *Shut Down* aktiviert wird (siehe (5.10), Tabelle 5.19).

| **Rolle** | **Administrator monitors.Monitor** |
|---|---|
| $D^{A}_{Ref}$, **referenzierte Agentendienste** | import xconsole.WaitForShutDownReq |
| $D^{O}_{Ref}$, **referenzierte Objektdienste** | $\varnothing$ |
| $\vec{z}$, **Zustandsvektor** | boolean *exitReq* |
| $ak$, **Aktion** | { *exitReq* = *((*WaitForShutDownReq*) agent*).<br>waitForShutDownReq( ); } (5.9) |

Tabelle 5.17: Rolle *Monitor* der Phase *Administrator monitors*

### 5.4.4 Geordneter Abbau der Kommunikation

| **Phase** | ***Destroy communication*** |
|---|---|
| $\gamma\|_R$ | *Terminator*<br>*Disconnector* |
| $\gamma\|_O$ | *Closing Channel* |

Tabelle 5.18: Markentypen der Phase *Destroy communication*

Schaltet die Transition *Shut Down*, so werden die Kommunikation und das Interaktionsverfahren geordnet angehalten. Zunächst wird unterbunden, daß neue Agenten dem Interaktionsverfahren nachträglich beitreten können. Hierfür ändert die Transition den in der Wächtermarke gespeicherten Zustand des Interaktionsverfahrens auf “shuttingDown” (siehe (5.11), Tabelle 5.19). Somit werden neu eintretende Agenten über die den Eingängen zugeordneten Ausgängen wieder entfernt (Tabelle 5.7).

Die Aktion der Rolle *Communicator* der Phase *Communication* weist eine Besonderheit auf. Die Rolle ruft den Dienst “WaitForShutDownReq” des Agenten auf, welcher jedoch im Standardszenario nicht zurückkehrt. Somit beendet sich die Aktion der Rolle nicht. Sie muß entsprechend der Definition 3.23 auf Seite 70 über die *Semantik des Schaltvorgangs hinsichtlich Rollen- und Objektmarken* durch Schalten der Transition *Shut Down* deaktiviert werden. Dies ist zulässig, da der Dienst “WaitForShutDownReq” als widerrufbar (*revocable*) deklariert wurde (Abbildung 5.6). Beim Deaktivieren der Aktion wird der Agent über den Widerruf informiert und führt daraufhin eigenständig geeignete Rücksetzmechanismen aus.

Sobald die Aktionen aller Rollenmarken deaktiviert sind, gibt es keine Rollenmarke mehr, welche Dienste des Objekts aufrufen könnte, so daß der Objektwechsel für die Ressource zum Objekt *Closing Channel* (Tabelle 5.21) sicher durchgeführt werden kann (siehe (5.13), Tabelle 5.19).

Die Elementaragenten erhalten in der Phase *Destroy communication* (Tabelle 5.18) die Rolle *Disconnector* zugewiesen (siehe (5.12), Tabelle 5.19), welche ihren Agenten von der Kommunikationsressource löst (siehe (5.14), Tabelle 5.20).

Der Administrator initiiert in der Rolle des *Terminator* den Abbau der Kommunikationsverbindung (siehe (5.15), Tabelle 5.22).

Erst wenn die Aktionen der Rollen aller Elementaragenten und des Administrators beendet sind, wird die Transition *Exit* aktiviert (siehe (5.16), Tabelle 5.23), welche alle Rollenmarken aus dem Interaktionsverfahren entfernt. Damit ist das Interaktionsverfahren beendet.

| **Transition** | | ***Shut Down*** | |
|---|---|---|---|
| $\cdot t$ | *Communication* | *ea*[*]: Communicator<br>c: Channel | |
| | *Administrator monitors* | *a*: Monitor | |
| | *Guard* | *g*: State | |
| **Wächterbedingung von *t*** | | *a.exitReq* | (5.10) |
| $t \cdot$ | *Guard* | *g* <- State( *g.state* = I_state.shuttingDown ) | (5.11) |
| | *Destroy communication* | *ea* <- Disconnector( *finished* = false ) | (5.12) |
| | | *a* <- Terminator( *finished* = false ) | |
| | | *c* <- Closing Channel( ) | (5.13) |
| | | nameService.bind(*a*, "ShutDown", *c*); | |

Tabelle 5.19: Beschriftung der Transition *Shut Down* und ihrer Kanten

Die Variable *ea*[*] ist als Mengenvariable der Rolle *Elementary Agent* definiert.

| **Rolle** | **Destroy communication.Disconnector** | |
|---|---|---|
| $D^{A}_{Ref}$**, referenzierte Agentendienste** | import network.dataflow.team.ea.Disconnect | |
| $D^{O}_{Ref}$**, referenzierte Objektdienste** | ∅ | |
| $z$**, Zustandsvektor** | boolean *finished* | |
| $ak$**, Aktion** | { *((*Disconnect*) agent*).disconnect( ) ; *finished* = true;} | (5.14) |

Tabelle 5.20: Rolle *Disconnector* der Phase *Destroy communication*

| **Objekt** | **Destroy communication.Closing Channel** |
|---|---|
| $D_{Def}$**, bereitgestellte Dienste** | implements network.dataflow.team.buffer.ShutDown |
| $D^{RS}_{Ref}$**, referenzierte Ressourcendienste** | import network.dataflow.team.buffer.ShutDown; |
| $z$**, Zustandsvektor** | import network.dataflow.team.buffer.State;<br>State *state* |
| $D_{Impl}$**, Dienstimplementierungen** | public State *shutDown*( )<br>{ return *state* = ((ShutDown) resource).shutDown( ); } |

Tabelle 5.21: Objekt *Closing Channel* der Phase *Destroy communication*

| **Rolle** | **Destroy communication.Terminator** |
|---|---|
| $D^{A}_{Ref}$**, referenzierte Agentendienste** | ∅ |
| $D^{O}_{Ref}$**, referenzierte Objektdienste** | import network.dataflow.team.buffer.ShutDown; |
| $\vec{z}$**, Zustandsvektor** | boolean *finished* |
| $ak$**, Aktion** | { nameService.resolve("ShutDown").shutDown( ); *finished* = true; } (5.15) |

Tabelle 5.22: Rolle *Terminator* der Phase *Destroy communication*

| **Transition** | | ***Exit*** |
|---|---|---|
| $\cdot t$ | *Destroy communication* | *ea*[*]: Disconnector<br>*a*: Terminator |
| **Wächterbedingung von** $t$ | | *ea.finished* && *a.finished* (5.16) |
| $t\cdot$ | ∅ | |

Tabelle 5.23: Beschriftung der Transition *Exit* und ihrer Kanten

Die Variable *ea*[*] ist als Mengenvariable der Rolle *Elementary Agent* definiert.

## 5.4.5 Verspäteter Zutritt

Nach der Eröffnung des Interaktionsverfahrens durch die Transition *Initial marking* wurde der Wächter in Form des Objekts *State* in der Phase *Guard* so konfiguriert, daß nur noch Elementaragenten durch das Tor, welches durch die Transition *Already Started* definiert ist, in das laufende Interaktion eintreten können. Ein verspäteter Elementaragent wird in die Phase *Late EA registers* (Tabelle 5.24) geschaltet, in welcher er die Rolle *Late EA* (Tabelle 5.26) annimmt.

Dort hat er Zugriff (siehe (5.18), Tabelle 5.25) auf das Objekt *Check In* (Tabelle 5.27) über welches er sich beim Datenpuffer anmeldet (siehe (5.20), Tabelle 5.28). Nach erfolgreicher Anmeldung schaltet ihn die Transition *Join* in die Phase *Communication*, in der er die Rolle eines *Communicator* annimmt.

| **Phase** | ***Late EA registers*** |
|---|---|
| $\gamma\|_R$ | *Late EA* |
| $\gamma\|_O$ | *Check In* |

Tabelle 5.24: Markentypen der Phase *Late EA registers*

| **Transition** | | ***Already Started*** |
|---|---|---|
| $\cdot t$ | *Elementary agent foyer* | *ea*: Elementary Agent |
| | *Late EA registers* | *c*[&]: Check In (5.17) |
| | *Guard* | *g*: State |
| **Wächterbedingung von $t$** | | *g.state* == I_state.running |
| $t\cdot$ | *Late EA registers* | *ea* <- Late EA( *finished* = false)<br>nameService.bind(e*a*, "Add", *c*[&]); (5.18) |

Tabelle 5.25: Beschriftung der Transition *Exit* und ihrer Kanten

Die Variable *c*[&] ist als Referenzvariable des Objekts *Check In* definiert.

| **Rolle** | **Late EA registers.Late EA** |
|---|---|
| $D^{A}_{Ref}$**, referenzierte Agentendienste** | ∅ |
| $D^{O}_{Ref}$**, referenzierte Objektdienste** | import network.dataflow.team.buffer.Add; |
| $\dot{z}$**, Zustandsvektor** | boolean *finished* |
| $ak$**, Aktion** | { nameService.resolve("Add").add( *agent*.Id );<br>*finished* = true*;* } |

Tabelle 5.26: Rolle *Late EA* der Phase *Late EA registers*

| **Objekt** | **Late EA registers.Check In** |
|---|---|
| $D_{Def}$**, bereitgestellte Dienste** | implements network.dataflow.team.buffer.Add |
| $D^{RS}_{Ref}$**, referenzierte Ressourcendienste** | import network.dataflow.team.buffer.Add; |
| $\dot{z}$**, Zustandsvektor** | ∅ |
| $D_{Impl}$**, Dienstimplementierungen** | public State *add*( )<br>{ ((Add) resource).add( ); } |

Tabelle 5.27: Objekt *Check In* der Phase *Late EA registers*

Interessant ist noch die Verwendung von Referenzvariablen in (5.17), Tabelle 5.25 und (5.19), Tabelle 5.28 als Beschriftung von eingehenden Kanten der Transitionen. Nach Definition 3.33 auf Seite 93 über die *Schaltregel für Interaktionsnetze* könnte die Transition nicht schalten, da auf das jeweilige Objekt durch Rollen, welche in der gleichen Phase sich befinden, noch zugegriffen wird. Das Objekt ist $\omega$-blockiert. Durch die Verwendung der Referenzvariablen ist die jeweilige Objektmarke nur durch eine Referenz

| **Transition** | | ***Join*** | |
|---|---|---|---|
| $\cdot t$ | *Late EA registers* | *ea*: Late EA | |
| | *Communication* | *c*[&]: Channel | (5.19) |
| **Wächterbedingung von *t*** | | *ea.finished* | (5.20) |
| $t \cdot$ | *Communication* | *ea* <- Communicator( *exitReq* = false )<br>nameService.bind(*ea*, "GetConnector", *c*[&]); | |

Tabelle 5.28: Beschriftung der Transition *Join* und ihrer Kanten

Die Variable *c*[&] ist als Referenzvariable des Objekts *Check In* definiert.

am Schaltvorgang beteiligt, die ausschließlich in Kontexttermen verwendet werden kann. Durch die Interpretation und Ausführung dieser Terme ist eine Beeinflussung mit laufenden Zugriffen von Rollen ausgeschlossen.

### 5.4.6 Frühzeitiges Ausscheiden

Die Aktion der Rolle *Communicator* der Phase *Communication* ruft den Dienst “WaitForExitReq” des Agenten auf. Läßt der Agent den Aufruf zurückkehren, so signalisiert er der Rolle den Wunsch auf Austritt aus dem Interaktionsverfahren.

Daraufhin schaltet ihn die Transition *Leave* (Tabelle 5.30) in die Phase *Unregister from communication* (Tabelle 5.29), in der er die Rolle *Leaving EA* (Tabelle 5.31) annimmt und Zugriff auf das Objekt *Check Out* (Tabelle 5.32) besitzt, über das er sich vom Datenpuffer abmeldet. Hat die Abmeldung erfolgreich stattgefunden, schaltet ihn die Transition *Exit EA* (Tabelle 5.33) aus dem Interaktionsverfahren.

| **Phase** | ***Unregister from communication*** |
|---|---|
| $\gamma\|_R$ | *Leaving EA* |
| $\gamma\|_O$ | *Check Out* |

Tabelle 5.29: Markentypen der Phase *Unregister from communication*

## 5.5 Kommunikationsressource

Die den Datenpuffer realisierende Kommunikationsressource besteht aus einer Vielzahl von Prozessen (siehe Abbildung 5.8), deren Anzahl quadratisch mit der Anzahl der ein-

| **Transition** | | ***Leave*** |
|---|---|---|
| $\cdot t$ | *Communication* | *ea*[*]: Communicator |
| | *Unregister from communication* | *c*[&]: Check Out |
| **Wächterbedingung von *t*** | | *ea.exitReq* |
| $t \cdot$ | *Unregister from communication* | *ea* <- Leaving EA( *finished* = false )<br>nameService.bind(*ea*, "Remove", *c*[&]); |

Tabelle 5.30: Beschriftung der Transition *Leave* und ihrer Kanten

Die Variable *ea*[*] ist als Mengenvariable der Rolle *Elementary Agent* definiert. Die Variable *c*[&] ist als Referenzvariable des Objekts *Check In* definiert.

| **Rolle** | **Unregister from communication.Leaving EA** |
|---|---|
| $D^{A}_{Ref}$, **referenzierte Agentendienste** | ∅ |
| $D^{O}_{Ref}$, **referenzierte Objektdienste** | import network.dataflow.team.buffer.Remove; |
| $\vec{z}$, **Zustandsvektor** | boolean *finished* |
| $ak$, **Aktion** | { nameService.resolve("Remove").remove( *agent*.Id );<br>*finished* = true; } |

Tabelle 5.31: Rolle *Leaving EA* der Phase *Unregister from communication*

| **Objekt** | **Unregister from communication.Check Out** |
|---|---|
| $D_{Def}$, **bereitgestellte Dienste** | implements network.dataflow.team.buffer.Remove |
| $D^{RS}_{Ref}$, **referenzierte Ressourcendienste** | import network.dataflow.team.buffer.Remove; |
| $\vec{z}$, **Zustandsvektor** | ∅ |
| $D_{Impl}$, **Dienstimplementierungen** | public State *remove*( )<br>{ ((Remove) resource).remove( ); } |

Tabelle 5.32: Objekt *Check Out* der Phase *Unregister from communication*

| **Transition** | | ***Exit EA*** |
|---|---|---|
| $\cdot t$ | *Unregister from communication* | *ea*: Leaving EA |

Tabelle 5.33: Beschriftung der Transition *Exit EA* und ihrer Kanten

| **Transition** | | ***Exit EA*** |
|---|---|---|
| **Wächterbedingung von *t*** | | *ea.finished* |
| ***t*·** | ∅ | |

Tabelle 5.33: Beschriftung der Transition *Exit EA* und ihrer Kanten

gesetzten Rechner wächst. Die einem *Elementaragenten* übergebene Referenz auf das Dienstobjekt der Klasse *Communication* wird von einem *Serverprozeß* verwaltet, welcher für alle auf einem Rechner laufenden Elementaragenten zuständig ist.

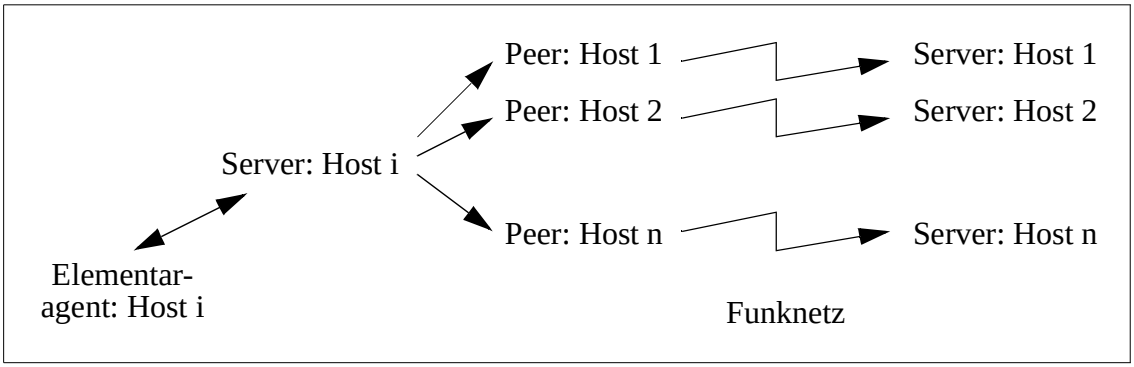


Abbildung 5.8: Prozeßlandschaft der Kommunikationsressource

Wird von dem Elementaragenten *schreibend* auf den Datenpuffer zugegriffen, so nimmt der genannte Serverprozeß die Nachricht entgegen und verteilt sie an sogenannte *Peerprozesse*, welche die Kommunikation zu den Partnerrechnern abwickeln. Die Nachrichtenübermittlung vom Serverprozeß zum Peerprozeß erfolgt *nicht blockierend*. Nachdem die Nachrichten an alle Peerprozesse weitergeleitet wurden, kehrt der Kontrollfluß zum Elementaragenten zurück.

Die Peerprozesse leiten die Nachricht über das Netzwerk an die Serverprozesse der entfernten Rechner weiter, welche die Nachricht in ihren Puffer stellen. Die Weiterleitung erfolgt *blockierend*. Dadurch ist es dem Peerprozeß möglich, Kommunikationsfehler zu entdecken und geeignet darauf zu reagieren. In der vorliegenden Implementierung wird nach erfolglosem Versenden einer Nachricht die logische Verbindung zum Partnerprozeß für eine Zeitspanne unterbrochen. Nachrichten, welche innerhalb der Zeitspanne eintreffen, werden verworfen. Sobald die Zeitspanne verstrichen ist, wird versucht, neu eintreffende Nachrichten wieder zuzustellen.

Durch diesen Mechanismus ist es möglich Netzwerkausfälle oder Segmentierungen des Netzes zu tolerieren und mit physisch getrennten (logisch jedoch zusammenhängenden) Datenpuffern weiterzuarbeiten. Sobald keine Störung mehr vorliegt, wird nach einer gewissen Zeitspanne wieder zu einem einzigen Datenpuffer zurückgekehrt.

Ein *lesender* Zugriff auf den Datenpuffer wird unmittelbar durch den lokalen Serverprozeß beantwortet.

Für *nachträglich registrierte* Elementaragenten wird, falls durch sie ein weiterer Rechner hinzukommt, ein Serverprozeß und $n + 1$ Peerprozesse auf dem hinzukommenden Rechner gestartet, sowie auf den $n$ bisherigen Rechnern jeweils ein Peerprozeß nachgestartet. Desweiteren werden die neuen Prozesse den bisherigen Prozessen bekannt gemacht.

Beim *frühzeitigen Austreten* von Elementaragenten werden, sobald kein Elementaragent mehr auf dem entsprechenden Rechner mit dem Datenpuffer verbunden ist, der Serverprozeß und $n$ Peerprozesse angehalten sowie auf den verbleibenden $n - 1$ Rechnern diezugehörigen Peerprozesse gestoppt. Das Fehlen der gestoppten Prozesse wird den anderen Prozessen mitgeteilt.

Alle Prozesse tragen sich bei einem *Namensdienst* ein und vor Beenden des Prozesses wieder aus. Das Informieren über Änderungen in der Prozeßlandschaft erfolgt über einen Triggermechanismus, welcher alle Prozesse auffordert, ihre Prozeßlisten zu aktualisieren. Die Prozesse greifen daraufhin auf den Namensdienst zu und lesen die für sie relevante Information aus.

Ein *Administrator* hat die Möglichkeit, diesen Triggermechanismus von Hand auszulösen. Dies ist dann nötig, wenn durch einen ungeordneten Abbruch von Kommunikationsprozessen manuelle Eingriffe in den Namensdienst erfolgen, an welche sich die Kommunikationsressource anpassen soll.

# Kapitel 6

# Schlußbemerkungen

## 6.1 Zusammenfassung

Die vorliegende Dissertation stellt eine Weiterentwicklung der Agententechnologie vor, durch die kohärentes Verhalten von Agenten mit Hilfe sogenannter *Interaktionsverfahren* zum Entwurfszeitpunkt entworfen und zur Ausführungszeit sichergestellt werden kann. Interaktionsverfahren werden durch ein entsprechendes *Interaktionsmodell* beschrieben.

Die Grundidee besteht darin, daß Agenten durch die Teilnahme an einem Interaktionsverfahren einen Teil ihrer Autonomie an das Verfahren und seine Entscheidungsmechanismen abtreten und sich diesem unterordnen. Dies wird dadurch erzielt, daß ein Interaktionsverfahren die Koordination der an ihn übertragenen Kompetenzen übernimmt. Ein Interaktionsverfahren ist somit gegenüber den teilnehmenden Agenten weisungsbefugt.

Um an Interaktionsverfahren teilnehmen zu können muß ein Agent eine Schnittstelle unterstützen, welche es dem Interaktionsverfahren ermöglicht auf die an ihn übertragen Kompetenz zuzugreifen und die kollektiv getroffenen Entscheidungen im einzelnen durchzusetzen. Hierzu sind unabhängig von Interaktionsverfahren für einen Anwendungsbereich Dienstklassen definiert, welche Schnittstellen zu Fähigkeiten eines Agenten bilden.

Ein Interaktionsverfahren definiert unter der Verwendung von Dienstklassen das Ablaufschema einer Interaktion. Das Ablaufschema abstrahiert von Agenten in Form von Rollen. Das Schema ist in einzelne Phasen strukturiert und definiert, wie die Rollen untereinander interagieren. Rollen sind die kleinsten aktiven Einheiten des Interaktionsmodells und nur innerhalb einer Phase gültig. Agenten, welche an einem Interaktionsverfahren teilnehmen, werden durch Rollen gesteuert, die Ihnen in den jeweiligen Phasen zugewiesen werden.

Dem Leser könnte das Bild eines Marionettentheaters in den Sinn gekommen sein, bei dem die Marionetten die Agenten und die Rollen dem Gestell entsprechen, welches auf

die Marionette über Fäden zugreift. Dieses Bild entspricht einer Entartung des Konzeptes dahingehend, daß die Marionetten keine Autonomie besitzen: Eine Marionette ist weder selbständig, noch hat sie ein Recht, sich eigenständig zu verhalten. Im Gegensatz zu dem Bild kann ein Agent über die Schnittstelle zwischen Rollen und Agenten das Verhalten seiner Rollen beeinflussen und somit indirekt auf den Ablauf des Interaktionsverfahrens Einfluß nehmen. Die Mechanismen des Interaktionsverfahrens stellen dennoch sicher, daß die Interaktion konform dem Entwurf abläuft. Wann und aufgrund welcher Information eine Rolle beeinflußt werden kann, ist vom Verfahren in den einzelnen Rollen festgelegt.

Diese Grundidee wurde zu einem Interaktionsmodell ausgebaut, in dem außer Agenten auch passive Ressourcen in eine Interaktion eingebunden werden können. Weiterhin stehen dem Entwickler Möglichkeiten zur Modularisierung zur Verfügung, mit deren Hilfe er einzelne Interaktionsverfahren strukturieren und geeignete Ausschnitte aus Interaktionen in anderen Verfahren wiederverwenden kann.

Das Interaktionsmodell wurde in drei Iterationsstufen eingeführt und deren Eignung jeweils durch den Entwurf eines Interaktionsverfahrens in Form einer Auktion evaluiert.

Als zweiter Schwerpunkt wurde die agentenorientierte Roboterarchitektur in Comros vorgestellt, in welcher die einzelnen Sensoren mit den zugehörigen Aktoren gruppiert und durch einen sogenannten Elementaragenten gesteuert werden. Der Elementaragent definiert selbst wieder ein virtuelles Sensor-Aktor-Paar. Elementaragenten werden durch Datenflußnetzwerke verbunden, welche aus einzelnen überlagerten Pipelines bestehen, welche das arbeitsteilige Zusammenwirken hinsichtlich einer Aufgabe beschreiben. Die Elementaragenten koordinieren sich über sogenannte Entscheidungsnetzwerke, welche die Weltmodelle der einzelnen Agenten durch Einsatz der Restriktionstechnik konsistent halten. Die Entscheidungsnetzwerke werden als Interaktionsverfahren definiert. Am Beispiel eines Gedankenexperiments wurde die Umsetzung der Architektur dargestellt

Anhand einer Komponente des Datenflußnetzwerkes, eines Datenpuffers mit Teamsichtbarkeit, wurde die Umsetzbarkeit eines im Interaktionsmodell spezifizierten Interaktionsverfahren nachgewiesen.

## 6.2 Ausblick

Die Arbeiten am Architekturkonzept in Comros sind nicht abgeschlossen, so daß die vorliegende Arbeit als Zwischenergebnis zu betrachten ist. Es steht zu erwarten, daß die Erfahrungen aus der Anwendung der Architektur in verschiedenen Bereichen Verfeinerungen notwendig erscheinen lassen – projektiert ist die Anwendung im SFB 514 “Aktive Exploration mittels Sensor/Aktor-Kopplung für adaptive Mess- und Prüftechnik”, im SFB 467 der bereits mehrfach genannt wurde, im Bereich der Sozionik in einem Projekt mit der Universität Bamberg mit dem Titel “Die Entstehung und Änderung sozia-

ler Strukturen in Gruppen intelligenter, multimotivierter, emotionaler Agenten" sowie im ebenfalls schon genannten Roboterfußball.

Unter Fortsetzung der Arbeiten von Rope (Role Oriented Programming Environment for Multiagent Systems) [BGKM 99, BKM 99, BML 97] ist das theoretische Konzept, welches in dieser Arbeit in Form des Interaktionsmodells vorgestellt wurde, zu einer Entwicklungsumgebung zu erweitern, welche unabhängig von der zugrundeliegenden Agentenarchitektur ist. Aufgrund der Verankerung im SFB 467 [SFB 99] steht zu erwarten, daß in absehbarer Zeit die erste Version verfügbar ist.

In dieser Arbeit wurde zu jeder Zeit der Bezug zu exemplarischen Anwendungen gesucht, um das Voranschreiten methodisch abzusichern. Die Anwendung im hinteren Teil der Arbeit konnte jedoch nur eine exemplarische Realisierung aufzeigen. Der wissenschaftliche Fortschritt dieser Arbeit liegt also nicht in der Evaluierung und partiellen Verbesserung einer bekannten Technologie, sondern ist im Grundlagenbereich anzusiedeln, welcher sich von den konkreten Fragestellungen der Anwendungsbereiche freizumachen versucht. Die prinzipielle Anwendbarkeit wurde mehrfach aufgezeigt. In den genannten Projekten wird sich zeigen, ob das Konzept mit entsprechender Werkzeugunterstützung Bestand hat.

# Literaturverzeichnis


[ABL+ 90] Alberts, Bruce; Bray, Dennis; Lewis, Julian; Raff, Martin; Roberts, Keith; Watson, James D.: Molekularbiologie der Zelle. 2. Weinheim; Basel; Cambridge; New York: VCH, 1990

[ABL+ 98] Avroutine, Viktor; Becht, Michael;Lafrenz, Reinhard; Levi, Paul; Muscholl, Matthias; Schanz, Michael: CoMRoS – Eine Multi-Agenten Architektur für intelligente Robotersysteme. In: Online-Publikation des 2. Workshop Kognitive Robotik: Von Sensordaten zu Information – Von Plänen zu Verhaltensmustern auf der 22. Jahrestagung Künstliche Intelligenz (KI-98), 15.-17. September, Bremen / Jana Köhler, Joachim Hertzberg, Thomas Röfer (eds)., 1998, (URL: http://www.informatik.uni-stuttgart.de/ipvr/bv/personen/lafrenz/ki-98/index.html)

[ArBö 83] Armbruster, Walter; Böge, Werner: Efficient, anonymous and neutral group decision procedures. In: Econoomica 51 (1983) Nr. 5, S. 1389-1405

[Bal 96] Balzert, Helmut: Lehrbuch der Software-Technik : Software-Entwicklung. Heidelberg; Berlin; Oxford: Spektrum, Akad. Verl., 1996

[Bal 98] Balzert, Helmut: Lehrbuch der Software-Technik : Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung. Heidelberg; Berlin: Spektrum, Akad. Verl., 1998

[Bec 97] Becht, Michael: Materialübergabe zwischen zwei fahrenden Robotern. Stuttgart, Univ., Diplomarbeit 1447, 1997

[BGKM 99] Becht, Michael; Gurzki, Torsten; Klarmann, Jürgen; Muscholl, Matthias: ROPE: Role oriented Programming Environment for Multiagent Systems. To appear in: Proceedings of the Fourth International Conference on Cooperative Information Systems (CoopIS'99), 2.-4. September 1999, Edinburgh, Scotland

[BKM 99] Becht, Michael; Klarmann, Jürgen; Muscholl, Matthias: ROPE: Role oriented Programming Environment for Multiagent Systems. In: Proceedings of the Third Annual Conference on Autonomous Agents (Agents 99), Technical abstracts of the software demonstrations, 1.-5. Mai, Seattle, WA, USA 1999

[BML 97] Becht, Michael; Muscholl, Matthias; Levi, Paul: Ein Framework für Kooperationsverfahren zwischen Roboteragenten. In: Autonome Mobile System , Tagungsband des 13. Fachgesprächs, Oktober, Stuttgart / P. Levi, Th. Bräunl und N. Oswald. Berlin u. a.: Springer, 1997, S. 166-177

[BML 98] Becht, Michael; Muscholl, Matthias; Levi, Paul: Transformable Multi-Agent Systems: A Specification Language for Cooperation Processes. In: Proceedings of the World Automation Congress (WAC '98), Sixth International Symposium on Manufacturing with Applications (ISOMA '98), 10.-14. Mai 1998, Anchorage, Alaska, USA/ Jamshidi, Mohammad et al.. Albuqerque: TSI Press, 1998, S. ISOMA-007.1–6

[BMS 96] Becht, Michael; Maisch, Oliver; ShojaZadeh, Hormoz: Klärung der Anforderungen einer flexiblen Fertigungsumgebung und Entwurf eines agentenorientierten Auftragsplanungssystems. Stuttgart, Univ., Studienarbeit Nr. 1502, 1996

[BoGa 88] Bond, A.H.; Gasser, L.: An analysis of problems and research in DAI. In: Readings in Distributed Artificial Intelligence/ A.H. Bond und L. Gasser (editors). San Mateo, USA: Morgan Kaufmann, 1988, S. 3-35

[BoVa 96] Boman, Magnus; Van de Velde, Walter (Hrsg.): Agents Breaking Away: 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW '96, Eindhoven, The Netherlands, 22.-25. Januar. Berlin: Springer, 1996

[Bus 96] Buschmann, Frank; et al.: Pattern-oriented software architecture: a system of patterns. 1. Chichester: Wiley, 1996

[Chi 98] Chiariglione, L.: Foundation for Intelligent Physical Agents, V. 0.2. Dublin, Juli 1998 (URL: http://drogo.cselt.stet.it/fipa/spec/fipa98/fipa98.htm)

[ChMa 84] Chang, J.; Maxemchuck, M.: Reliable Multicast Protocols. In: ACM Transactions on Computer Systems 2(1984) Nr. 3, S. 251–273

[CoLe 90] Cohen, P.R.; Levesque, H.J.: Intention is choice with commitment. In: Journal of Artificial Intelligence 42 (1990) Nr. 3, S. 213-261

[DaEd 94] Davies, Winton H. E.; Edwards, Peter: Agent-K: An Integration of AOP and KQML. In: Proceedings of the CIKM'94 Workshop on Intelligent Agents, Dez. 1994, Gaithersburg, Maryland/ T. Finin and Y. Labrou. 1994 (http://www.csd.abdn.ac.uk/~pedwards/publs/agentk.html)

[DeMü 89] Demazeau, Yves; Müller, Jean-Pierre(Hrsg.): Decentralized A. I. : *proceedings of the first European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'89)*; Cambridge, England, 16.-18.Aug. 1989. Amsterdam: North Holland, 1990

[Dur 99] Durfee, Edmund H.: Distributed Problem Solving and Planning. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence / Weiss, Gerhard (Hrsg.). Cambridge (Ma.), London: The MIT Press, 1999, S. 121-164

[EeSi 98] Eeles, Peter; Sims, Oliver: Building Business Objects. 1. Chichester: Wiley, 1998

[Feh 92] Fehling, Rainer.: Hierarchische Petrinetze. Hamburg: Kovacs, 1992. Zugl. Dortmund, Univ., Diss., 1991

[FGG+98] Franke, U.; Gavrila, D.; Görzig, S.; Linder, F.; Paetzold, F.; Wöhler, C.: Autonomous Driving Goes Downtown. In: IEEE Intelligent Systems & their applications 13 (1998) Nr. 6, S. 40-48

[FLM 97] Finin, Tim; Labrou, Yannis; Mayfield, James: KQML as an Agent Communication Language. In: Software Agents/ Bradshaw, Jeffrey M. (ed.). Melano Park, Californien, USA: AAAI Press / The MIT Press, 1997, S. 291-316

[FuTi 91] Fundenberg, Drew; Tirole, Jean: Game Theory. 1. Cambridge, MA, London, England: MIT Press, 1991

[GöFr 98] Görzig, S.; Franke, S.: ANTS – Intelligent Vision In Urban Traffic. In: Proc. of 1998 IEEE International Conference on Intelligent Vehicles, 28. - 30. Oktober, Stuttgart, 1998, S. 545-549 (http://www.daimler-benz.com/research/events/pdf/IV980210.PDF)

[GHJV 94] Gamma, Erich; Helm, Richard, Johnson, Ralph, Vlissides, John: Design Patterns: Elements of Reusable Object-Oriented Software. 1. Erscheinungsort: Addison-Wesley Pub., 1994

[Hak 83a] Haken, H.: Synergetics - An introduction. Berlin: Springer, 1983

[Hak 83b] Haken, H.: Advanced Synergetics. Berlin: Springer, 1983

[HaLe 92] Handel, Stefan; Levi, Paul: Restriktionsbasiertes Verhandlungskonzept für eine dezentrale, kooperative Aktionsplanung. In: Proc. der 8. Fachgespräche Autonome Mobile Systeme, November 1992, Karlsruhe / U. Rembold u. a. (Hrsg.). Karlsruhe: Universität Karlsruhe, 1992, S. 93-105

[HaMo 90] Halpern, Joseph Y.; Moses, Yoram: Knowledge and Common Knowledge in a Distributed Environment. In: Journal of the Association for Computing Machinery 37 (1990) Nr. 3, S. 549–587

[Han 95] Hahndel, Stefan: Ein verteiltes, verhandlungsgesteuertes Planungsverfahren für flexible Fertigungsumgebungen. München, Technische Univ., Diss., 1995

[Hew 77] Hewitt, C.: Viewing control structures as patterns of passing messages. In: Journal of Artificial Intelligence 8 (1977), S. 323-364

[HSK 99] Hayden, S.C.; Carrick, C. Yang, Q.: A Catalog of Agent Coordination Patterns. In: Proceedings of the Third Annual Conference on Autonomous Agents (Agents 99), 1.-5. Mai, Seattle, WA, USA, 1999, S. 412-413

[HuPi 91] Huber, P.; Pinci, V. O.: A Formal Executable Specification of the ISDN Basic Rate Interface. In: 12. International Conference on Applications and Theory of Petri Nets, Aarhus, 1991, S. 1-21

[HuSt 99] Huhns, Michael N.; Stephens, Larry M.: Multiagent Systems and Societies of Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence / Weiss, Gerhard (Hrsg.). Cambridge (Ma.), London: The MIT Press, 1999, S. 79-120

[Jen 96] Jennings, Nicholas R.: Coordination Techniques for distributed Artificial Intelligence. In: Foundation of Distributed Artificial Intelligence / G. M. O. O'Hare und N. R. Jennings (eds.). New York, USA: John Wiley & Sons, 1996, S. 187-210

[JSW 98] Jennings, Nicholas R.; Sycara, Katia; Wooldridge, Michael: A Roadmap of Agent Research and Development. In: Autonomous Agents and Multi-Agent Systems 1(1998) Nr. 1, S. 7–38

[Ken 98] Kendall, Elizabeth.: Agent Roles and Role Models: New Abstraction for MAS Analysis and Design. In: Online-Publikation des Interdisziplinären Workshops über Intelligente Agenten im Informations- und Prozeßmanagement auf der 22. Jahrestagung Künstliche Intelligenz (KI-98), 15.-17. September, Bremen / Christoph Klauck (ed.), 1998, (URL: http://www.tzi.de/tzi/tw-98/ki-98/workshops/KI98WS/papersD.html)

[KBM 98] Klarmann, Jürgen; Becht, Michael; Muscholl, Matthias: Modellierung flexibler Workflows mit teilausführbaren Aktivitäten. In: Workshop im Rahmen der DCSCW'98 "Flexibilität und Kooperation in Workflow-Management-Systemen", September, Dortmund / Bericht 18/98-I, 1998

[Kin 98] Kinny, David: The Agentis Agent Interaction Model. In: Intelligent Agents V: Agent Theories, Architectures, and Languages - 5th International Workshop, ATAL'98, 4.-7. Juli, Paris / Jörg P. Müller, Munindar P. Singh, Anand S. Rao (editors). Berlin: Springer, 1999, S. 331-344

[KLR+ 92] Kinny, D.; Ljungberg, M.; Rao, A. S.; Sonenberg, E.; Tidhar, G.; Werner, E: Planned team activity. In: Artificial Social Systems - Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds, MAAMAW-92 (LNAI Volume 830) / C. Castelfranchi und E. Werner (editors). Heidelberg: Springer, 1992, S. 226-256

[KrPo 88] Krasner, Glenn E.; Pope, Stephen: A cookbook for using the model-view-controller user interface paradigma in Smalltalk-80. In: Journal of Object-Oriented Programming 1 (1988) Nr. 3, S. 26-49

[LBLM 98] Levi, Paul; Becht, Michael; Lafrenz, Reinhard; Muscholl, Matthias: COMROS - A Multi-Agent Robot Architecture. In: 4th International Symposium on Distributed Autonomous Robotic Systems (DARS 4), 25.-27. Mai, Karlsruhe / R. Dillmann et al.. Berlin u. a.: Springer, 1998

[LBMR 94] Levi, Paul; Bräunl, Thomas; Muscholl, Matthias; Rausch, Alexander: Architektur und Ziele der Kooperativen Mobilen Robotersysteme Stuttgart. In: Autonome Mobile Systeme, Tagungsband des 10. Fachgesprächs, 13.-14. Oktober, Stuttgart / P. Levi und Th. Bräunl. Berlin: Springer, 1994, S. 262-273

[Les 95] Lesser, Victor (Hrsg.): First International Conference on Multi-Agent Systems, San Francisco, 12.-14. Juni. Menlo Park, Cambridge, London: AAAI Press / The MIT Press, 1995

[Les 98] Lesser, Victor R.: Reflections on the Nature of Multi-Agent Coordination and Its Implications for an Agent Architecture. In: Autonomous Agents and Multi-Agent Systems 1(1998) Nr. 1, S. 89–111

[Lev 89] Levi, Paul: Architectures of individual and distributed autonomous agents. In: Proc. of the Internalional Conference of Intelligent Autonomous System IAS-2, Dezember, Amsterdam, 1989, S. 315-324

[Lev 92] Levi, Paul: Architecture and Restriction-based Planning of Autonomous Traffic Agents. In: Journal for the Integrated Study of Artificial Intelligence Cognitive Science and Applied Epistemology (CC AI) 1992 (9) Nr. 1, S. 43-64

[Lev 96] Levi, Paul: *Robotik*. In: Wörterbuch der Kognitionswissenschaften/ hrsg. von Gerhard Strube et al.. Stuttgart: Klett-Cotta, 1996

[LMB 95] Levi, Paul; Muscholl, Matthias; Bräunl, Thomas: Cooperative Mobile Robots Stuttgart: Architecture and Tasks. In: Intelligent Autonomous Systems (IAS-4), Karlsruhe / U. Rembold, R. Dillmann, L.O. Hertzberger, and T. Kanade. Amsterdam, Oxford, Washington DC: IOS Press, 1995, S. 310-317

[LSA 98] Levi, Paul; Schanz, Michael; Avroutine, Viktor: Chaos-theory-based analysis of manufacturing. In: Proc. of the Internalional Conference of Intelligent Autonomous System IAS-5, 1.-4. Juni, Sapporo, Japan, 1998, S. 564-571

[LSAL 99] Levi, Paul; Schanz, Michael; Avroutine, Viktor; Lammert, Robert: Modeling and stability analysis of the dynamic behavior in load sharing systems. To appear in: Proceedings of the Int. Seminar on Modelling and Planning for Sensor-Based Intelligent Roboter Systems / H. Christensen (ed.). Berlin: Springer, 1999

[Mül 93] Müller, Jürgen (Hrsg.): Verteilte Künstliche Intelligenz: Methoden und Anwendungen. 1. Mannheim: BI Wissenschaftsverlag, 1993

[MuLe 95] Muscholl, Matthias; Levi, Paul: Modelling Co-operation of Autonomous Agents by Petri Nets. In: 28th ISATA, Mechatronics - Efficient Computer Support for Engineering, Manufacturing, Testing & Reliability, September, Böblingen / John I. Soliman und Dieter Roller. Croydon: Automotive Automation Limited, 1995, S. 255-262

[MuLe 96] Muscholl, Matthias; Levi, Paul: Entscheidungsnetzwerke für selbstorganisierende Roboterarchitekturen. In: Autonome Mobile Systeme, Tagungsband des 12. Fachgesprächs, Oktober, München / G. Schmidt and F. Freiberger. Berlin: Springer, 1996, S. 236-245

[MoPi 94] Mortensen, K. H.; Pinci, V. O.: Modelling the Work Flow of a Nuclear Waste Management Program. In: Application and Theory of Petri Nets 1994: Proceedings of the15. International Petri Net Conference, 1994, Zaragoza / R. Valette (ed.). Berlin: Springer, 1994, S. 376-395

[NIIP 98] NIIP Reference Architecture. Dezember 1998. (URL: http://www.niiip.org/public-forum/index-ref-arch.html)

[OsLe 97] Oswald, Norbert; Levi, Paul: Cooperative Vision in a Multi-Agent Architecture. In: 9th Int. Conf. on Image Analysis and Processing (ICIAP) 1997 / A. Del Bimbo. Lectures Notes in Computer Science, LNCS 1310, Berlin u. a.: Springer, 1997, S. 709-716

[Ont 99] Ontolingua Server. August 1999. (URL: http://ontolingua.stanford.edu)

[Par 99] Parunak, H. Van Dyke: Industrial and Practical Applications of DAI. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence / Weiss, Gerhard (Hrsg.). Cambridge (Ma.), London: The MIT Press, 1999, S. 377-421

[Rau 98] Rausch, Willi Alexander: Interaktion autonomer Roboter: Kooperative Navigation autonomer Roboterfahrzeuge unter Einsatz von Funk und Ultraschall. Aachen: Shaker, 1998. Zugl. Stuttgart, Univ., Diss., 1998

[Red 96] Redlich, Jens-Peter: CORBA 2.0: Praktische Einführung für C++ und Java. 1. Bonn; Reading, Mass.: Addison-Wesley, 1996 (Praktische Informatik)

[Rei86] Reisig, Wolfgang: Petrinetze: eine Einführung. 2. Auflage. Berlin u. a.: Springer, 1986 (Studienreihe Informatik)

[ROL 95] Rausch, Alexander; Oswald, Norbert; Levi, Paul: Cooperative Crossing of Traffic Intersection in a Distributed Robot System. In: Sensor Fusion and Network Robotics VIII (SPIE), 22.-27. Oktober, Philadelphia, Pennsylvania, USA / Schenker, P.S.; McKee, G.T. Bellingham, WA, USA: SPIE-Press, 1995, S. 218-229

[Ros 85] Rosenschein, J.S.: Rational Interaction: Cooperation Among Intelligent Agents. Stanford, Univ., Diss., 1985

[RoZl 94] Rosenschein, J. S.; Zlotkin, G.: Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers. 1. Boston, MA, USA: MIT Press, 1994

[Roth 97] Rothermel, Kurt (Hrsg.): Mobile Agents: First International Workshop, MA'97, Berlin, 7.-8. April. Berlin: Springer, 1997 (Lecture notes in computer science; 1219)

[Roth 98] Rothermel, Kurt (Hrsg.): Mobile Agents: Second International Workshop, MA'98, Stuttgart, 9.-11. September. Berlin: Springer, 1998 (Lecture notes in computer science; 1477)

[Rüd 92] Rüdebusch, Tom: CSCW: generische Unterstützung von Teamarbeit in verteilten DV-Systemen. Wiesbaden: Deutscher Universitäts-Verlag, 1992. Zugl. Karlsruhe, Technische Universität., Diss., 1992

[SaGa 96] Shaw, Mary; Garlan, David: Software Architecture : Perspectives on an Emerging Discipline. 1. New Jersey: Prentice Hall, 1996

[Sar 93] Saraswat, Vijay A.: Concurrent Constraint Programming. 1. Cambridge, MA, London England: MIT Press, 1993

[Schu 99] Schuh, Hans: Freie Fahrt in der Kolonne. In: DIE ZEIT 1999 (99) Nr. 25, S. 27-28 (http://www.ZEIT.de/tag/aktuell/199925.deichsel_.html)

[Sen 99] Sengers, Phoebe: Designing Comprehensible Agents. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 31.7.-6.8., Stockholm, Schweden/ Carl Gustaf Jansson et al.. San Francisco, USA: Morgan Kaufmann Pub., 1999, S. 1227-1232

[SFB 99] Sonderforschungsbereich 467 an der Universität Stuttgart: Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion. Stuttgart, 1999 – Arbeits- und Ergebnisberichte Januar 1997 - Dezember 1999

[Sha 91] Shapiro, R. M.: Validation of a VLSI Chip Using Hierarchical Coloured Petri Nets. In: High-level Petri Nets. Theory and Application/ Jensen, K.; Rozenberg, G. (Hrsg). Berlin u. a.: Springer, 1991, S. 667-687

[Sho 89] Shoham, Yoav: Time for Action. In: Proceedings of the Eleventh International Joint Conference in Artificial Intelligence, August, Detroit, Michigan. USA, 1989, S. 954-959

[Sho 93] Shoham. Yoav: Agent-oriented programming. In: Journal of Artificial Intelligence 60 (1993) Nr. 1, S. 51- 92

[Sho 97] Shoham. Yoav: An Overview of Agent-Oriented Programming. In: Software Agents/ edited by Jeffrey M. Brandshaw. Menlo Park, Calif. USA: AAAI Press / The MIT Press, 1997, S. 271-290

[Smi 80] Smith, Reid G.: The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver. In: IEEE Transaction on Computers C-29 (1980) Nr. 12, S. 1104-1113

[Smi 96] Smith, Einar: A Survey on High-Level Petri-Net Theory. In: Bulletin of the European Association for Theoretical Computer Science (EATCS) Juni 1996 (96) Nr. 59, S. 267-293

[SP 88] Silberschatz, Abraham; Peterson, James L.: Operating System Concepts. 2. Reading MA u. a.: Addison-Wessley Pub., 1988

[SRG 99] Singh, Munindar P.; Rao, Anand S.; Georgeff, Micael P.: Formal Methods in DAI: Logic-Based Representation and Reasoning. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence / Weiss, Gerhard (Hrsg.). Cambridge (Ma.), London: The MIT Press, 1999, S. 331-376

[Sta 80] Starke, Peter H.: Petri-Netze. Berlin: VEB Deutscher Verlag der Wissenschaften, 1980

[Sta 90] Starke, Peter H.: Analyse Von Petri-netz-modellen. Stuttgart: Teubner, 1990 (Leitfäden Und Monographien Der Informatik)

[Ste 93] Steinmann, Horst, Schreyögg, Georg: Management: Grundlagen derUnternehmensführung: Konzepte – Funktionen – Fallstudien. 3. Wiesbaden: Gabler, 1993

[StVe 98] Stone, P.; Veloso, M.: Task Decomposition and Dynamic Role Assignment for Real-Time Strategic Teamwork. In: Intelligent Agents V: 5. Workshop on Agent Theories, Architectures, and Languages, 4.-7. Juli, 1998, Paris, Frankreich/ Jörg. P. Müller, Munindar P. Singh, Anand S. Rao (eds.). Berlin: Springer, 1999, S. 293-308

[Sun 94] Sundermeyer, Kurt: VKI: Verteilte Künstliche Intelligenz. In: 12. Frühjahrschule Künstliche Intelligenz (KIFS) 1994, 26.2.-5.3.1994, Günne / Möhnesee / Sundermeyer, Kurt. Daimler-Benz AG: Forschung Systemtechnik Berlin, 1994

[Tan 90] Tanenbaum, Andrew S.: Computer-Netzwerke. 1. Wolfram's Fachverlag, 1990

[WBW 98] Westkämper, E.; Balve, P.; Wiendahl, H.-H.: Auftragsmanagement in wandlungsfähigen Unternehemensstrukturen. In: PPS-Management 3 (1998) Nr. 1, S. 22-26

[Wei 99] Weiss, Gerhard (Hrsg.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. 1. Cambridge (Ma.), London: The MIT Press, 1999

[Win 93] Winklhofer, Andreas: Zeitrepräsentation und merkmalsgesteuerte Suche zur Terminplanung. Sankt Augustin: Infix, 1993. Zugl. München, Techn. Univ., Diss., 1993

[WJK 99] Wooldridge, M.; Jennings, N. R.; Kinny, D.: A Methodology for Agent-Oriented Analysis and Design. In: Proceedings of the Third Annual Conference on Autonomous Agents (Agents 99), 1.-5. Mai, Seattle, WA, USA, 1999, S. 69-76

[WoJe 94] Wooldridge, Michael; Jennings, Nicholas R. (Hrsg.): Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages, Amsterdam, 8.-9. August. Berlin: Springer, 1994 (Lecture Notes in Computer Science; 890)

[Woo 99] Wooldridge, Michael: Intelligent Agents. In: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence / Weiss, Gerhard (Hrsg.). Cambridge (Ma.), London: The MIT Press, 1999, S. 27-77