Bayerische Julius-Maximilians-Universität
Institut für Informatik
Lehrstuhl für Künstliche Intelligenz und
Angewandte Informatik (VI)
Am Hubland, Würzburg

Diplomarbeit

# Holonische Multiagentensimulation

Bearbeiter
Steffen Glückselig

Betreuer
Prof. Dr. Frank Puppe
Dipl.-Inform. Manuel Fehler

vorgelegt am
15. März 2005

# Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Würzburg, den 15. März 2005

# Inhaltsverzeichnis

1. Einleitung ..... 9
1.1. Motivation ..... 9
1.2. Ziel und Struktur der Arbeit ..... 10
2. Grundlagen ..... 13
2.1. Agenten ..... 13
2.2. Multiagentensysteme ..... 16
2.3. Multiagentensimulationen ..... 21
3. Theorie holonischer Systeme ..... 25
3.1. Holone ..... 25
3.2. Intraholonische Strukturen ..... 30
3.3. Gründung eines Holonen und Bestimmung des Kopfes ..... 43
3.4. Aufgabenteilung ..... 46
3.5. Beenden oder Verlassen eines Holonen ..... 51
3.6. Eigenschaften holonischer Systeme ..... 53
3.7. Anwendungsbeispiele von Holonen ..... 55
4. Realisierung ..... 65
4.1. Vorgehensmodell für Analyse und Entwurf ..... 65
4.2. Anforderungen an die Implementierung ..... 75
4.3. Das Holonic Augmentation Plugin ..... 77
4.4. Modellierung eines holonischen Systems ..... 85
5. Beispielanalyse und Diskussion ..... 95
5.1. Effizienz von holonischen Systemen ..... 95
5.2. Entwicklung von holonischen Systemen ..... 106
6. Zusammenfassung und Ausblick ..... 109
6.1. Zusammenfassung ..... 109
6.2. Ausblick ..... 110
Datenstrukturen und Primitive ..... 113
A.1. Datentypen ..... 113
A.2. Primitive des Headholon-Features ..... 114
A.3. Primitive des Subholon-Features ..... 118
Danksagung ..... 125
Abkürzungsverzeichnis ..... 127
Literaturverzeichnis ..... 131

# Abbildungsverzeichnis

2.1. Ein Agent und seine Umwelt ..... 14
3.1. Grafische Darstellung einer Holarchie ..... 26
3.2. Hierarchie und Heterarchie im Vergleich ..... 27
3.3. Die holonische Struktur als UML-Diagramm ..... 31
3.4. Eine Holarchie mit Kopfholonen als Repräsentanten ..... 32
3.5. Kopfholone kommunizieren ..... 33
3.6. Ein Holon bestehend aus autonomen Subholonen ..... 34
3.7. Ameisen beim Bau einer Brücke ..... 35
3.8. Ein Holon, der aus verschmolzenen Subholonen besteht. ..... 36
3.9. Ein Holon als moderierte Gruppe ..... 37
3.10. Fünf unabhängige Agenten ..... 38
3.11. Organisationsform: Markt ..... 38
3.12. Organisationsform: Virtuelles Unternehmen ..... 39
3.13. Organisationsform: Allianz ..... 40
3.14. Organisationsform: Strategisches Netzwerk ..... 41
3.15. Organisationsform: Gruppe ..... 42
3.16. Organisationsform: Gesellschaft ..... 42
3.17. Formelles and informelles Rollenspiel ..... 43
3.18. Taskverteilung in einer Hierarchie ..... 47
3.19. Das Contract Net Protocol nach FIPA-Spezifikation ..... 49
3.20. The Extended Contract Net Protocol ..... 51
3.21. Die Architektur des TeleTruck-Systems ..... 58
3.22. Planen in TeleTruck ..... 59
3.23. Agenten als Erweiterung von Objekten ..... 63
4.1. Die SODA-Methodologie im Überblick ..... 69
4.2. Zielgraph für einen Teil der TeleTruck-Domäne ..... 70
4.3. Ausführlicher Zielgraph der TeleTruck-Domäne ..... 72
4.4. Auswahldialog für Holonentypen ..... 78
4.5. Spezifizieren von Interaktionsregeln im HolonTypEditor ..... 79
4.6. Angeben von Fähigkeiten im HolonTypeEditor ..... 80
4.7. Das *Holonic abstraction level*-Panel ..... 81
4.8. Das Headholon-Panel ..... 81
4.9. Zuweisung von Fähigkeiten an eine Agentenklasse in SeSAm ..... 82
4.10. Das Subholon-Panel ..... 83
4.11. Der Holarchie-Browser ..... 85
4.12. Beispiel für eine Semaphore ..... 87

4.13. Beispiel für die Verwendung von IsMemberOfHolon ..... 88
4.14. Kommunikationsvorgang: Subholon bewirbt sich für eine Rolle. ..... 92
4.15. Kommunikationsvorgang: Kopfholon bietet eine Rolle an. ..... 93
5.1. Die Monitoring-Welt ..... 97
5.2. Ergebnisgraphik für VÜ_ohneH ..... 100
5.3. Ergebnisgraphik für VÜ_mitH_1 ..... 101
5.4. Ergebnisgraphik für VÜ_mitH_2 ..... 102
A.1. Auswahl eines Skills für den ComposedType *Competency*. ..... 119
A.2. Verwendung der Semaphore ..... 123

# Tabellenverzeichnis

3.1. Vergleich zwischen komplexen und agentenbasierten Systemen ..... 64
5.1. Parameter aller Modelle ..... 99
5.2. Parameter der holonischen Modelle ..... 99
5.3. Ergebnisse der Evaluation zusammengefaßt ..... 102

# 1 Einleitung

## 1.1 Motivation

*Computer science is the first engineering discipline ever in which the complexity of the objects created is limited by the skill of the creator and not limited by the strength of the raw materials.*
--Brian K. Reid

Die Entwicklung und Wartung komplexer Systeme ist eine der größten Herausforderungen für den Menschen im Bereich der Technik. Besonders Softwaresysteme werden von vielen als die komplexesten vom Menschen geschaffenen Systeme betrachtet. Sie erfordern ein tiefes Verständnis des Problems und des zur Lösung des Problems zu entwickelnden Systems.

Die Erforschung von Techniken zur Realisierung komplexer (Software-) Systeme wird mit hohem Aufwand betrieben. Ergebnisse dieser Forschung sind Techniken wie beispielsweise das prozedurale Programmieren, das objektorientierte Programmieren, das komponentenbasierte Programmieren und seit kurzem das agentenorientierte Programmieren. All diese Techniken sind entworfen worden, um besser mit komplexen Systemen umgehen zu können.

Mit der anwachsenden Vernetzung von Rechenressourcen, beispielsweise durch das Internet, wird das *verteilte* Problemlösen zunehmend interessanter. Agenten, ein Konzept aus der Forschung im Bereich der Künstlichen Intelligenz, eignen sich ausgesprochen gut für den Einsatz als kooperative und Organisationen bildende Problemlöser in solchen verteilten Systemen.

Die in dieser Arbeit beschriebenen Holonen sind eine Möglichkeit, mehrere Agenten zusammenfassend zu betrachten. Sie haben damit eine ähnliche Funktion wie die Komponenten im komponentenbasierten Programmieren und erlauben die abstrahierende Betrachtung einer Menge miteinander kooperierender Agenten.

In den 1990er Jahren rückte das Holonenkonzept in das Blickfeld der Forschung und wurde eingehend für einen Einsatz im Bereich der intelligenten Manufaktursysteme (*intelligent manufacturing systems*) (IMS) untersucht. Der Teilbereich der IMS, der sich mit Holonen beschäftigt, ist das Gebiet der holonischen Manufaktursysteme (*holonic manufacturing systems*) (HMS) (siehe [Vo04]), in denen Holonen wegen ihrer Widerstandsfähigkeit gegenüber Systemstörungen

(durch Selbstorganisation) und ihrer Anpassung an Änderungen in ihrer Umgebung (durch Autonomie) zum Einsatz kommen. Dadurch soll eine hohe Produktivität in einer dynamischen Umgebung gewährleistet werden. Beispielsweise soll die Umstellung eines Herstellungsprozesses möglichst schnell erfolgen können, um auf kurzfristige Änderungen der Anforderungen des Marktes reagieren zu können.

TeleTruck ist ein holonisches Multiagentensystem, das im Speditionswesen eingesetzt wird. Durch den Einsatz von Holonen werden die Auftragsvergabe und die Zusammenstellung von Lastwagengespannen optimiert. Das System war auch im Vergleich mit Lösungen aus der Operations Research sehr erfolgreich. Es wird in Abschnitt 3.7.1 näher beschrieben.

Die positiven Resultate im Bereich der holonischen Manufaktursysteme (siehe unter anderem Bussmann und McFarlane [BF99]) und Multiagentensysteme (TeleTruck) zeigen das hohe Potential der Holonen auf.

Bevor solche Systeme in der Realität eingesetzt werden, unterlaufen sie verschiedenen Tests, um Fehler zu finden, diese zu entfernen und das System zu verbessern. Für diese Tests werden u.a. Simulationen verwendet, in denen das in einem Modell nachgebildete System in einer virtuellen Umgebung zum Einsatz kommt. So können kostengünstig Fehler gefunden und Paramtereinstellungen optimiert werden.

Auch in Multiagentensimulationen, die nicht zum Entwickeln von realen Systemen, sondern beispielsweise zum Testen einer Hypothese erstellt werden, können Holonen hilfreich sein. Durch die abstrahierende Betrachtung mehrerer Agenten erlauben sie den Überblick über das zu beobachtende System zu behalten und Modelle von einer allgemeineren Ebene ausgehend zu erstellen.

## 1.2 Ziel und Struktur der Arbeit

*Keep simple things simple, make complex things possible.*
--Unbekannt

Mit der vorliegenden Arbeit soll das Holonenkonzept vorgestellt und die Möglichkeit der Modellierung holonischer System in SeSAm realisiert werden. Um die Modellierung von Holonen in SeSAm zu unterstützen, muss ein Vorgehensmodell für Analyse und Entwurf eines holonischen Systems entwickelt werden,

das sich auf die vorgestellten theoretischen Grundlagen stützt. Auf die Ergebnisse des Vorgehensmodells soll in der Modellierung nahtlos aufgesetzen werden können. Während der Modellierung sollen dem Entwickler Werkzeuge zur Verfügung stehen, die ihm seine Arbeit erleichtern. Schließlich soll es dem Entwickler möglich sein, das holonische Modell zur Laufzeit zu untersuchen, um Fehler und Verbesserungsmöglichkeiten entdecken zu können.

**Kapitel 2**

Beschreibt die Grundlagen, auf die im Fortgang der Arbeit zurückgegriffen wird: Agenten, Multiagentensysteme und Multiagentensimulationen

**Kapitel 3**

Beschreibt die theoretischen Grundlagen holonischer Systeme. Der Begriff des Holonen und weitere im holonischen Umfeld gebräuchliche Begriffe werden erläutert. Es werden die Eigenschaften eines Holonen und seine internen Strukturen vorgestellt. Es wird geklärt, wie ein Holon entsteht, wie seine Teile einander Aufgaben zuweisen und wie ein Holon beendet werden kann. Die Eigenschaften einer miteinander interagierenden Menge von Holonen werden aufgeführt. Schließlich werden zwei exemplarische Einsatzgebiete von Holonen vorgestellt.

**Kapitel 4**

Das Vorgehensmodell zu Analyse und Entwurf holonischer Systeme wird erarbeitet und anhand eines Beispiels erläutert. Aus den theoretischen Grundlagen und dem Vorgehensmodell werden die Anforderungen an die Implementierung ermittelt. Danach wird die Implementierung mit ihren Fähigkeiten vorgestellt und die Modellierung eines holonischen Systems mit einem Beispiel verdeutlicht.

**Kapitel 5**

Die Implementierung wurde anhand von zwei Gesichtspunkten evaluiert: (i) "Wie effizient ist ein Modell mit Holonen im Vergleich zu einem Modell mit normalen Agenten?" und (ii) "Wie gut wird der Modellierer beim Entwickeln holonischer Modelle unterstützt?"

**Kapitel 6**

Die Ergebnisse der Arbeit werden zusammengefasst und ein Ausblick auf Ausbau- und Verbesserungsmöglichkeiten der holonischen Erweiterung von SeSAm wird gegeben.

**Anhang A**

Fasst die in der holonischen Erweiterung von SeSAm bereitgestellten Datentypen und Primitive zusammen und erläutert ihre Anwendung.

# 2 Grundlagen

Bevor die Modellierung von Holonen in SeSAm in den nachfolgenden Kapiteln vorgestelt wird, sollen in diesem Kapitel allgemeine Grundlagen für das Verständnis geschaffen werden. Zuerst wird eine Einführung in Agenten gegeben. Desweiteren werden Multiagentensysteme und deren Simulation erläutert.

## 2.1 Agenten

Das Agentenkonzept ist in der Literatur unterschiedlich definiert (siehe hierzu u.a. [WJ95] und [FG96]). In dieser Arbeit wird folgende Definition von Wooldridge [WJ00] zugrunde gelegt, da sie (i) in der Wissenschaft weit verbreitet und anerkannt ist und (ii) die in der Definition verlangten Eigenschaften in SeSAm umsetzbar sind:

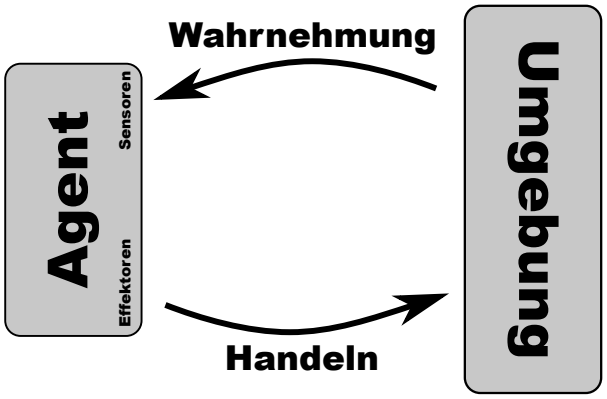
> Ein Agent ist ein gekapseltes Computersystem, das in einer Umgebung situiert ist und das in dieser Umgebung zu flexiblen autonomen Aktionen fähig ist, um seine Konstruktionsziele zu erreichen.

Aus dieser kurzen, informellen Charakterisierung lassen sich folgende drei Eigenschaften eines Agenten extrahieren, die das Konzept eines Agenten definieren:

1. **Autonomie:** Ein Agent besitzt Kontrolle über seinen internen Zustand und sein Verhalten. Er kann ohne das direkte Eingreifen von Menschen oder anderen autonomen Einheiten in seiner Umgebung agieren [WJ95].

2. **Situiertheit:** Ein Agent ist in seine Umgebung eingebettet. Er nimmt sie über Sensoren wahr und verändert sie durch Effektoren (siehe Abbildung 2.1).

3. **Flexibilität:** Ein Agent verändert sein Problemlöseverhalten und passt sich so seiner Umgebung an. Zum einen benötigt ein Agent die Fähigkeit zu **reaktivem Verhalten**, d.h. er muss fähig sein auf Veränderungen in der Umwelt schnell zu reagieren, damit er seine Konstruktionsziele erreichen kann. Zum anderen kann ein Agent auch **proaktiv** tätig werden. Dann nimmt




Ein Agent nimmt seine Umgebung durch Sensoren wahr und handelt in ihr mit Hilfe von Effektoren.

**Abbildung 2.1. Ein Agent und seine Umwelt**

er opportunistisch neue Ziele an und ergreift die Initiative um seine Konstruktionsziele zu erreichen, d.h. er zeigt *zielorientiertes Verhalten*.

Für gewöhnlich existieren Agenten jedoch nicht allein, sondern sie sind zusammen mit anderen Agenten in ein System eingebunden. Interaktion zwischen Agenten ist meist notwendig, weil die Problemlösefähigkeiten des Einzelnen beschränkt sind. Ein Agent, der *flexibel* mit anderen Agenten (und möglicherweise Menschen) interagieren können soll, benötigt daher zusätzlich zu den zuvor genannten Fähigkeiten auch **soziale Fähigkeiten**. Ein anspruchsvoller Aspekt dieser Fähigkeiten ist die *Kooperation*. Generell ist mit der Fähigkeit zur sozialen Interaktion die Fähigkeit zur *Kommunikation* verbunden. Systeme, die aus mehreren Agenten bestehen, werden in Abschnitt 2.2 näher beschrieben.

Auf die wichtigsten in der Definition genannten Begriffe soll in den nachfolgenden Abschnitten kurz detaillierter eingegangen werden [Ge99].

### 2.1.1 Autonomie

Es werden drei Arten von Autonomie unterschieden, welche in beliebigen Kombinationen und Qualitäten auftreten können: (1) *Zustandsautonomie*: der interne Zustand eines Agenten hängt nur vom vorherigen Zustand und den Wahrnehmungen des Agenten ab und kann nicht von außen ohne den Willen des Agenten geändert werden. (2) *Aktionsautonomie*: die Aktionen eines Agenten hängen nur vom gegenwärtigen Zustand des Agenten ab. (3) *Berechnungssauto-*

*nomie*: der Agent verfügt über eigene Rechenressourcen, oder besitzt in einer fairen Art und Weise Zugang zu Rechenressourcen.

### 2.1.2 Zielorientiertes Verhalten

Ein Agent besitzt eine implizite oder explizite Darstellung seiner Ziele und versucht diese durch *pro-aktives* [WJ95] Verhalten zu erreichen. Implizite Ziele können dabei vom Systementwickler vorgegeben sein. Explizite Ziele können in Datenstrukturen der Agenten-Architektur in einer Wissenssprache gespeichert sein. Zum Beispiel werden in der BDI-Architektur Ziele explizit durch Wünsche (*desires*) dargestellt [GR91].

### 2.1.3 Annahmen

Ein Agent besitzt explizite oder implizite Repräsentationen seiner Umgebung. Annahmen (*beliefs*) sind das sich ändernde Wissen über die Welt. Das Wissen über die Welt beinhaltet manchmal auch Wissen über andere Agenten (*Modell*). Ein Agent kann Wissen über andere Agenten und die Welt erwerben oder erweitern, indem er *schlussfolgert*.

### 2.1.4 Beschränkte Problemlösefähigkeit

"Beschränkte Problemlösefähigkeit" (*bounded rationality*) wurde 1957 von Herbert Simon [BR05] definiert. Der Begriff beschreibt die Eigenschaft eines Agenten, sich nahezu optimal in Anbetracht seiner Ziele und der ihm zur Verfügung stehenden Ressourcen zu verhalten. Die wichtigsten Ressourcen, die einem Agenten für gewöhnlich nur beschränkt zur Verfügung stehen sind CPU-Zeit und Speicher.

### 2.1.5 Kommunikation und Koordination

Um *soziale Interaktion* zu ermöglichen müssen Agenten in der Lage sein untereinander durch eine gemeinsame Sprache zu kommunizieren. Die Fähigkeit zur **Kommunikation** ist essentiell, um das Verhalten verschiedener Agenten zu koordinieren, lokal zugängliche Informationen gemeinsam zu nutzen und Annahmen an einander anzupassen. Für die direkte Kommunikation wird meist

eine Agentenkommunikationssprache (*agent-communication language*) (ACL), wie beispielsweise KQML (*Knowledge Query and Manipulation Language*), verwendet. Durch die Verwendung solcher Standards wird der Einsatz heterogener Agenten in einem offenen System ermöglicht.

In einem **offenen System** ist zur Entwicklungszeit nicht bekannt, welche Agenten das System bevölkern werden, da sie zur Laufzeit eingefügt werden können. Außerdem kann sich die Zusammensetzung des Systems mit der Zeit ändern und sehr heterogen sein. D.h. Agenten können von unterschiedlichen Leuten, mit unterschiedlichen Softwarewerkzeugen und Techniken implementiert worden sein [Sy98].

**Koordination** ist eine Eigenschaft eines Systems von interagierenden Agenten. Je höher die Koordination in einem (Teil-)System ist, desto weniger redundante Aktivitäten werden im System durch die Agenten ausgeführt. *Kooperation* ist Koordination in einer nicht-antagonistischen Umgebung. *Verhandlung* ist Koordination in einer wettkampforientierten Umgebung, in der die Agenten *eigennützig* sind. Um eine effiziente Koordination zu ermöglichen sollte ein Agent ein Modell der Agenten besitzen, mit denen er interagiert [We00].

Es werden zwei Arten der Koordination unterschieden: (1) **Implizite Koordination** ohne Kommunikation. Die Agenten können entweder so entworfen sein, dass das Verhalten zum Erreichen eines bestimmten Ziels aus dem Verhalten des einzelnen Agenten hervorgeht, oder einige Agenten können ein Modell der Ziele und Absichten der anderen Agenten erstellen. Dies wird dann dazu verwendet, das eigene Verhalten an das erwartete Verhalten der anderen Agenten anzupassen. (2) **Explizite Koordination**, bei der Kommunikation verwendet wird, um über gemeinsame Pläne und Ressourcenverteilungen zu verhandeln.

## 2.2 Multiagentensysteme

*Large systems are built in a distributed fashion in order to master complexity.*
--Nicholas R. Jennings

Ein System von miteinander interagierenden Agenten wird Multiagentensystem (MAS) genannt. Die Agenten nehmen im System die Rollen verteilter *Problemlöser* ein. Das Studium, die Konstruktion und Anwendung dieser Systeme ist das Interessenfeld der *Verteilten Künstlichen Intelligenz*, einem Zweig des Forschungsgebietes der *Künstlichen Intelligenz*.

Die Verwendung von Agenten als Bausteinen eines solchen Systems resultiert in der **Modularität** von MAS. Durch die Kapselung von Zuständen und Verhalten in einem Agenten wird eine **Abstraktion** des Systems erreicht. Diese beiden Konzepte sind der Schlüssel zum Umgang mit komplexen Systemen von großem Umfang, wie „Die Parabel von den Uhrmachern“ veranschaulicht:

**Die Parabel von den Uhrmachern.** In dieser Parabel geht es um zwei Uhrmacher, die sehr komplizierte Uhren herstellten. Einer baute seine Uhren so zusammen, dass sie sofort wieder auseinanderfielen, wenn er gestört wurde oder sie nicht in einer einzelnen Sitzung fertigstellen konnte. Der andere Uhrmacher baute seine Uhren so, dass er ausgehend von kleinen Modulen diese zu immer größeren zusammensetzte bis die Uhr fertig war. Der erste Uhrmacher wurde sehr arm, weil er auf seine Weise nur wenige Uhren herstellen konnte. [Wa02]

### 2.2.1 Eigenschaften von Multiagentensystemen

Der Einsatz von Multiagentensystemen ist nicht prinzipiell und bei allen Problemstellungen anzuraten. Manche Problemstellungen eignen sich besser für eine Bearbeitung mit Agenten als andere. Vor der Erläuterung der Eigenschaften solcher Problemstellungen werden die Charakteristiken eines MAS dargestellt, damit verständlich wird, warum MAS in den später beschriebenen Problemstellungen eine Lösung sind [Sy98].

- Jeder Agent besitzt **unvollständiges Wissen oder Fähigkeiten**, um Probleme zu lösen. Diese Beschränkungen zwingen die Agenten miteinander zu interagieren, um durch Kooperation das Problem zu lösen. Dies resultiert in Organisationsformen, wie sie in Abschnitt 3.2.4 beschrieben werden.

- Es gibt **keine globale Kontrolle**. Dies ermöglicht die Autonomie des einzelnen Agenten. Da der Agent über eigene Berechnungsressourcen verfügt, ist er in dieser Hinsicht unabhängig von anderen Agenten oder einer globalen Kontrolle.

- **Daten werden dezentral gespeichert**. Jeder Agent speichert seine Daten und seinen Zustand selbst und kann nur auf diese Information direkt zugreifen. Um sein Wissen zu erweitern muss ein Agent mit anderen Agenten kommunizieren oder mit der Umwelt interagieren (z.B. Experimente ausführen).

- **Berechnungen sind asynchron**. Es gibt keinen globalen Takt. Lokale Berechnungen und Aktionen der Agenten werden nicht synchronisiert und sind nebenläufig.

### 2.2.2 Eigenschaften von Problemstellungen

Die im vorherigen Abschnitt beschriebenen Charakteristiken erlauben es mit MAS u.a. folgende sechs Punkte zu leisten [Sy98]:

1. Probleme zu lösen, die so **groß** sind, dass man sie nicht mit einem zentralen Problemlöser bearbeiten kann, weil nur beschränkt Ressourcen zur Verfügung stehen oder das Risiko eines Engpasses oder eines Ausfalls in der zentralen Komponente zu groß ist. MAS bieten Flexibilität und Robustheit durch die Möglichkeit der Selbstreorganisation im Falle eines Ausfalls einer einzelnen Komponente. Die Möglichkeit der Selbstorganisation wird oft durch Redundanz hergestellt, d.h. durch mehrere Agenten mit denselben Fähigkeiten. Eine andere Möglichkeit besteht darin, Agenten mit unterschiedlichen Fähigkeiten geschickt zu kombinieren.

2. Die **Kopplung von mehreren existierenden Altsystemen**. Altsysteme (*legacy systems*) wurden meist monolithisch entworfen und bieten nicht die Fähigkeiten für umfassende Interaktionen zwischen ihnen. Anstatt solche Systeme um die erweiterte Fähigkeit der Interaktion neu zu programmieren, kann man sie in Agenten einbinden. Ein Agent umhüllt dann ein Altsystem und ermöglicht so die Verwertung der vom Altsystem angebotenen Dienste durch andere Agenten, die ebenfalls Altsysteme repräsentieren können (siehe z.B. [Je94]).

3. Lösungen für Probleme zu bieten, die auf natürliche Art auf eine **Gesellschaft von autonomen und interagierenden Agenten** abgebildet werden können. Beispiele eines solchen Szenarios sind Verkehrsmanagement und -kontrolle [Bu97] und Luftverkehrsüberwachung [NI01].

4. Lösungen zu Problemen zu bieten, bei denen Informationsquellen **räumlich verteilt** sind. Dies ist beispielsweise bei der Erdbebenüberwachung und dem Informationssammeln im Internet der Fall.

5. Lösungen anzubieten, wenn **Expertise verteilt** ist. Z.B. ist bei Produktionsanlagen das Wissen verteilt. Es gibt Experten für das Design, für die Ent-

wicklung, für die Zusammenstellung der Produktionsanlagen, für die Ablaufsteuerung, etc.. Diese Experten können durch Agenten dargestellt werden, die dann auf ihrem Wissen basierend den Produktionsprozess planen und optimieren. Dabei können Agenten ihr Wissen z.B. aus Expertensystemen oder menschlichen Experten beziehen.

6. Die **Leistungserhöhung** eines Systems entlang der folgenden Dimensionen:
   a. **Effiziente Berechnungen:** Mit MAS ist es möglich die Vorteile von nebenläufigen Berechnungen auszunutzen. Agenten kann man als einzelne Threads oder Prozesse sehen, die ein Problem parallel bearbeiten.
   b. **Verlässlichkeit:** Das Wiederherstellen der Systemeigenschaften nach Komponentenausfällen kann durch das Finden von Agenten mit ähnlichen Fähigkeiten in der Phase der Neuorganisation ermöglicht werden.
   c. **Erweiterbarkeit:** Die Anzahl und die Fähigkeiten der Agenten, die ein Problem bearbeiten, kann verändert werden.
   d. **Robustheit**: Die Fähigkeit des System mit Ungewissheit umzugehen, kann durch den Austausch von Informationen zwischen Agenten erreicht werden.
   e. **Wartbarkeit:** Ein modular aufgebautes System ist einfacher zu warten als ein monolithisches System.
   f. **Ansprechbarkeit:** Ein Resultat der Modularität ist, dass Anomalien lokal behandelt und nicht ins gesamte System propagiert werden müssen. Diese weitläufige Kommunikation verlangsamt das System und verringert seine Ansprechbarkeit.
   g. **Flexibilität**: Agenten mit verschiedenen Fähigkeiten können sich organisieren, um gemeinsam ein Problem zu lösen.
   h. **Wiederverwendung**: Agenten mit speziellen Fähigkeiten können in verschiedenen Agententeams wiederverwendet werden.

### 2.2.3 Formale Definition

Multiagentensysteme werden von Fischer et al. [Fi03] definiert als eine Menge von **prototypischen Agenten**, die durch einen **Agentenverzeichnisdienst** (*agent directory service*) (ADS) beim Vorgang der Selbstorganisation unterstützt werden. Dies geschieht durch die Bereitsstellung eines Telefonbuchdienstes (*white pages*) und kann mit einem Gelbeseitendienst (*yellow pages*) ergänzt werden. Jeder

Agent in einem Multiagentensystem hat eine eindeutige Adresse, die er anderen Agenten zugänglich machen kann, indem er sich beim ADS registriert. Alle Agenten kennen automatisch die Kennung des ADS und können so mit diesem kommunizieren.

Ein **Telefonbuchdienst** erlaubt es Agenten sich gegenseitig zu finden und Namen in Adressen aufzulösen. Ein **Gelbeseitendienst** erlaubt das Suchen von Agenten anhand von Eigenschaften und Fähigkeiten. Zumindest der Telefonbuchdienst ist eine Schlüsselkomponente von verteilten Multiagentensystemen [Wr04]. In einigen System oder Simulationsumgebungen, wie z.B. SeSAm, wird dieser Dienst direkt vom System bereitgestellt.

Ein Multiagentensystem kann daher definiert werden als $\mathcal{MAS}_{prot} := (\mathcal{A}_{prot}, ADS)$. Wobei

**$\mathsf{A}_{\mathtt{prot}}$**

Die Menge $\{A^1, \cdots A^n\}, n \in \mathbb{N}$ von prototypischen Agenten. Diese Agenten sind die potentiell zur Verfügung stehenden Problemlöser.

**ADS**

Ein spezialisierter prototypischer Agent, der den Agentenverzeichnisdienst zur Verfügung stellt.

Instanzen beider Agententypen, $\mathsf{A}_{\mathtt{prot}}$ und ADS, können zur Laufzeit des Systems dynamisch hinzugefügt und entfernt werden. Der folgende Formalismus erlaubt die Darstellung eines offenen Systems als MAS.

Der Anfangszustand eines System kann durch $\mathcal{MAS}_{init} = (\mathcal{A}_{init}, ADS_{init})$ beschrieben werden, wobei

$$\mathcal{A}_{init} = \{A_1^1, \cdots, A_{k_1}^1, \cdots, A_1^n, \cdots, A_{k_n}^n\}, k_1, \ldots, k_n \in \mathbb{N}$$

und

$$\forall A_j^i \in \mathcal{A}_{init} : A^i \rhd A_j^i \wedge A^i \in \mathcal{A}_{prot}$$

gilt. $A^i \rhd A_j^i$ (zu lesen als „$A^i$ wird instanziiert von $A_j^i$“) besagt, dass $A_j^i$ eine Instanz des prototypischen Agenten $A^i$ ist. Instanzen erben das Verhalten und

das zum Zeitpunkt der Instanziierung vorhandene Wissen des jeweiligen Prototypen und können das Wissen erweitern.

$ADS_{init}$ repräsentiert *ein* ADS zu dem Zeitpunkt, in dem der erste Agent des Teilsystems gestartet wird, das unter der Kontrolle des Entwicklers steht. Es wird weiterhin angenommen, dass nicht alle Agenten in $\mathcal{A}_{init}$ durch den Systementwickler entworfen wurden. Die Menge von Agenten, die nicht vom Entwickler entworfen wurden kann als $\mathcal{A}_{open} = \{A_1^1, \cdots, A_{k_1}^1, \cdots, A_1^m, \cdots, A_{k_1}^m\}$ für $1 \leq m \leq n$ beschrieben werden. Wenn $\mathcal{A}_{open^*}$ die Menge der Prototypen für die Instanzen in $\mathcal{A}_{open}$ darstellt, dann braucht der Entwickler von $\mathcal{A}_{prot} \setminus \mathcal{A}_{open^*}$ nur die Eigenschaften der Prototypen zu kennen, um seinen Agenten den Zugriff auf Dienste von Agenten aus $\mathcal{A}_{open}$ zu ermöglichen.

Aus der Anfangskonfiguration $\mathcal{MAS}_{init}$ des Multiagentensystems entwickelt sich die dynamische Konfiguration $\mathcal{MAS}_t$ als $\mathcal{MAS}_t = (\mathcal{A}_t, ADS_t)$, wobei

$$\mathcal{A}_t = \{A_1^{1,t}, \cdots, A_{l_1}^{1,t}, \cdots, A_1^{n,t}, \cdots, A_{l_n}^{n,t}\}, l_1, \ldots, l_n \in \mathbb{N}$$

und

$$\forall A_j^{i,t} \in \mathcal{A}_t : A^i \blacktriangleright A_j^{i,t} \wedge A^i \in \mathcal{A}_{prot}$$

gilt. $\blacktriangleright$ (zu lesen als „wird umgewandelt in“) ist eine Abkürzung für $\rhd \circ \rightsquigarrow^*$, das für $A^i \rhd A_j^i$ mit $A_j^i \in \mathcal{A}_{init}$ und $A_j^i \rightsquigarrow^* A_j^{i,t}$ steht. Der Operator $\rightsquigarrow$ bezeichnet die Transformation von $A_j^i$ durch einen einzelnen Berechnungsschritt.

## 2.3 Multiagentensimulationen

*Developing good multi-agent models is still something of an art, rather than a science.*
--Nigel Gilbert

Agentenbasierte Systeme müssen - wie andere komplexe Systeme auch - vor ihrem Einsatz getestet und beurteilt werden. Das Modellieren und Simulieren von Multiagentensystemen ist daher ein wesentlicher Aspekt bei deren Entwicklung.

Eine Simulation ist die Nachahmung einer realen Sache oder Sachverhalts. Sie versucht bestimmte Eigenschaften des Verhaltens eines physikalischen oder abstrakten Systems durch das Verhalten eines anderen (einfacheren) Systems nachzubilden. Simulationen sind für gewöhnlich reduktionistisch, d.h. sie formulieren die grundlegenden Prinzipien eines Systems neu. Es werden Teile des Systems außer Acht gelassen, die den Wissenschaftler nicht interessieren und von denen interessante Teile nicht abhängen. Auf diese Weise können Systeme, die in der Realität komplex und groß sind, in eine Simulation umgewandelt werden, die eine handhabbare Größe darstellt.

Die Motive für den Einsatz von Simulationen sind [Kl04]:

- Der Gegenstand der Untersuchung ist nicht zugänglich.
- Der Wissenschaftler möchte das reale System, das untersucht wird, während seiner Experimente nicht stören.
- Die Zeitskala des untersuchten Systems ist sehr klein oder sehr groß.
- Das System, das untersucht werden soll, existiert nicht mehr oder existiert noch nicht.
- Der Wissenschaftler möchte das System verstehen. Dazu sollen Hypothesen über das System getestet werden.

Zusammenfassend gibt es zwei Arten von Simulationen. Zum ersten Simulationen eines Systems, auf das der Wissenschaftler keinen Zugriff hat oder auf das er nicht zugreifen möchte um seine Experimente durchzuführen. Zum zweiten dienen Simulationen dazu, ein echtes System zu verstehen. Diese Ziele können durch die Verwendung einer Multiagentensimulation erreicht werden. Zusätzlich übertreffen Multiagentensimulationen traditionelle Simulation in Gebieten in denen (a) das menschliche Verhalten, (b) Systeme, die ihre Dynamik aus flexiblen lokalen Interaktionen beziehen, (c) Systeme, in denen Individualität und/oder Lokalität wichtig sind, (d) sozio-technische Systeme, (e) Mehrebenensysteme oder (f) emergente Phänomene und selbstorganisierende Systeme betrachtet werden [Kl04].

In einer Multiagentensimulation wird ein Multiagentensystem in einer virtuellen Umgebung ausgeführt. Das Modell einer Multiagentensimulation besteht, wie

das reale System, aus aktiven Einheiten (Agenten), passiven Einheiten (z.B. Ressourcen) und den Interaktionen zwischen den Einheiten.

Die Vorteile des Einsatzes von Multiagentensimulationen sind unter anderem [Kl04]:

- Echte Multiagentensysteme können direkt in eine Multiagentensimulation übersetzt werden.
- Sie erlauben das Modellieren von Anpassung und Evolution.
- Sie erlauben das Modellieren eines Systems, das aus verschiedenartigen Entitäten zusammengesetzt ist.
- Sie erlauben die Betrachtung des System auf unterschiedlichen Ebenen.

Die Nachteile der Multiagentensimulation sind dagegen:

- Die Entwicklung und Simulation komplexer Systeme ist kostspielig in Bezug auf Zeit und Aufwand.
- Der Körnungsgrad (*level of granularity*), der dem hypothetischen System am besten entspricht, ist nicht leicht erkennbar.
- Es gibt gegenwärtig keinen präzisen Formalismus, um Multiagentensimulationen (und -systeme) zu beschreiben.
- Die Annahmen, auf denen das Modell beruht müssen sorgfältig ausgewählt werden. Kleine Änderungen an den Grundlagen eines Modells können große Auswirkungen haben.

# 3 Theorie holonischer Systeme

## 3.1 Holone

Die Erforschung, Beschreibung und Steuerung komplexer Systeme, wie sie in allen Bereichen der Natur und in vom Menschen geschaffenen Systemen vorkommen, ist ein wichtiges und weit verbreitetes Forschungsgebiet der Informatik. Eine natürlichsprachliche Beschreibung der Architektur komplexer Systeme wurde von dem Schriftsteller und Sozialphilosophen Arthur Koestler gegeben [Ko67], dabei prägte er den Begriff des Holonen.

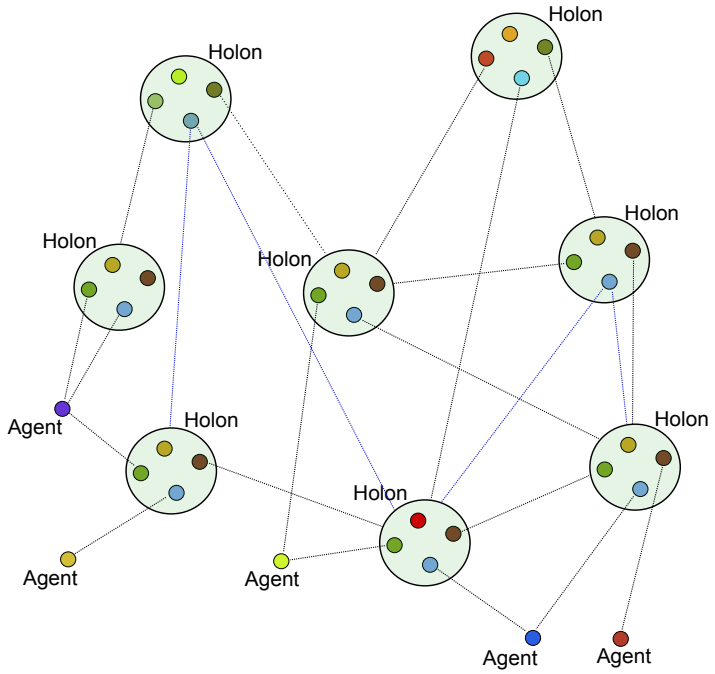
Anhand einiger Beispiel aus einer Reihe von sozialen und biologischen Organisationen betont Koestler seine Beobachtung, dass jedes System mit einem gewissen Grad an Zusammenhang und Stabilität hierarchisch geordnet ist. Als **Organisation** wird dabei ein System bezeichnet, das aus miteinander interagierenden Einheiten (Agenten) und deren Beziehungen zueinander besteht. Mit „Die Parabel von den Uhrmachern“ (S. 17) wird die Beobachtung unterstützt, dass komplexe Systeme sich aus einer Hierarchie von weniger komplexen und stabilen Zwischenstufen aufbauen müssen.

Eine wichtige Eigenschaft solcher stabilen Zwischenstufen wurde von Koestler mit dem Begriff **Janus-Effekt** bezeichnet. In der römischen Mythologie is Janus der Gott der Durchgänge und Tore, der Anfänge und der Enden. Ein auffallendes Merkmal dieser Gottheit sind seine zwei Gesichter, die in entgegengesetzte Richtungen blicken.

Wie Janus haben auch die Zwischenstufen zwei Seiten, die in entgegengesetzte Richtungen blicken. Die erste Seite blickt dabei in der Hierarchie nach unten und repräsentiert die *Ganzheit* des Elements in der Hierarchie. Als ein Ganzes ist dieses Element in sich geschlossen, stabil, vollständig und strebt seine **Autonomie** an. Die andere Seite des Elements schaut in der Hierarchie als ein *Teil* eines Größeren nach oben. In dieser Hinsicht zeigt das Element den Willen zur **Kooperation**, um Teil des größeren Ganzen zu werden.

Diese Elemente werden **Holone** genannt. Der Begriff ist eine Wortneuschöpfung aus dem Griechischen `holos`, das 'das Ganze' bedeutet und dem Suffix `on`, das man als 'Teil' übersetzen kann, wie z.B. in Proton. Ein System von miteinander

interagierenden Holonen nennt Koestler **Holarchie** (siehe Abbildung 3.1). Die Holonen können kooperieren, um ein gemeinsames Ziel zu erreichen. Holarchien werden dynamisch erzeugt und wieder aufgelöst. Dies entspricht der Fähigkeit der **Selbstorganisation**, die aus der Autonomie und dem Willen zur Kooperation der einzelnen Holon resultiert.




Die großen Kreise symbolisieren Holone, die kleinen Kreise stehen für Subholone des größeren Holonen.

Einzelne kleine Kreise sind Agenten, die die Blätter der Holarchie darstellen. Die Kanten zwischen Subholonen und Holonen bzw. Agenten geben eine 'besteht aus'-Beziehung wieder.

**Abbildung 3.1. Grafische Darstellung einer Holarchie**

Zur Beschreibung von Holonen werden folgende Begriffe definiert.

### `Subholon`

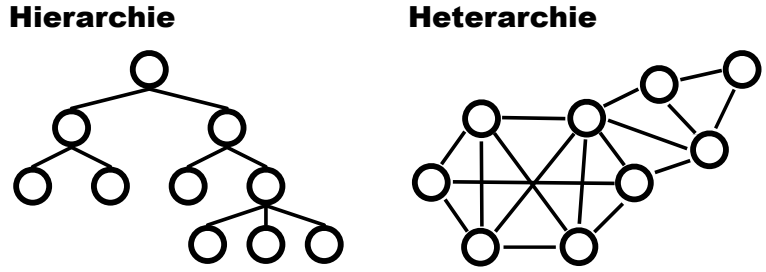
Wenn der Teil-Charakter eines Holonen betont werden soll, wird der Begriff *Subholon* verwendet um die Beziehung zwischen verschiedenen Holonen in der Holarchie zu klären.

### `Superholon`

Wenn der Begriff *Superholon* verwendet wird, soll der Ganzes-Charakter des Holonen unterstrichen werden.

Ein Superholon kann aus mehreren Subholonen bestehen, die wiederum Superholone für andere Holone darstellen können. Auf der anderen Seite kann der Superholon selbst wieder Subholon bei einem oder mehreren anderen Superholonen sein. Ein solches System von Holonen kann daher als *rekursive Struktur* betrachtet werden. Neben der Autonomie und der Kooperation ist die **Rekursion** ein weiteres wichtiges Merkmal der Holonen. Wegen der Möglichkeit, dass ein Holon Teil von verschiedenen anderen Superholonen sein kann, die sich gegenseitig durch wechselnde Beziehungen beschränken, wird die Holarchie manchmal auch als Heterarchie von Holonen bezeichnet.

Der Begriff **Heterarchie** wurde vom Kybernetiker Warren S. McCulloch eingeführt [HH05]. In einer heterarchischen Architektur kommunizieren die Agenten als gleichgestellte Partner. Es gibt keine festen Meister/Diener-Beziehung, wie das in Hierarchien der Fall ist [Ba98] (siehe Abbildung 3.2 für eine Gegenüberstellung). Weiss et al. [We00] definieren eine Heterarchie als eine Koordinationsstruktur, in der ein Agent einen anderen Agenten beschränkt, der ihn wiederum selbst beschränken kann.




Die Hierarchie weist eine feste Weisungsstruktur auf, die meist baumartig ist. Jeder Knoten, außer den Blättern und der Wurzel, besitzt einen übergeordneten und einen untergeordneten Knoten.

In der Heterarchie sind die einzelnen Knoten gleichgestellt. Kanten repräsentieren bidirektionale Beziehungen. Die Weisungsstruktur entsteht durch Verhandlungen o.ä. Mechanismen.

**Abbildung 3.2. Hierarchie und Heterarchie im Vergleich**

Holone haben einige interessante Eigenschaften, welche die von traditionellen Agenten erweitern. Ausgehend von den in Abschnitt 2.1 erläuterten Eigenschaften traditioneller Agenten wird in den nachfolgenden Abschnitten auf die Erweiterungen dieses Konzepts durch Holone eingegangen.

### 3.1.1 Autonomie

Durch die Mitgliedschaft in einem Holonen akzeptiert jeder Subholon einen bestimmten Grad der Einschränkung seiner Autonomie und Freiheit in der Kommunikation. Auf diesen Sachverhalt wird später (Abschnitt 3.2) genauer eingegangen. Subholone verpflichten sich für die Ziele des Superholonen, in dem sie Mitglied sind. Es ist dabei wichtig festzuhalten, dass Holone immer genügend Autonomie behalten, um zu entscheiden, ob sie einen Superholonen verlassen möchten (siehe Abschnitt 3.5).

### 3.1.2 Gemeinsames zielorientiertes Verhalten

Intelligente Agenten zeigen normalerweise zielorientiertes Verhalten. Die Subholonen eines Superholonen verfolgen immer noch ihre eigenen Ziele. Zumindest ein Ziel muss jedoch allen Subholonen eines Holonen gemein sein. Ansonsten würde Kooperation in Form eines Holonen keinen Sinn machen. Die Ziele des Superholonen werden von den gemeinsamen Zielen der Subholonen hergeleitet. Es ist nicht notwendig, dass die Ziele des Superholonen auch die Ziele seiner Subholonen sind. Es ist jedoch notwendig, dass zwischen den Zielen des Superholonen und seiner Subholonen kein Widerspruch besteht. Zu den daraus resultierenden Problemen und möglichen Lösungen unter Abschnitt 3.1.3 mehr.

### 3.1.3 Annahmen

In traditionellen MAS sind Annahmen idealerweise lokal beim Agenten gespeichert. Der Agent ist selbst dafür verantwortlich, dieses Wissen konsistent mit seinen vorhergehenden Wahrnehmung und seinem Wissen zu halten, damit er vernünftig handeln kann. Inkonsistenz in diesem Wissen kann zu chaotischem Verhalten führen.

In holonischen MAS ist das Wissen auch lokal beim einzelnen Holon gespeichert. Das Wissen des Superholonen muss jedoch konsistent mit dem Wissen jedes einzelnen Subholonen sein, weil sein Wissen vom Wissen der Subholone hergeleitet wird. Es ist sehr wahrscheinlich, dass sich die einzelnen Subholonen über verschiedene Aspekte der Realität oder ihrer virtuellen Umgebung nicht einig sind. Daher ist es für den Superholon sehr schwer sein Wissen mittels *Standardlogik* konsistent zu halten. Da aus einer inkonsistenten Wissensbasis jede belie-

bige Schlussfolgerung gezogen werden kann und daraus chaotisches Verhalten resultiert, muss der Zustand der Inkonsistenz verhindert werden. Hierfür sind verschiedene Alternativen zur Standardlogik vorgeschlagen worden, u.a. die *Parakonsistente Logik*, die mehr als die zwei gebräuchlichen booleschen Werte verwendet [Ge99].

### 3.1.4 Beschränkte Problemlösefähigkeit

Da jeder Agent Beschränkungen in seiner Problemlösefähigkeit unterliegt (z.B. wenn mehrere Agenten sich auf einem Computer befinden und sich dessen Ressourcen teilen müssen), scheint die Gründung von Organisationen zum gemeinsamen Problemlösen (*problem-solving organizations*, [Sy98]) angebracht. Holone können als eine solche Organisation zum gemeinsamen Problemlösen und Optimieren des Verhaltens auf einer globaleren Ebene als der des individuellen Agenten angesehen werden.

### 3.1.5 Kommunikation und Koordination

Kommunikation ist wesentlich in Organisationen für verteiltes Problemlösen. Sie wird besonders dann notwendig, wenn zwischen den Problemen, die von unterschiedlichen Problemlösern bearbeitet werden, logische Abhängigkeiten bestehen und dadurch *Koordination* notwendig ist. Die Fähigkeit zur Kommunikation muss daher bei Holonen unbedingt vorhanden sein.

### 3.1.6 Erhöhte Gruppenfähigkeiten

Der Superholon kann über mehr und bessere Fähigkeiten verfügen als seine einzelne Subholonen. Zum Beispiel kann eine einzelne Ameise keine Brücke bauen um einen Spalt zu überqueren. Eine Ameisengruppe kann jedoch eine solche Brücke bauen. Die einzelnen Ameisen in der homogenen Gruppe besitzen immer noch nicht diese Fähigkeit. Dennoch entsteht die Fähigkeit zum Brückenbau und zwar durch die Kooperation der Ameisen untereinander. Kooperation ist daher ein wichtiger Faktor bei der Erhöhung der Gruppenfähigkeit. Hier gilt, dass „Die Summe größer als ihre Teile“ ist. In Abschnitt 3.2.1 wird dieses Beispiel nochmals aufgegriffen.

## 3.2 Intraholonische Strukturen

Informell betrachtet besteht ein Holon aus mehreren anderen Holonen (Sub-Holone) oder es handelt sich um einen atomaren Holon, was einem traditionellen Agenten entspricht. Im Falle der Zusammengesetztheit kann der Grad der Autonomie der einzelnen Subholone in einem weiten Spektrum variieren. Inwieweit die Teilnahme an einem Superholon die Autonomie der Teilnehmer einschränkt wird durch *Verhandlungen* zwischen den potentiellen Subholonen und dem Kopf geregelt. Die Rolle des Kopfes wird nachfolgend erläutert.

Um diese interne Strukturen auch formal definieren zu können, wird nun der Formalismus, der in Abschnitt 2.2.3 eingeführt wurde, um holonische Strukturen erweitert.

Es sei $\mathcal{H}$ die Menge aller Holone. $\mathcal{MAS}_t$ ist dann, nach Fischer et al. [Fi03], rekursiv definiert als:

- für jedes $a \in \mathcal{A}_t, h = (\{a\}, \{a\}, \emptyset) \in \mathcal{H}$ , d.h. jeder Agent ist ein atomarer Holon, und

- $h = (Head, Subholons, C) \in \mathcal{H}$ , wobei $Subholons \in 2^{\mathcal{H}} \setminus \emptyset$ die Menge von Subholonen ist und $Head \subseteq Subholons$ die nicht-leere Menge von Holonen darstellt, die den Superholon h nach außen repräsentieren und die zuständig für die Koordination der repräsentierten Subholonen sind. $C \subseteq Commitments$ stellt die Menge der Verpflichtungen (*commitments*) dar, auf die sich bei der Erzeugung des Holonen h die Subholonen geeinigt haben.

*Verpflichtungen* sind Versprechen eines Agenten einer bestimmten Handlungsweise zu folgen. Es gibt *psychologische* und *soziale* Verpflichtungen. Bei psychologischen Verpflichtungen handelt es sich um Bindungen des Agenten an seine Annahmen und Vorhaben. Je stärker eine solche Verpflichtung ist, desto weniger wird der Agent seine Annahmen und Vorhaben überdenken. Soziale Verpflichtungen dagegen sind Verpflichtungen, die ein Agent gegenüber einem anderen Agenten eingeht. [We00] Diese Verpflichtungen setzen die Subholonen, die wiederum selbst Superholonen sein können, zueinander in Beziehung.

Die formale Definition der rekursiven Struktur von oben kann als UML Diagramm visualisiert werden (siehe Abbildung 3.3). Der Aufbau dieses Diagramms entspricht dabei der des Diagramms für das aus der Softwareentwicklung bekannten *Composite Patterns*. Es gibt atomare Objekte (Agenten) und zusammengesetzte Objekte (SuperHolonen).

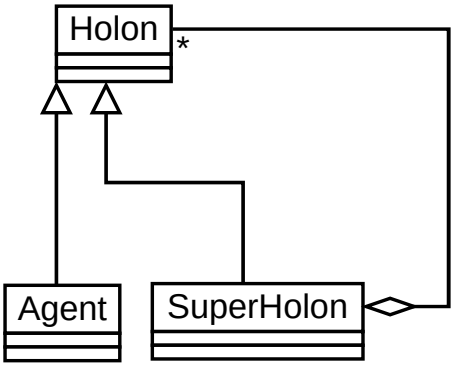
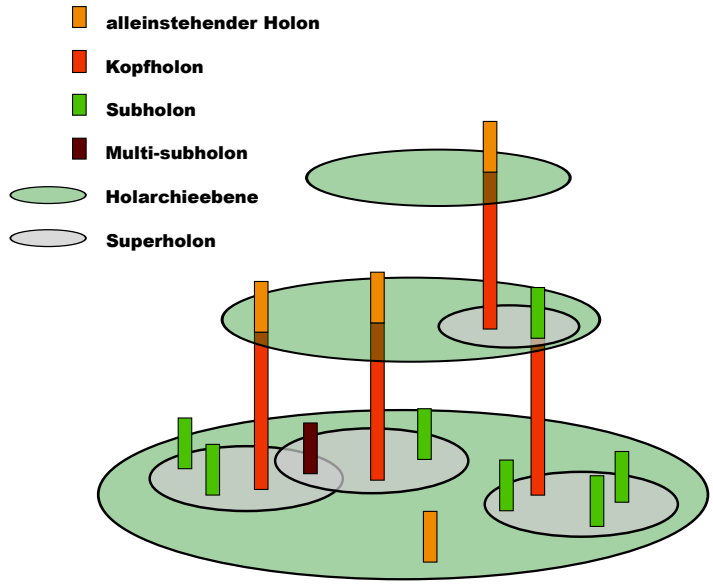



**Abbildung 3.3. Die holonische Struktur als UML-Diagramm**

Die Einführung des Konzepts des **Kopfholonen**, der für die Repräsentation des Superholonen gegenüber der Umgebung zuständig ist und der für die Einhaltung der Verpflichtungen seiner Mitglieder sorgt, geschieht aus Gründen der Effizienz. Es ist durchaus möglich, dass jeder Subholon den Superholonen nach außen repräsentiert, wodurch der Superholon vollständig dezentral organisiert wäre. Das führt jedoch zu einem Anstieg der Kommunikation innerhalb des Superholonen. Die einzelnen Subholonen müssen die Modelle, die sie von den anderen Mitgliedern im Holon aufgebaut haben, ständig anpassen und aktuell halten. Nur so können Anfragen von außen an den zuständigen Subholonen weitergeleitet werden. Die Bestimmung eines Kopfholonen und seine soziale Bevollmächtigung ist daher kommunikationstechnisch günstiger.

Von einer **sozialen Bevollmächtigung** (*social delegation* [MS03]) spricht man bei der Übertragung der Aufgabe, eine soziale Gruppe durch ein Mitglied oder eine kleine Untermenge der Mitglieder zu repräsentieren (siehe Abbildung 3.4). Diese Art der Bevollmächtigung is wesentlich, wenn große soziale Gruppen interagieren, da sie die Anzahl der notwendigen einzelnen Kommunikationsvorgänge reduziert. Ohne die Bevollmächtigung weniger durch viele sind many-to-many Kommunikationen notwendig. Im schlimmsten Fall muss jedes Mitglied aus der einen mit jedem Mitglied aus der anderen Gruppe verhandeln oder Informationen austauschen. Durch die Bevollmächtigung kann dies im Extremfall

auf eine one-to-one Kommunikation reduziert werden. Jeder Bevollmächtigte ist dann dafür zuständig, das Verhandelte an diejenigen weiterzugeben, die er repräsentiert, was nur einer one-to-many Kommunikation entspricht. Diese Vorgehensweise ist auch in der menschlichen Gesellschaft beobachtbar. Die Vorzüge und Vorgehensweisen bei der sozialen Bevollmächtigung sind beispielsweise in der *Streikschlichtung* leicht zu identifizieren. Die Interessenvertreter zweier Gruppen treffen dabei zusammen, verhandeln über eine bestimmte Sachlage, z.B. Tarife, und geben ihre Entschlüsse dann an die Mitglieder der jeweiligen Gruppen weiter. Durch dieses Vorgehen wird der Kommunikationsaufwand drastisch reduziert.



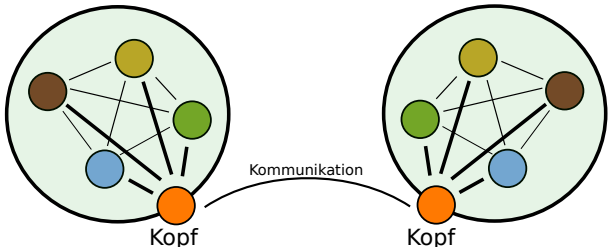

Die Kopfholonen repräsentieren den Holonen, dem sie vorstehen, in der nächsthöheren Ebene der Holarchie. [HI04]

**Abbildung 3.4. Eine Holarchie mit Kopfholonen als Repräsentanten**

Im Falle einer solchen zentralisierten Struktur, in der der Kopfholon eine Rolle ähnlich dem ADS (siehe Abschnitt 2.2.3) einnimmt, braucht nur dieser sein Wissen über die Mitglieder im Holon auf dem aktuellen Stand zu halten. Die Subholonen fordern dann nur die Informationen beim Kopfholon an, die für ihre Aufgabenstellung notwendig sind. Auf diesen Zusammenhang wird in Abschnitt 4.1 detaillierter eingegangen werden. In diesem zentralisierten Ansatz könnte sich der Kopfholon jedoch zu einem Flaschenhals in der Kommunikation entwickeln. Dem kann durch die Delegation einiger der Kommunikationsaufgaben an Subholonen entgegengewirkt werden.

Einerseits kann der Kopf aus der Menge der Subholone durch einen Wahlmechanismus ausgesucht werden. Einige dieser Mechanismen werden in Abschnitt 3.3 beschrieben. Andererseits kann der Kopfholon für die Lebensdauer des Holonen erzeugt werden und danach wieder terminieren. Dieser Mechanismus wird in Abschnitt 3.7.1 beschrieben. Wenn `Kopf` die Menge der Repräsentanten des Holonen ist, dann ist `Subholone\Kopf` die Menge der Subholone, die vollständig im Superholon eingekapselt sind, mit denen also von außen nicht kommuniziert werden kann. Diese Menge wird `Körper` genannt. *Verpflichtungen* kann man als das Bindemittel zwischen `Kopf` und `Körper` betrachten.

Durch soziale Bevollmächtigung ist ein Holon von außen nicht von anderen Agenten in $A_t$ zu unterscheiden. Dies begünstigt die Einbindung von Holonen in bereits bestehende Agentensysteme. Auch die Kommunikation zwischen Holonen kann durch die Kopfholonen abstrahiert werden, d.h. es ist nicht notwendig die im Holon realisierte Organisation zu kennen, um mit ihm kommunizieren zu können. Der tatsächliche Holon versteckt sich quasi hinter dem Kopf, wie in Abbildung 3.5 dargestellt.



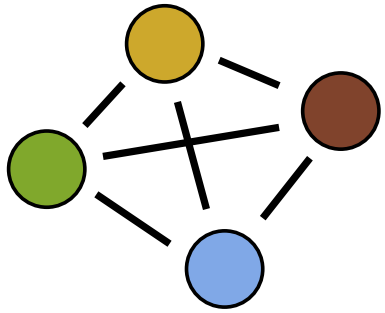

Die beiden Kopfholonen eines Superholonen kommunizieren stellvertretend für die jeweiligen Subholonen.

**Abbildung 3.5. Kopfholone kommunizieren**

Holone stellen keine einzelne Organisationsform dar, sondern können jede beliebige Organisationsform umsetzen. Man kann Holone als ein Mittel zur Kapselung von Organisationsstrukturen betrachten. In den nachfolgenden Abschnitten werden die wichtigsten Organisationsformen beschrieben. Das Spektrum der Organisationsformen wird durch zwei Extreme begrenzt.

### 3.2.1 Vollständige Autonomie der Subholonen

Das eine Extrem in der intraholonische Beziehung zwischen Subholonen ist die, in der die Subholonen ihre vollständige Autonomie behalten. Hier sind sich die Subholonen nicht darüber bewusst, dass sie in einem Superholonen teilnehmen. Der Superholon entsteht vielmehr durch die *Kooperation* der Subholonen und existiert nur implizit. Abbildung 3.6 zeigt die interne Struktur eines solchen Holonen und die möglichen Interaktionen zwischen seinen Subholonen.



Der Superholon existiert nicht explizit, sondern ist eine Sichtweise auf ein System von kooperierenden Einheiten.

**Abbildung 3.6. Ein Holon bestehend aus autonomen Subholonen**

Im folgenden Beispiel wird klar, dass die hier beschriebene intraholonische Struktur nur eine andere Art ist, ein traditionelles Multiagentensystem zu betrachten. Der Superholon, im Beispiel der Brückenbauholon, existiert nicht als eigentliches Gebilde in der Realität. Die holonische Struktur ist nur eine Hilfe bei der Entwicklung von MAS, weil sie es ermöglicht, das System auf verschiedenen Abstraktionsebenen zu betrachten. Diese Ebenen entsprechen den Ebenen in der Holarchie. Die verschiedenen Betrachtungsebenen sind somit explizit im System enthalten.

#### Beispiel: Brückenbauende Ameisen als Holon

Ameisen kann man als Agenten betrachten, die vollständig autonom Probleme lösen. Mit 'vollständig autonom' wird die Tatsache bezeichnet, dass es keinen zentralen Befehlshaber gibt, der jeder Ameise sagt, was sie zu tun hat. Vielmehr leitet sich das Verhalten der einzelnen Ameise direkt aus ihrer Umgebung ab.

Es soll ein Spalt, z.B. zwischen zwei Ästen, überwunden werden, weil auf der anderen Seite eine Futterquelle entdeckt wurde. Ameisen sammeln sich und beginnen eine Brücke zu bauen, die aus ihnen selbst besteht und anderen Ameisen ermöglicht auf die andere Seite des Spalts zu gelangen. Der Brückenbau ist kein geplantes Vorgehen, sondern ein emergentes Phänomen, das aus den Verhaltensweisen der einzelnen Ameisen hervorgeht. Die Ameisen koordinieren sich, z.B. indem sie ihre Körper aneinander haken, um die Brücke zu bauen. In dieser Ameisengruppe gibt es keinen ausgezeichneten Anführer oder zentralen Koordinator, dennoch kann man die Ameisengruppe als Holon bezeichnen, dem Brückenbauholon.

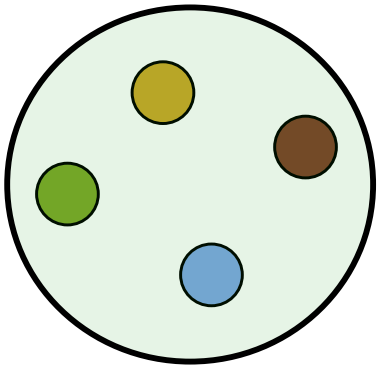
**Abbildung 3.7. Ameisen beim Bau einer Brücke**

### 3.2.2 Vollständiger Verlust der Autonomie der Subholonen

Das andere Extrem der intraholonischen Beziehungen ist der Fall, in dem die Subholone ihre individuelle Autonomie vollständig verlieren, d.h. der Kopfholon hat vollständige Verfügungsgewalt über die Subholonen. Dadurch, dass die Subholonen den Anweisungen des Kopfholonen unbedingt folgen, ist die Mitgliedschaft in einem weiteren Superholon ausgeschlossen.

Fischer et al. [Fi03] erwähnen eine noch stärkere Form dieser intraholonischen Struktur. Hier *verschmelzen* die Subholonen, um einen neuen Agenten zu erschaffen. Der resultierende Superholon besteht aus den Subholonen, d.h. er besitzt deren Fähigkeiten und Wissen. Die einzelnen Subholonen existieren jedoch nicht mehr (siehe Abbildung 3.8). Wenn der Superholon terminiert wird, dann können die einzelnen Subholonen wieder erschaffen werden. Dies muss jedoch nicht der Fall sein. Für das Verschmelzen und die evtl. spätere Teilung werden spezielle Mechanismen benötigt [IS92]. In Abschnitt 3.1.3 wurden bereits

mögliche Schwierigkeiten beim Verschmelzen erwähnt, u.a. inkonsistentes Wissen. Speziell bei heterogenen Agenten führen diese Probleme dazu, dass eine Verschmelzung praktisch nicht möglich ist. Aufgrund dieser Schwierigkeiten wurde von einer Umsetzung in dieser Arbeit Abstand genommen und stattdessen die erste hier erwähnte Beziehungsstruktur gewählt.

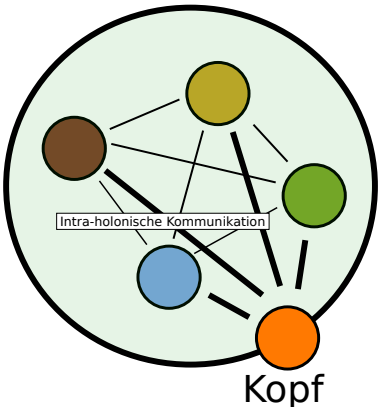


Die einzelnen Subholonen sind im Superholonen vollständig aufgegangen, können nach dessen Beendigung jedoch wieder erschaffen werden.

**Abbildung 3.8. Ein Holon, der aus verschmolzenen Subholonen besteht.**

### 3.2.3 Ein Holon als moderierte Gruppe

Unter Verwendung des Konzepts des Kopfholonen ist es möglich die zwei internen Strukturen, die in den vorherigen Abschnitten beschrieben wurden, als Instanz einer Struktur zu betrachten, die „Holon als moderierte Gruppe“ genannt wird [Ge99]. Dabei ist der Kopfholon der Moderator der Gruppe. Die vollständige Autonomie der Subholone (siehe Abschnitt 3.2.1) kann durch einen Kopfholon realisiert werden, der die Kommunikation und das Verhalten der Subholonen in keiner Weise beschränkt. Der vollständige Verlust der Autonomie (siehe Abschnitt 3.2.2) kann durch einen Kopf umgesetzt werden, der die Kommunikation unter den Subholonen vollständig einschränkt und der den Subholonen Verhalten aufzwingt. In beiden Fällen können die Restriktionen vor dem Eintritt in den Superholon verhandelt werden. Zusammenfassend besteht also die Möglichkeit, beide beschriebenen Extreme durch einen Holonen als moderierte Gruppe umzusetzen. Im nachfolgenden Abschnitt soll gezeigt werden, dass alle interessanten Strukturen, die zwischen den beiden Extremen liegen, ebenfalls dargestellt werden können. Die moderierte Gruppe mit der zentralen Instanz des Kopfholonen ist eine generische Struktur für die Implementierung von intraholonischen Beziehungen.




Der Moderator, auch Kopfholon genannt, repräsentiert die anderen Subholonen gegenüber der Umwelt.

**Abbildung 3.9. Ein Holon als moderierte Gruppe**

## 3.2.4 Andere Organisationsformen

Das Spektrum der *Organisationsformen*, das zwischen den beiden in vorherigen Abschnitten beschriebenen Extremen liegt wurde in [MS03] in sieben unterschiedliche Organisationsformen unterteilt. Die Reihenfolge in der nachfolgenden Aufzählung spiegelt dabei die zunehmend stärker werdende Verknüpfung zwischen den Subholonen wieder. Dadurch, dass die Beziehungen starrer werden, werden die Organisationsformen stabiler, die Autonomie der einzelnen Mitglieder nimmt durch neue Beschränkungen gleichzeitig ab. Im Folgenden werden diese Formen kurz anhand von Beispielen aus dem Unternehmensbereich erläutert.

### Alleinstehende, autonome Agenten

Die Agenten stehen in keiner Beziehung zueinander und arbeiten nicht zusammen, um ein Problem zu lösen, d.h. sie betreiben keine Aufgabenteilung (*task-sharing*). Die einzige Interaktion besteht zwischen Agenten, die etwas anbieten, und Agent, die etwas benötigen. Diese Einschränkung in der Interaktion ist so stark, dass nicht von einer Organisation in der oben definierten Weise gesprochen werden kann. In Abschnitt 3.2.1 wird eine ähnliche Konstellation beschrieben, die als Organisationsform bezeichnet werden kann. Diese ist im holonischen Umfeld dennoch wenig interessant, weil keine explizite holonische Struktur vorliegt. Als abstrahierende Sichtweise auf eine Menge von Agenten, die gemeinsam

Problemlösen, kann eine holonische Herangehensweise dennoch hilfreich sein, wie in Abschnitt 3.2.1 bereits erläutert wurde.

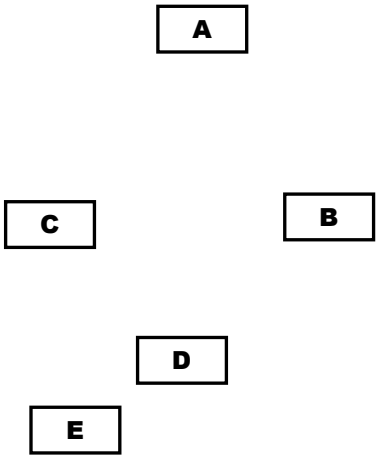
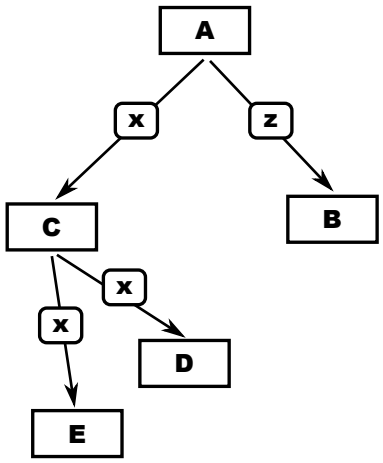



**Abbildung 3.10. Fünf unabhängige Agenten**

## Markt

In einem Markt soll eine optimale Aufgabenverteilung durch das Wetteifern der beteiligten Agenten erreicht werden. Dafür ist nur minimale Kommunikation unter ihnen notwendig, denn ein Großteil der Kommunikation läuft über eine zentrale Stelle, die Gebote annimmt und aus der Anfrage neue Preise berechnet.



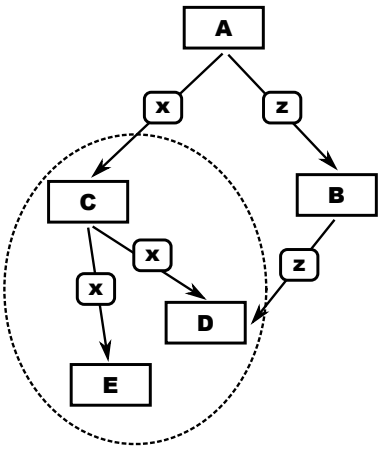

Fünf Unternehmen, die Aufgaben verteilen. Unternehmen A ist Kopfholon für die Aufgaben x und z.

**Abbildung 3.11. Organisationsform: Markt**

Wie die vorherige Form, schränkt auch diese Form die Subholonen nicht ein. Es steht den Agenten frei an mehreren Märkten gleichzeitig teilzunehmen. Ein Subholon, der eine Aufgabe über einen Marktmechanismus an andere Subholonen verteilt, dient als Kopfholon für die Bearbeitung dieser Aufgabe. Die daraus resultierende Struktur ist baumähnlich mit optionalen Kanten zwischen verschiedenen Unterbäumen, wenn ein Holon in mehreren Superholonen Mitglied ist. Ändert sich für die Bearbeitung einer neuen Aufgabe der Kopfholon, so wird ein neuer Superholon mit diesem Kopf erzeugt. In Abschnitt 3.4.2 wird gezeigt, wie ein Markt für die Aufgabenverteilung innerhalb eines Superholonen genutzt werden kann.

## Virtuelles Unternehmen

Ein *virtuelles Unternehmen* (VU) ist ein Zusammenschluss von Unternehmen, die ihre Fähigkeiten kombinieren, um ein Produkt zu erzeugen, das keiner von ihnen allein erzeugen könnte. Nach außen, gegenüber anderen Unternehmen, erscheint ein VU als einzelnes Unternehmen, da es von einem der teilnehmenden Unternehmen repräsentiert wird. Dieses Unternehmen besitzt die *soziale Bevollmächtigung* durch die anderen Mitglieder des VUs. In diesem Sinn ist ein VU die erste Organisationsform in dieser Liste, die holonische Eigenschaften aufweist.



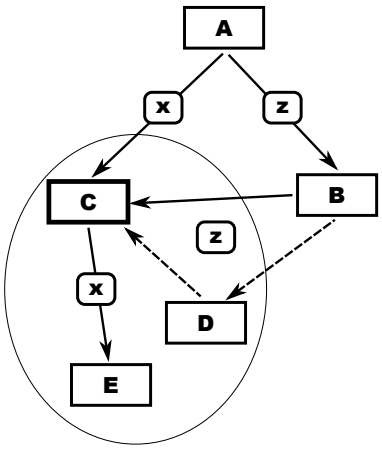

Unternehmen C, D und E bilden ein virtuelles Unternehmen.

**Abbildung 3.12. Organisationsform: Virtuelles Unternehmen**

Die Mitglieder eines VUs können auch in anderen VUs teilnehmen. Jedes Mitglied eines VUs kann neue Aufgaben von außen akzeptieren und kann dann als Kopfholon für die Bearbeitung dieser Aufgabe im VU fungieren oder die Bearbeitung an den aktuellen Kopfholon des VUs weitergeben. Wenn es die Aufgabe nicht selbständig lösen kann, so werden erst die anderen Mitglieder des VUs um Hilfe gebeten und erst, wenn auch das nicht zum Erfolg führt, werden externe Ressourcen bemüht und evtl. ein neues VU für die Bearbeitung dieser Aufgabe gegründet.

## Allianz

Anders als im VU ist es in einer Allianz nicht mehr jedem Mitglied möglich, sie nach außen hin zu repräsentieren. Auch darf in einer Allianz nur der gewählte Repräsentant neue Aufträge annehmen. Diese Organisationsform ist demzufolge zentralistischer als die bisherigen. Dennoch sind die internen Beziehungen einer Allianz kooperativ und nicht konkurrenzbetont oder autoritär.



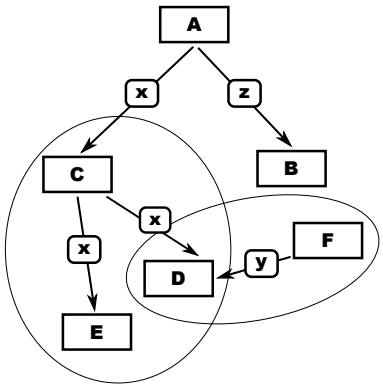

Unternehmen C, D und E bilden eine Allianz. D reicht eine Aufgabe an den Kopf C weiter.

**Abbildung 3.13. Organisationsform: Allianz**

## Strategisches Netzwerk

Der Unterschied zur Allianz liegt in den Beziehungen zwischen den Mitgliedern. Im Falle des strategischen Netzwerks (SN) sind diese Beziehungen *autoritär*. Typischerweise wird ein SN von einem Unternehmen dominiert, das bereits eine gute Position im durch das SN angezielten Markt aufweist. Dieses Unternehmen koordiniert das Netzwerk und repräsentiert es nach außen. Strategische Netzwerke

sind zuverlässiger als die vorherigen Organisationsformen, weil sie Aufgabenverteilung durch vertraglich gesicherte autoritäre Strukturen erreichen. Die Mitglieder eines Netzwerks sind zu einem hohen Grad vom Netzwerk abhängig, da die meisten ökonomischen Abwicklungen innerhalb des Netzwerks stattfinden. Der Grund für das Einrichten eines Netzwerks ist, einen Wettkampfvorteil gegenüber anderen Mitbewerbern durch das Bündeln von Ressourcen zu erreichen. Beispielsweise kann in einem Unternehmen im Netzwerk ein Produkt entwickelt und in einem anderen produziert werden.

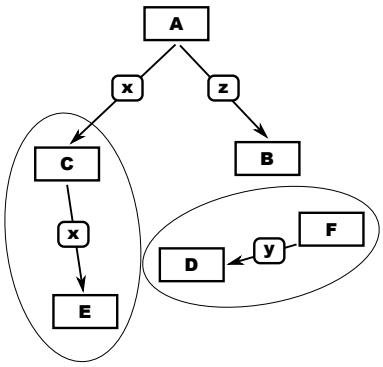



Ein System mit zwei Strategischen Netzwerken, wobei Unternehmen D in zwei Netzwerken eingegliedert ist.

**Abbildung 3.14. Organisationsform: Strategisches Netzwerk**

## Gruppe

Die Mitglieder einer Gruppe stehen noch enger miteinander in Verbindung als dies beim Netzwerk der Fall ist. Der Leiter der Gruppe ist das stärkste Unternehmen in der Gruppe. Die Verträge, die die Beziehungen in der Gruppe bestimmen, werden nicht bilateral, sondern unter Einbeziehung aller Unternehmen in der Gruppe erarbeitet. Diese Verträge sind hochgradig einschränkend und resultieren in einer autoritären Hierarchie. Dies führt dazu, dass ein Mitglied einer Gruppe nicht auch Mitglied anderer Gruppen sein kann. Der Leiter wird über die Aufgaben aller Mitglieder informiert und die Mitglieder müssen Aufgabenzuweisungen durch den Leiter akzeptieren. Nur das leitende Unternehmen darf neue Beziehungen zu externen Unternehmen oder Kunden selbständig eingehen.

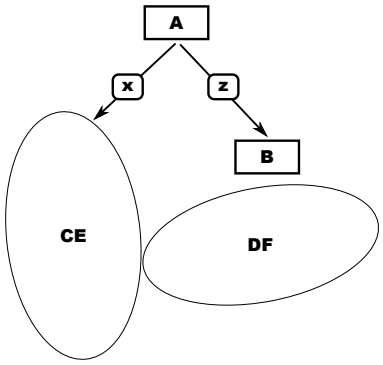



Ein System mit zwei Gruppen.

**Abbildung 3.15. Organisationsform: Gruppe**

## Gesellschaft

Wie bereits in Abschnitt 3.2.2 beschrieben, besteht das andere Ende des Spektrums der Organisationsformen im totalen Verlust der Autonomie der Subholonen eines Superholonen. Alles Wissen und alle Ressourcen der Subholonen werden zu einem neuen Agenten kombiniert. Es ist möglich, dass einer der ursprünglichen Subholone dabei die anderen Subholonen aufnimmt. Dadurch reduziert sich der Verwaltungsaufwand, weil die Kopfholonen der übrigen Subholone nicht mehr benötigt werden. Durch die Kombination verringert sich auch der Kommunikationsaufwand, weil der neue Kopfholon mit allen Subholonen, auch denen aus vorher separaten Superholonen, direkt kommunizieren kann.

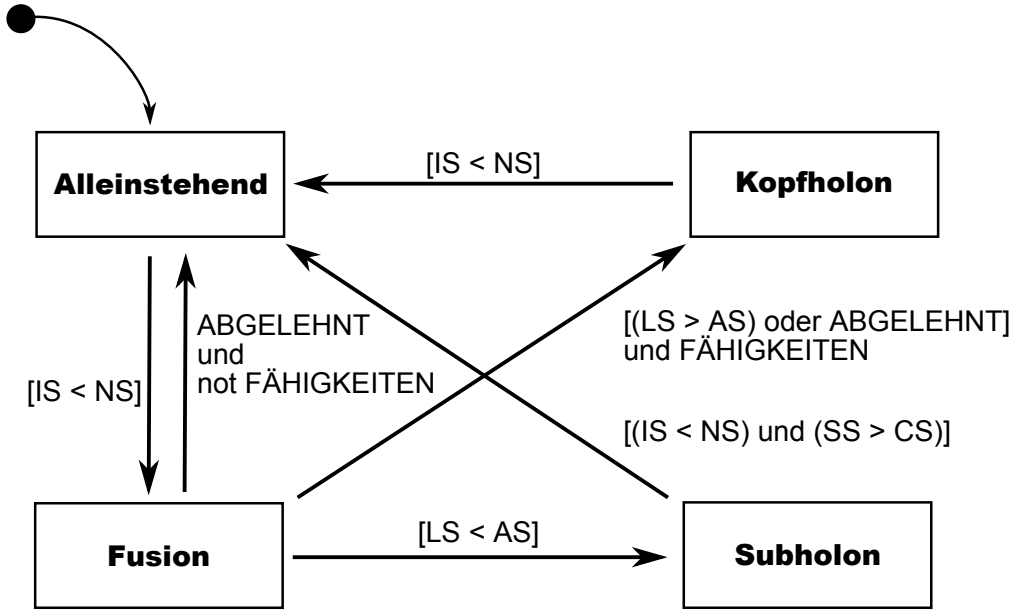



Die Unternehmen C, D, E, und F haben zwei Gesellschaften CE und DF gebildet.

**Abbildung 3.16. Organisationsform: Gesellschaft**

## 3.3 Gründung eines Holonen und Bestimmung des Kopfes

Nachdem die internen Strukturen eines Holonen im letzten Abschnitt beschrieben wurden, werden nun die Möglichkeiten erläutert, die zur Gründung eines Holonen führen und wie der Kopfholon bestimmt werden kann. Es gibt zwei Wege, durch die die Gründung eines Holonen angeregt werden kann. Zuerst besteht die Möglichkeit, dass ein Holon, der die notwendigen Fähigkeiten (siehe 'Allianz' in Abschnitt 3.2.4) eines Kopfholonen besitzt, den Bedarf an einem neuen Holon bemerkt. Er wird dann bei andere Holonen im System anfragen, ob diese sich für eine gemeinsame Problemlösung in einem Superholon zusammenfinden möchten. Zweitens gibt es die Möglichkeit, dass ein Holon, der nicht die Fähigkeiten zum Leiten eines Superholonen besitzt, zu dem Schluss kommt, dass er Mitglied in einem Superholon werden möchte. Er wird daher bei anderen Holonen anfragen, ob sie eine freie Stelle haben, die er einnehmen kann. Durch Verhandlungen wird er dann seine Position in der Organisation, die durch den Superholon repräsentiert wird, festlegen. In Abschnitt 3.4 werden einige Mechanismen beschrieben, die bei der Gründung eines Superholonen zum Einsatz kommen können, um für den Holon geeignete Mitglieder zu finden.




Zustandsautomat für das Gründen und Verlassen von Superholonen und die Mitgliedschaft als Subholon.

**Abbildung 3.17. Formelles and informelles Rollenspiel**

Holonen können sich in Bezug auf ihre Mitgliedschaft in Superholonen in verschiedenen Zuständen befinden (Alleinstehend, Kopf, Körper). Diese Zustände und die Übergänge zwischen ihnen können in einen Zustandsautomaten umgesetzt werden, wie ihn Abbildung 3.17 zeigt. Gegenüber dem Original aus [HI04] berücksichtigt der hier abgebildete Automat die Fähigkeiten des anfragenden Holonen.

**IS - Instant Satisfaction**
: Die Befriedigung eines Holonen mit seiner bisherigen Rolle als Subholon, Kopfholon oder alleinstehend.

**NS - Necessary Satisfaction**
: Ein Schwellwert, der den notwendigen IS-Wert angibt, damit die gegenwärtige Aufgabe erledigt wird. Wird dieser Schwellwert unterschritten, so wird die Bearbeitung der Aufgabe abgebrochen.

**LS - Leadership Satisfaction**
: Die Befriedigung, die einem Holonen dadurch entsteht, dass er Leiter, d.h. Kopfholon ist.

**AS - Accumulative Satisfaction**
: Die Befriedigung ein Mitglied im Körper eines Holonen zu sein.

**SS - Self Satisfaction**
: Die Befriedigung eines Holonen mit seiner eigenen Arbeit.

**CS - Collective Satisfaction**
: Die Befriedigung eines Holonen mit seiner eigenen und der Arbeit der anderen Holonen in dem Superholon, der gerade betrachtet wird.

**ABGELEHNT**
: Es wurde dem anfragenden Holonen nicht erlaubt im Superholonen teilzunehmen.

**FÄHIGKEITEN**
: Die Fähigkeiten, die für die Rolle des Kopfholonen benötigt werden sind beim anfragenden Holonen vorhanden.

Nach der Initialisierung des System befinden sich alle Holonen im Zustand 'Alleinstehend', d.h. sie sind nicht Mitglied in einem Superholonen. Falls die Befriedigung eines Holonen mit seiner bisherigen Rolle über einem vorgegebenen

Schwellwert bleibt, so bleibt er auch im Zustand 'Alleinstehend'. Die **Befriedigung** und der **Schwellwert** können beide dynamische Werte sein. Meist wird jedoch die Befriedigung aus verschiedenen Parametern vom Holonen selbst berechnet und der Schwellwert bleibt auf einem vom Entwickler festgesetzten Wert. Durch verschiedene Ereignisse, beispielsweise die Unfähigkeit eine Aufgabe zu bearbeiten, kann die Befriedigung unter den Schwellwert fallen. Dadurch ändert sich der Zustand im Zustandsautomaten von 'Alleinstehend' nach 'Fusion'. Dies führt dazu, dass sich der Holon um eine Mitgliedschaft in einem Superholon bemüht. Hierzu führt er Verhandlungen mit bereits existierenden Superholonen. In Kapitel 4 wird erläutert, wie der Agent die Superholonen finden kann. Falls er von allen Superholonen, bei denen er angefragt hat, als Mitglied abgelehnt wird, so gründet er selbst einen Superholonen, in dem er die Rolle des Kopfholonen einnimmt. Es kann vorkommen, dass der Holon möglicherweise nicht die Fähigkeiten mitbringt, Kopfholon zu sein, z.B. weil ihm bestimmte Kommunikationsfähigkeiten fehlen. Daher kann der fusionierende Holon nur dann einen neuen Superholonen gründen, wenn er auch die notwendigen Fähigkeiten besitzt. Andernfalls wird er wieder zurück in den Zustand 'Alleinstehend' wechseln. Falls der anfragende Holon jedoch in einem Superholon als Mitglied akzeptiert wird, tritt er ihm als Subholon bei.

In dem Zustand 'Subholon' bleibt ein Holon solange, bis seine Befriedigung wiederum unter einen vorgegebenen Schwellwert sinkt und er zusätzlich mehr Vertrauen in seine eigene als in die Arbeit der anderen Subholonen hat, z.B. wenn er unzufrieden mit der Arbeit der anderen Mitglieder ist. Diese zweite Bedingung zeigt die kooperative Natur der Beziehungen im Holon. Der Subholon würde den Superholon nicht verlassen, wenn er nicht auch unzufrieden mit der Arbeit der anderen Subholonen wäre. Ein Kopfholon im Gegensatz verlässt einen Holon einfach, wenn er unzufrieden ist. Da der Kopfholon jedoch ein wichtiges Mitglied des Holonen ist, ist sein Austritt aus dem Holon komplizierter als der Austritt eines Körperholonen. Beide werden in Abschnitt 3.5 beschrieben. Ein Kopfholon, der einen Holon verlässt, muss entscheiden, ob er den gesamten Holon beendet oder ob er seine Rolle als Kopf an einen anderen internen oder externen Holon delegiert. Wenn dies nicht möglich ist, so wird wiederum der gesamte Holon beendet.

## 3.4 Aufgabenteilung

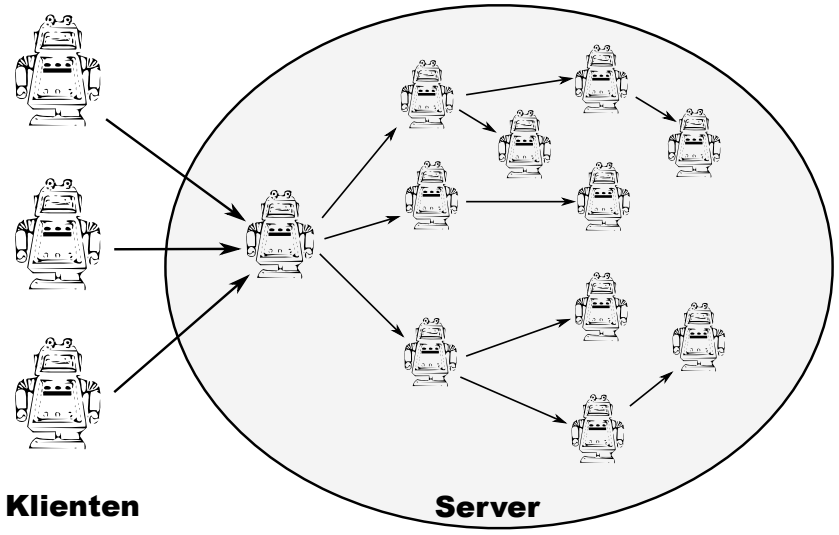
Bei der Aufgabenteilung wird eine Aufgabe, die von einem Holonen allein nicht oder nicht zufriedenstellend bearbeitet werden kann, in kleinere Teilaufgaben zerteilt, die dann an verschiedene Problemlöser verteilt werden. Zuerst muss eine Aufgabe so zerteilt werden, dass sie durch das Zusammenarbeiten mehrerer Holone leichter zu bearbeiten ist. Das ist dann der Fall, wenn die Teilaufgaben möglichst unabhängig sind. Die Bearbeitung einer Teilaufgabe soll nicht von der Bearbeitung einer anderen Teilaufgabe abhängen, da sonst weitere Koordination nötig ist. Da Aufgabenteilung ein schweres Problem darstellt, wird sie meist vom Entwickler des Systems vorgegeben. Durch *hierarchisches Planen* kann sie auch von Agenten vorgenommen werden oder sie kann der Aufgabenrepräsentation inhärent sein, z.B. wenn die Aufgabe in einem Und-Oder-Graphen dargestellt ist. Nach der Zerteilung müssen die Teilaufgaben Problemlösern zugewiesen werden. In den nachfolgenden Abschnitten werden verschiedene Methoden der Aufgabenzuweisung erörtert. Die komplexeren dieser Methoden dienen dazu, den bestmöglichen Problemlöser für das Bearbeiten einer Aufgabe zu finden. Nach dem Zuweisen an einzelne Problemlöser können diese die ihnen zugewiesene Teilaufgabe wiederum zerlegen und zuweisen, oder selbst bearbeiten. Wenn die Bearbeitung abgeschlossen ist, werden etwaige Ergebnisse von dem Holonen, der die Aufgabe zerteilt und zugewiesen hat, wieder zusammengesetzt.

Die hier beschriebenen Mechanismen helfen auch bei der Bildung eines neuen Holonen. Beispielsweise wird das *extended contract net protocol* erfolgreich in dem in Abschnitt 3.7.1 beschriebenen TeleTruck-System eingesetzt.

### 3.4.1 Aufgabenzuweisung durch Unterordnung

Der einfachste Ansatz um Aufgabenzuweisung zu realisieren besteht darin, hierarchische Unterordnung in einer statischen, zentralen Struktur zu verwenden. Das bedeutet, dass Holone aus einer höheren Ebene der Organisation Aufgaben an Holone der darunterliegenden Ebenen verteilen können. Diese untergeordneten Holone können der Zuweisung nicht widersprechen, aber sie können die Aufgabe ggf. zerteilen und an ihnen untergeordnete Holonen weitergeben (siehe Abbildung 3.18).

In strategischen Netzwerken und Gruppen (siehe Abschnitt 3.2.4) kann diese Art der Aufgabenzuweisung eingesetzt werden.




Pfeile geben die Richtung von Anfragen wieder. Die Agenten in den Graphenblättern bearbeiten die Anfragen schließlich.

**Abbildung 3.18. Taskverteilung in einer Hierarchie**

## 3.4.2 Marktmechanismen

Marktmechanismen werden in wettkampforientierten Umgebungen eingesetzt. In Umgebungen, in denen die Holonen eigennützig sind. Eigennützige Holonen sind nicht aufrichtig, d.h. sie geben nicht den tatsächlichen Wert einer Sache, z.B. die Kosten für die Bearbeitung einer Aufgabe, an. Marktmechanismen werden eingesetzt, um dieser Unaufrichtigkeit Rechnung zu tragen und eine ungefähre Annäherung an den tatsächlichen Wert zu erhalten. Für diesen Zweck spricht man von *Produzenten*, die ein Gut zu einem bestimmten Preis verkaufen möchten, auf der einen Seite. Auf der anderen Seite gibt es *Konsumenten*, die Güter erwerben möchten. Natürlich wollen Produzenten einen möglichst hohen Preis erzielen, während Konsumenten einen möglichst niedrigen Preis vorziehen.

Bei der Aufgabenverteilung sind die Güter das Bearbeiten einer Aufgabe. Die Preise sind entweder die Kosten (Ressourcen- und Berechnungskosten) für das Bearbeiten einer Aufgabe oder der Wert der Aufgabe. Problemlöser stellen in diesem Markt die Produzenten dar. Konsumenten sind Holone, die daran interessiert sind, eine Aufgabe bearbeitet zu bekommen, dies aber selbst nicht tun können. Konsumenten bieten Produzenten einen (virtuellen) Betrag, der ihrer

Einschätzung nach dem Wert der Aufgabe entspricht (oder darunter liegt). Produzenten verkaufen nur dann, wenn der ihnen angebotene Betrag den Kosten für die Bearbeitung der Aufgabe entspricht. Der Markt kann als Holon modelliert werden, in dem Produzenten und Konsumenten Subholonen sind. Eine zentrale Komponente, im holonischen Umfeld bietet sich hier der Kopf des Marktholonen an, sorgt für den Abgleich zwischen den Preisvorstellungen von Produzenten und Konsumenten. Es können auch dezentrale Marktmechanismen eingesetzt werden [Wa98]. Sowohl der zentralisierte, als auch der dezentrale Ansatz führt zu einer optimierten Aufgabenverteilung zwischen den Teilnehmern im Markt.

Wie der Name bereits nahe legt wird die Aufgabenzuweisung durch Marktmechanismen in der in Abschnitt 3.2.4 beschriebenen Organisationsform Markt eingesetzt. Außerdem eignet sich der Mechanismus zum Einsatz in virtuellen Unternehmen, da die im VU teilnehmenden Unternehmen zwar miteinander kooperieren, aber dennoch im wirtschaftlichen Wettkampf stehen und somit eigennützig und unaufrichtig sind.

### 3.4.3 Contract Net Protocol

Bei Holonen, die in einer kooperativen Beziehung zueinander stehen, kann das Contract Net Protocol (CNP) für die Aufgabenverteilung eingesetzt werden. Anders als bei Marktmechanismen geht man hier davon aus, dass die beteiligten Holone aufrichtig sind, d.h. den tatsächlichen Preis für die Bearbeitung einer Aufgabe nennen. In Abbildung 3.19 ist das CNP, gemäß der Spezifikation der *Foundation for Intelligent Physical Agents* (FIPA) [CN05] als UML Sequenzdiagramm wiedergegeben. Die zentrale problemlösende Einheit, die für die Bearbeitung einer Aufgabe verantwortlich ist, wird hier *Manager* genannt. Der Manager erhält eine Aufgabe, zerteilt diese in Teilaufgaben und verwendet eine *first-price-sealed-bid Auktion* um diese and die bestgeeignetsten Problemlöser weiterzuverteilen. Diese Problemlöser werden *Contractors* oder *Teilnehmer* genannt. Nach der Bearbeitung der Teilaufgaben schicken die Teilnehmer ihre Ergebnisse an den Manager zurück. Dieser erzeugt aus den Teilergebnissen ein Gesamtergebnis. Das Besondere an diesem Mechanismus ist, dass jeder Teilnehmer wiederum als Manager fungieren kann. Er kann die erhaltene Teilaufgabe seinerseits zerteilen und über das CNP Bearbeiter suchen. Das Protokoll kann hiermit rekursiv angewendet werden.

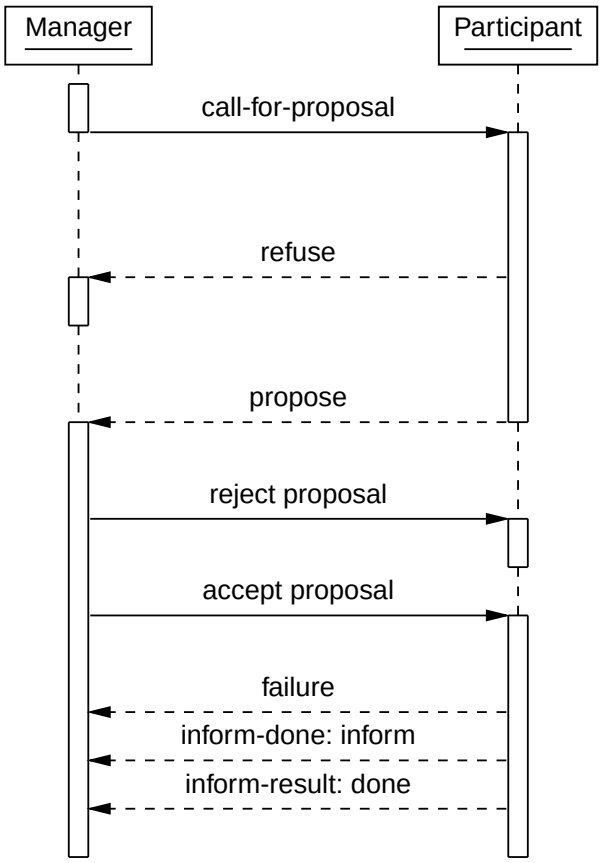



**Abbildung 3.19. Das Contract Net Protocol nach FIPA-Spezifikation**

Jeder Teilnehmer, der ein *call-for-proposal* (CFP) empfangen hat, kann entweder ablehnen, oder ein Gebot abgeben. Da der Manager sich nicht darauf verlassen kann, dass jeder Teilnehmer auch eine Antwort schickt, wird mit dem CFP auch eine Frist gesendet, bis zu der eine Antwort vorliegen muss. Antworten, die nach Ablauf der Frist eintreffen, werden vom Manager nicht mehr berücksichtigt. Nach Ablauf der Frist wählt der Manager unter den Geboten das beste aus und schickt dem entsprechenden Teilnehmer eine Nachricht, dass sein Gebot akzeptiert wurde. Alle anderen Teilnehmer erhalten eine Absage. Nach dem Bearbeiten der Aufgabe schickt der ausgewählte Teilnehmer eine von drei möglichen Antworten: *failure* informiert den Manager darüber, dass die Aufgabe nicht erfolgreich bearbeitet werden konnte. *inform-done: inform* signalisiert dem Manager, dass die Aufgabe erfolgreich bearbeitet wurde. Mit *inform-result: done* liefert der Teilnehmer einen Bericht über die erfolgreiche Bearbeitung oder das Ergebnis der Bearbeitung.

Die rekursive Struktur und die Einsatzmöglichkeit in kooperativen Umgebungen macht das CNP interessant für den Einsatz im holonischen Umfeld.

### 3.4.4 Extended Contract Net Protocol

Das *Extended Contract Net Protocol* (ECNP) ist eine Erweiterung zum klassischen CNP. Es wurde von Fischer et al. [Fi97b] zum Einsatz im TeleTruck-System (siehe Abschnitt 3.7.1) entwickelt. Die Erweiterung ist notwendig, weil das klassische CNP keine einfache Möglichkeit bietet mit Aufgaben umzugehen, die die Fähigkeiten der einzelnen Agenten übersteigt. Wie oben schon erwähnt, ist das Zerteilen der Aufgabe in Teilaufgaben dann ein Problem, wenn die Zerteilung nicht vom Entwickler vorgegeben, oder dem Problem immanent ist. Die zentrale Zerteilung ist im generellen Fall NP-hart (es handelt sich um ein Rucksack-Problem). Deshalb ist das ECNP ein Algorithmus zur *dezentralen* Aufgabenzerteilung.

Der erste Teil des ECNP bis zum Bieten des Teilnehmers entspricht demselben Teil im klassischen CNP. Danach unterscheiden sich die beiden Protokolle. Im ECNP sind die *reject*- und *accept-proposal* Nachrichten durch *definitive grant*, *definitive reject*, *temporal grant* und *temporal reject* ersetzt.

Jedes Mal, wenn der Manager ein Gebot empfangen hat, vergleicht er es mit den bisher empfangenen. Ist das neue Gebot besser als das bisher beste, so sendet er dem Bieter ein *temporal grant* und merkt sich das Gebot als das neue beste Gebot. Dem Bieter des bisher besten Gebots sendet er ein *temporal reject*, womit das vorher empfangene *temporal grant* annulliert wird. Ist das neue Gebot schlechter als das bisher beste, so antwortet er einfach mit einem *temporal reject*. Ein Holon, der ein *temporal grant* empfängt, macht eine Sicherungskopie seiner bisherigen Pläne und erstellt einen neuen Plan, der davon ausgeht, dass ihm die Aufgabe zugewiesen wird. Bieter, die ein *temporal reject* empfangen, zuvor aber ein *temporal grant* empfangen haben, stellen ihre vorherigen Pläne wieder her.

Wenn das beste Gebot nicht die gesamte Aufgabe abdeckt, so wird der verbleibende Teil der Aufgabe neu ausgeschrieben und wie zuvor beschrieben verhandelt. Dieser Vorgang wird so lange wiederholt, bis eine Menge von Geboten vorliegt, die die gesamte Aufgabe abdecken. Aus diesen berechnet der Manager dann seinerseits ein Gebot, dass er an den Klienten schickt. Wenn der Klient das Gebot ablehnt, dann schickt der Manager an alle Teilnehmer ein *definitive reject*.

Wird das Gebot akzeptiert, so schickt er ein *definitive grant* an alle Bieter, deren Gebot berücksichtigt wurde (siehe Abbildung 3.20 für den Vorgang als UML Sequenzdiagramm).

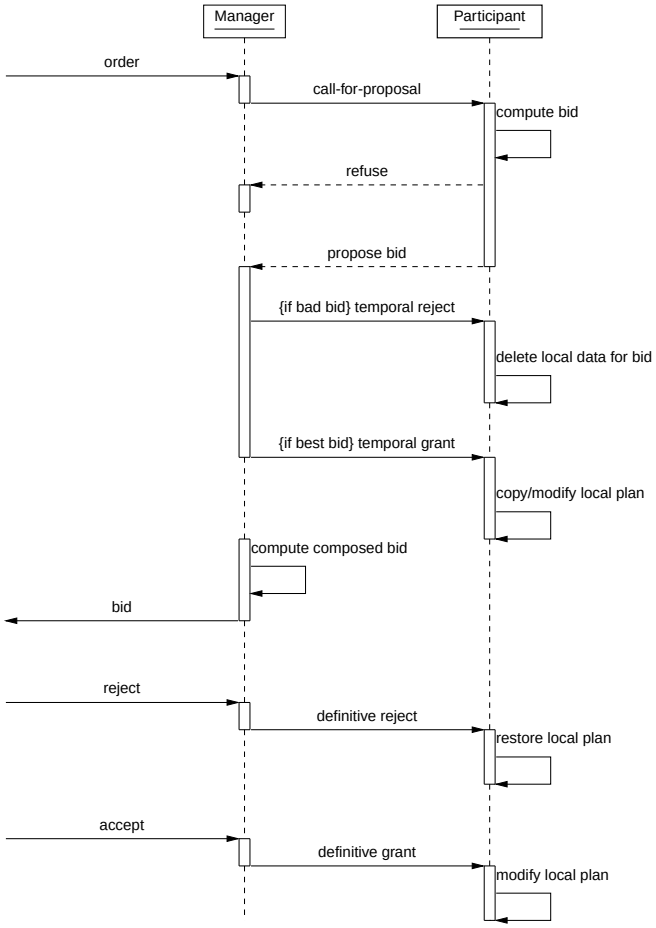



**Abbildung 3.20. The Extended Contract Net Protocol**

## 3.5 Beenden oder Verlassen eines Holonen

Ein Kopfholon kann einen Holon verlassen, ohne einen Nachfolger zu bestimmen. Dazu kann der Kopfholon die Beendigung des Holonen allen Subholonen mitteilen, so dass diese sich vorbereiten können. Das Verlassen des Kopfholonen kann auch abrupt geschehen, z.B. bei einem Ausfall des Kopfholonen. Dann werden die Subholonen nicht vom Verlassen des Kopfholonen benachrichtigt, sondern müssen dies selbst feststellen. Wenn der Kopfholon nicht mehr an Kommunikationsvorgängen teilnimmt, kann das ein Anzeichen für den Ausfall des Kopfholonen sein. Es kann aber auch ein Zeichen für andere Probleme sein, beispielsweise dem Auftreten eines Kommunikationsengpasses. Wegen der verringerten

Ansprechbarkeit ist in jedem Fall die Leistung des Systems beeinträchtigt. Daher ist es für einen Subholonen, der eine solche Beeinträchtigung entdeckt, angebracht, den Holonen zu verlassen. So kann das System sich neu organisieren und seine Leistung wieder optimieren. Da eine Neuorganisierung bei solchen Beeinträchtigungen angebracht ist, ist es nicht sinnvoll die Informationen, die für den Wiederaufbau eines Holonen nach dem Ausfall des Kopfes notwendig sind, beim Zustandekommen des Superholonen an die Subholonen zu verteilen und dort ständig aktuell zu halten. Dies würde zwar dem Credo bei der Entwicklung von Multiagentensystemen nach möglichst hoher Dezentralisierung entsprechen. Der Aufwand zum Verteilen und regelmäßigen Aktualisieren dieser Information ist jedoch höher als die Kosten für eine Neuorganisierung des Systems oder Teilen davon.

Demgegenüber ist das Verlassen oder der Ausfall eines Subholonen weit weniger dramatisch. Ein Subholon ist immer so autonom, dass er entscheiden kann, ob er Mitglied in einem Superholon bleibt oder ihn verlässt (siehe Abbildung 3.17). Deshalb muss die Organisation eines Holonen so konzipiert sein, dass Körperholone ihn jeder Zeit verlassen können. Um das System nicht unnötig zu belasten (z.B. durch das Warten auf Antworten in Kommunikationsprotokollen), sollte der Körperholon sein Verlassen jedoch beim Kopfholon bekannt geben. Dies erlaubt es dem Kopfholonen die Datenstrukturen, die er zur Verwaltung des Holonen unterhält zu aktualisieren. Natürlich kann es auch vorkommen, dass ein Körperholon ausfällt, ohne dem Kopf eine entsprechende Nachricht schicken zu können. Dies kann jedoch, wie beim Ausfall des Kopfholonen, bemerkt werden, wenn der entsprechende Körperholon nicht mehr auf Nachrichten reagiert. Dann entfernt ihn der Kopf ebenfalls aus seiner Datenstruktur und somit aus dem Holon.

Verläßt ein Subholon einen Superholonen oder fällt er aus, so wird die Funktionsfähigkeit des Superholonen eingeschränkt. Führt dies dazu, dass der Superholon seine Aufgabe(n) nicht mehr erfüllen kann, so muss der Kopfholon versuchen, den Ausfall zu kompensieren. Dazu muss ein Holon gefunden werden, der den ausgefallenen Subholonen ersetzt. Kann kein solcher Ersatz gefunden werden, so benachrichtigt der Kopfholon die restlichen Subholone von der Auflösung des Superholonen.

Sowohl die in Abschnitt 3.3, als auch die in diesem Abschnitt beschriebenen Mechanismen können für die Gründung und das Beenden bzw. Verlassen von

Organisationsformen, wie sie in Abschnitt 3.2 vorgestellt wurden, verwendet werden.

## 3.6 Eigenschaften holonischer Systeme

Problemstellungen, die von den in Abschnitt 2.2.1 genannten Eigenschaften eines MAS Gebrauch machen, können durch MAS gut bewältigt werden. Zusätzlich zu den Merkmalen von traditionellen Agenten (siehe Abschnitt 2.1), bieten Holone die Möglichkeit der Rekursion. D.h. Holone können aus anderen Holonen (Subholone) bestehen und selbst Mitglied in einem anderen Holon (Superholon) sein. Diese Erweiterung des traditionellen Konzepts des Agenten passt sehr gut zur Idee der **fast zerlegbaren Systeme** (*nearly decomposable systems*). Der Begriff wurde von Herbert Simon [Si69] eingeführt und steht für die Beobachtung, dass in komplexen Systemen Abhängigkeiten zwischen Subsystemen eine Interaktion zwischen diesen notwendig macht. Diese Interaktionen sind schwach, können jedoch nicht vernachlässigt werden. Die Verbindungen zwischen Subsystemen mit Abhängigkeiten oder Komponenten, sind stärker als die Verbindung zu anderen Subsystemen. Es ist daher angebracht Komponenten nicht weiter zu zerlegen, sondern sie als unzerlegbar zu behandeln. Konzeptionell entsprechen diese Komponenten den Holonen. Sie kapseln abhängige Teilsysteme oder die Bearbeitung von abhängigen Teilproblemen. Eine weitere Beobachtung von Simon ist die, dass komplexe Systeme aus stabilen Zwischenstufen aufgebaut werden sollten. Diese Beobachtungen legen nahe, dass Holone ein wichtiges Werkzeug für die Entwicklung von komplexen Systemen sind.

Eigenschaften eines Systems, die den Einsatz von Holonen anraten (siehe [Ge99]), werden im folgenden beschrieben.

### 3.6.1 Abstrahierte Ausführung

Aktionen können in unterschiedlicher Granularität betrachtet werden. Man kann sie auf verschiedenen Abstraktionsebenen betrachten. Aktionen auf der Makroebene werden vom Kopfholon ausgeführt. Dieser zerlegt und verteilt die für das Ausführen der abstrakten Aktion notwendigen konkreteren Teilaktionen an seine Subholonen. Abstrakte Aktionen und ihre konkreteren Ausbildungen sind an einem Zielgraphen besonders leicht ersichtlich (siehe Abbildung 4.2). Diese Verteilung von konkreteren Teilaktionen kann auch in einem traditionellen MAS

realisiert werden. Aber die Beziehungen zwischen den einzelnen Agenten und der Gruppe müssten zusätzlich dargestellt werden. Ein holonisches System bietet die Fähigkeit zur Einführung neuer Holone für neue Abstraktionsebenen und deren Repräsentation von Hause aus.

### 3.6.2 Hierarchische Struktur

Ein Anwendungsgebiet, das eine hierarchische Struktur aufweist, eignet sich besonders gut für eine holonische Lösung. Unterschiedliche Hierarchieebenen können durch Super- und Subholonen modelliert werden. Deren Beziehungen untereinander geben die hierarchische Struktur des Systems wieder.

### 3.6.3 Zerlegbarkeit von Problemen

Die Zerlegbarkeit bzw. die Verteiltheit eines Problems ist eine sehr wichtige Eigenschaft der Umgebung und eine Voraussetzung für den sinnvollen Einsatz von traditionellen agentenbasierten Systemen. Die aus der Zerlegung resultierenden Subprobleme können an einzelne Agenten zur Bearbeitung verteilt werden. Wie oben bereits gezeigt, sind komplexe Systeme jedoch meist nur *fast* zerlegbar. In hybriden Fällen existieren sowohl zerlegbare, als auch unzerlegbare Teilprobleme. Da Holonen in einer Hierarchie angeordnet sind und Aktionen unterschiedlicher Granularität ausführen können, eignen sie sich auf eine natürliche Art und Weise für die Bearbeitung solcher hybriden Fälle. Durch die Verwendung des Kopfholonen als zentralisierte Instanz können auch zentralistische Algorithmen für die Problemlösung in einem Superholonen eingesetzt werden.

### 3.6.4 Kommunikation

Kommunikation ist notwendig, wenn zwischen Subproblemen eines verteilten Problemlöseversuchs Abhängigkeiten existieren. Diese Abhängigkeiten machen Koordination und daher meist Kommunikation notwendig. Holone bieten Mittel zur effizienten Kommunikation zwischen den Mitgliedern eines Superholonen, weil Kommunikation ein wesentlicher Aspekt für das Umsetzen ihres kooperativen *Teil*-Charakters ist. Ein solches Mittel sind z.B. Gelbe Seiten, die durch den Kopfholonen verwaltet werden (siehe Abschnitt 2.2.3). Ein anderes Mittel ist die Beschränkung der möglichen Kommunikationspartner für die unterschiedli-

chen Aufgabenbereiche in einem Holonen. D.h. ein Subholon weiß, bei welchen anderen Subholonen eine Anfrage wegen einer Information oder der Hilfe bei einer Aufgabe sinnvoll ist. Auf diese Möglichkeit wird in Abschnitt 4.1 weiter eingegangen. Die Kommunikation zwischen Superholonen, oder Agenten generell, kann solche Vorteile nicht bieten.

### 3.6.5 Kooperation

Holone weisen eine starke soziale Komponente auf. Der unbeschränkte Einsatz von Holonen ist nur in einer kooperativen Umgebung möglich. In einer strikt unkooperativen Umgebung (siehe Abschnitt 2.1.5) ist es nicht sinnvoll Holone einzusetzen, da es für sie keine Beweggründe gäbe, größere Beziehungsstrukturen aufzubauen. Wenn in dieser unkooperativen Umgebung kooperative Teilbereiche existieren, können Holone jedoch verwendet werden, um diese Teilbereiche zu modellieren.

## 3.7 Anwendungsbeispiele von Holonen

In diesem Abschnitt werden einige Anwendungen beschrieben, in denen Holonen zum Einsatz kommen. Diese Anwendungen sollen stellvertretend für verschiedene Anwendungsfelder stehen. Die Anwendungen werden beschrieben, ihre holonischen Eigenschaften und die Vorteile des Einsatzes von Holonen erläutert.

In einigen dieser Anwendungsfelder steht die holonische Lösung in direkter Konkurrenz zu Lösungen des *Operation Research* (OR), einem vielerforschten zentralistischen Problemlöseansatz. Es ist schwer OR in Bezug auf die Optimalität der generierten Lösungen zu übertreffen. Wegen der lokal beschränkten Sichtweite der Agenten, laufen Multiagentensysteme Gefahr in lokalen Maxima zu verharren. Bei zentralistischen Algorithmen ist diese Gefahr weit weniger groß. Im Gegenzug sind Lösungen einer zentralen Komponente weniger flexibel und robust, d.h. sie können in geringerem Maß auf Störungen und Ausfälle im System reagieren. Obwohl MAS keine notwendigerweise optimalen Lösungen darstellen, bieten sie die Fähigkeit sich schnell von unvorhergesehenen Ereignissen zu erholen.

### 3.7.1 TeleTruck

TeleTruck ist ein System für holonisches Management von Fahrzeugbeständen im Speditionswesen. Es wurde von Fischer et al. [Fi97a] entwickelt und erfolgreich implementiert. Disposition von Fahrzeugbeständen ist der Prozess des Erstellens eines Zeitplans für eine Menge von Fahrzeugen und deren Fahrer. Alle eintreffenden und angenommenen Aufträge sollen mit minimalen Kosten und maximalem Profit ausgeführt werden. Dieses Problem, das auch Auftragsverteilung (*order dispatching*) genannt wird, ist wegen seiner Komplexität ein interessantes Forschungsproblem, sowohl im Bereich der Operation Research, als auch der Multiagentensysteme. Die Schwierigkeit, mit Dynamik (z.B. dem Ausfall von Fahrzeugen oder Fahrern) und Unsicherheit (z.B. die Dauer eines Auftrags) umzugehen ist eine besondere Herausforderung. Bis zum Jahr 1997 gab es daher keine Softwarelösung, die all diese Teilaspekte berücksichtigte. Oft wurden Zeitpläne durch einem Experten von Hand erstellt.

Der zu verwaltende Fahrzeugbestand besteht meist aus Fahrzeugen verschiedener Typen. Lastwagen und Anhänger können unveränderbaren oder austauschbaren Laderaum besitzen. Container können im Besitz der eigenen Firma, von Partnern oder Kunden sein. Bei den Fahrern muss deren Fahrzeit, die durch Vertrag und Gesetz beschränkt ist, berücksichtigt werden. Es gibt weitere Faktoren im Hinblick auf Fracht, Zeitfenster und Empfänger eines Auftrags, die hier nicht näher behandelt werden sollen.

Prinzipiell können zwei Typen von Problemen im Speditionswesen unterschieden werden. Zum einen das generellere *pick-up-and-delivery*-Problem, bei dem Lieferungen zwischen beliebigen Punkten bewerkstelligt werden müssen. Zum anderen das *vehicle-routing*-Problem, bei dem Lieferungen von einem zentralen Punkt ausgehend zu verschiedenen Zielen transportiert werden müssen. Wird dabei nur ein Fahrzeug betrachtet, so entspricht das Problem dem in der Informatik bekannten Problem des Handlungsreisenden, das bekanntlich NP-vollständig ist. Daher wird es manchmal als Vergleichstest für OR-Lösungen eingesetzt. In [Fi97a] wurde dieser Ansatz gewählt, um die Lösungen, die das TeleTruck-System erstellte, mit denen eines OR-Systems vergleichen zu können.

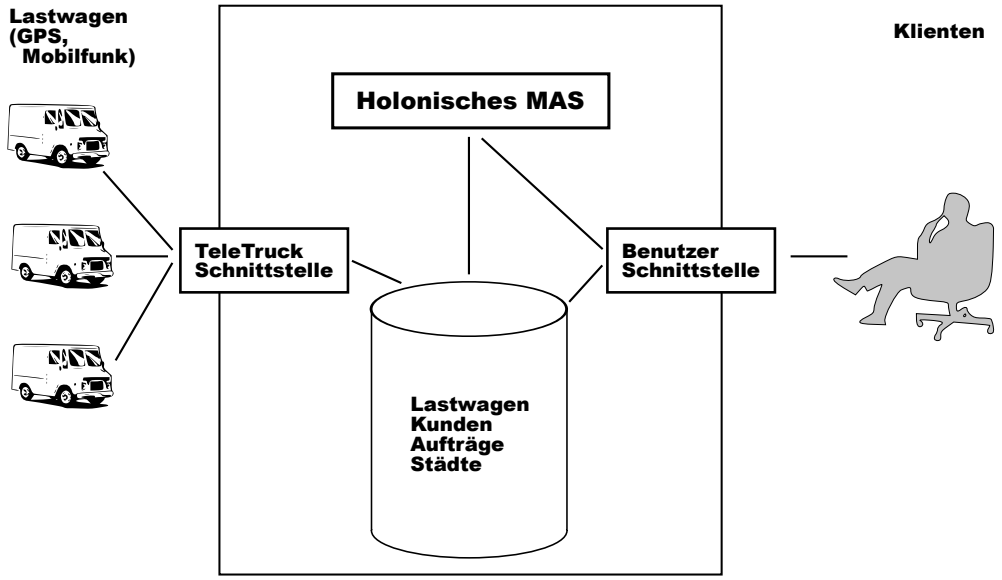
In TeleTruck werden Lastwagen, Anhänger, Ladefläche, Fahrgestell und Fahrer als Agenten modelliert. Dies gibt die tatsächlichen Gegebenheiten wieder. Ein Lastwagen ist keine monolithische Einheit, sondern besteht aus verschiedenen Komponenten (Anhänger, Ladefläche, etc.), die im Laufe der Zeit ihre Beziehun-

gen ändern können. Dies geschieht beispielsweise, wenn ein Anhänger von einem Lastwagen ab- und an einen anderen Lastwagen angekoppelt wird. Um einen Auftrag ausführen zu können, müssen sich die verschiedenen Komponenten zusammenfinden. Der Verband zu einem Ganzen, das Aufträge bearbeiten kann, wird Vehikel genannt. Da alle Komponenten während der Existenz eines Vehikels ausgetauscht werden können, wurde eine persistente Komponente eingeführt, die das Vehikel repräsentiert. Diese Komponente wird *Plan'n'Execute Unit* oder *PnEU* genannt. Das PnEU kann planen und koordinieren. Offensichtlich besitzt dieses Szenario eine holonische Struktur, wie sie in Abschnitt 3.2 beschrieben wurde.

Im täglichen Einsatz besitzt das Dispositionsszenario eine größere Dynamik als bisher beschrieben. Neue Aufträge treffen ein, während andere Aufträge bereits bearbeitet werden. Diese neuen Aufträge müssen in die bereits bestehenden Pläne schnellstmöglich aufgenommen werden. Die verfügbare Information kann unvollständig, unsicher oder einfach falsch sein. Während der Lastwagen beladen wird oder zu einem seiner Zielorte fährt, kann viel passieren. Auch der Verkehr und mögliche Engpässe müssen im System berücksichtigt werden. Diese sich ändernden Umstände können ein Neuplanen erforderlich machen. Der hohe Grad an Dynamik dieser Umgebung behindert zentrale OR-Algorithmen, weil bei großen Veränderungen neu geplant werden muss. Im Gegensatz dazu erlaubt der Einsatz eines MAS bei nur lokalen Veränderungen die Beschränkung auf ein lokales Neuoptimieren der Wegpläne.

Der Kern von TeleTruck ist ein holonisches MAS. Dieses System beinhaltet die Repräsentationen der Transportkomponenten. Es ist verantwortlich für die Zusammenstellung der Komponenten zu Vehikel-Holonen und dem Verplanen der Aufträge, für das die PnEU-Agenten zuständig sind. Es gibt einen PnEU, der die Bildung neuer Holone aus ungenützten Komponenten koordiniert. Alle Komponenten, PnEUs und Vehikel werden in einem weiteren Holon gruppiert, der vom Firmenagenten geleitet wird. Dieser repräsentiert das System, d.h. die Spedition, gegenüber dem Anwender oder ähnlichen Systemen anderer Firmen (siehe Abbildung 3.21).

Für die Wegplanung und zur Ressourcenverteilung werden zwei Protokolle eingesetzt. Zum einen ein einfaches Protokoll zur Ressourcenverteilung (*resource allocation protocol*) (RAP), das aus einer Anfrage- und einer Antwortnachricht besteht. Zum anderen das *extended contract net protocol* (ECNP), das bereits in Abschnitt 3.4.4 beschrieben wurde.




Die Lastwagen sind mit on-board Computern und einer Global Positioning System (GPS) Komponente ausgestattet. Sie kommunizieren über die TeleTruck-Schnittstelle mit dem System. Das holonische MAS verwaltet alle Ressourcen und kommuniziert über die Benutzerschnittstelle mit dem Klienten.

**Abbildung 3.21. Die Architektur des TeleTruck-Systems**

Neue Aufträge werden den PnEU durch das ECNP mitgeteilt. Diese fordern Ressourceninformationen bei ihren Sub-Komponenten an und können dadurch entscheiden, ob die Ressourcen für die Bearbeitung eines Auftrags ausreichen. Wenn das PnEU zu dem Schluss kommt, dass die Ressourcen ausreichen, berechnet es einen Plan und dessen Kosten. Diese Kosten bilden dann das Gebot, das an den Firmenagenten geschickt wird. Falls die Ressourcen zur Bearbeitung des Auftrags nicht ausreichen sollten, wird wegen der fehlenden Ressourcen bei Agenten angefragt, die diese bereitstellen können. Wenn diese Agenten die Ressource direkt zur Verfügung stellen können, so berechnen sie die anfallenden Kosten und melden diese dem anfragenden Agenten. Andernfalls veräußern sie die noch fehlenden Teile des Auftrags wiederum per ECNP. Dieser Vorgang wiederholt sich, bis der Auftrag vollständig abgedeckt ist.

In Abbildung 3.22 [Fi97a] veräußert der Firmenagent einen neuen Auftrag an die PnEUs. Das Vehikel links ist bereits vollständig. Falls es noch freie Ressourcen hat, mit denen es den Auftrag bearbeiten kann, wird es die dabei anfallenden Kosten berechnen und dem Firmenagent melden. Dem Vehikel in der Mitte fehlt die Hänger-Komponente. Das PnEU wird erst versuchen mit der gegenwärtigen

Konstellation den Auftrag abzudecken. Falls das nicht möglich ist, wird es weitere Komponenten hinzuwerben, die die fehlenden Ressourcen bereitstellen. Hierzu wird das ECNP verwendet. Als letztes hat der PnEU auf der rechten Seite der Abbildung überhaupt keine Ressourcen. Daher wird er sofort ein ECNP mit Komponenten beginnen, die Ladefläche zur Verfügung stellen (Lastwagen und Hänger). Da der Hänger nur Ladefläche bietet, wird noch eine Komponente benötigt, die eine Motor besitzt. Der Hänger wird daher beim Lastwagen anfragen. Der Lastwagen erhält zwei Anfragen, eine direkt vom PnEU, die andere vom Hänger. Er kann in beiden Protokollen mitbieten, weil er sicher sein kann, dass nur eines erfolgreich sein wird. In der direkt vom PnEU kommenden Anfrage fehlt ein Fahrer. Der Lastwagen sucht daher nach einem Fahrer und berechnet schließlich die Kosten für die beiden Lösungen. Wenn der Lastwagen ausreichend Ladekapazität besitzt, ist es offenbar günstiger den Lastwagen mit einem Fahrer zu wählen, als zusätzlich den Hänger. Wenn der Lastwagen nicht genügend Kapazität besäße, so würde der Hänger um weitere Ressourcen gebeten werden. Dies führt zu zwei gleichen Geboten, die der PnEU von Hänger und Lastwagen erhält. Er könnte dann zufällig wählen. Wenn der Firmenagent das Angebot des PnEU akzeptiert, verbindet sich dieser mit Komponenten, die mittels ECNP ermittelt wurden zu einem Vehikel. Für die Verwaltung der noch ungenützten Komponenten wird ein neuer PnEU generiert. Nachdem ein Vehikel alle ihm zugewiesenen Aufträge bearbeitet hat, löst er sich auf und der PnEU terminiert.

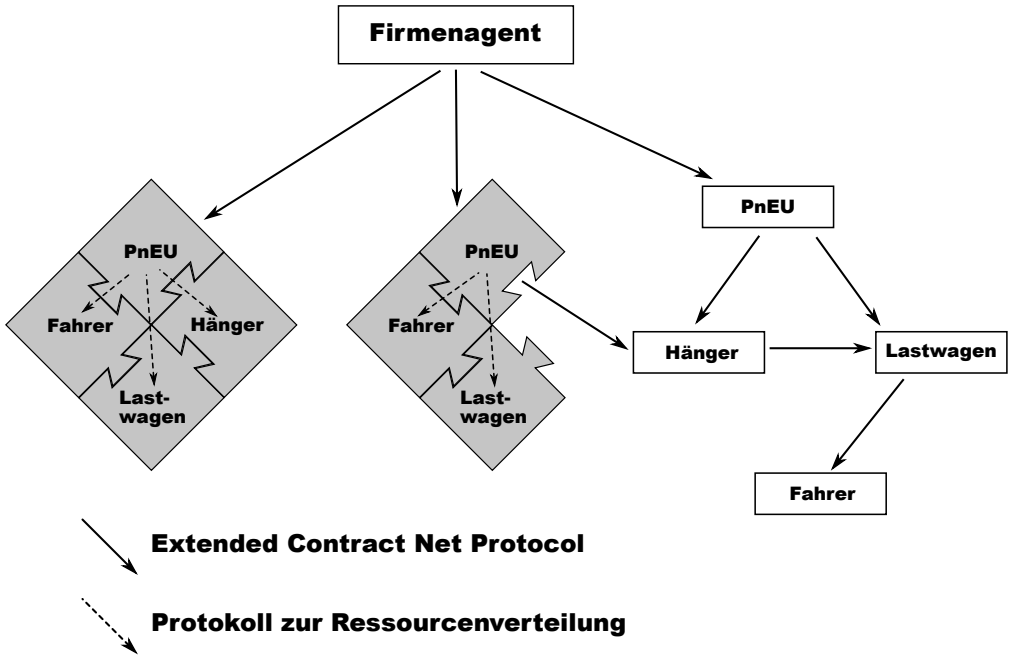



**Abbildung 3.22. Planen in TeleTruck**

Die Initiallösung, die durch Verwendung von ECNP und dem Ressourcenverteilungsprotokoll erstellt wird, wird mit dem *Simulated Trading*-Algorithmus (ST) optimiert. Dieser Algorithmus wurde von Bachem et al. [Ba94] entwickelt. Im Folgenden wird er kurz beschrieben, weil er für die Konkurrenzfähigkeit des TeleTruck-Systems mit OR-Systemen wesentlich ist. ST ist ein randomisierter Algorithmus, der einen Markt implementiert. Im TeleTruck-System wird er verwendet, um einen möglichst guten Wegplan zu generieren. Die Vehikel versuchen ihren Weg- und Zeitplan zu optimieren, indem sie nacheinander Aufgaben kaufen und verkaufen. Der Handel erstreckt sich über mehrere Runden, wobei jede Runde aus mehreren Entscheidungszyklen besteht. In jedem Zyklus senden die Agenten ein Angebot zum Verkauf oder Kauf eines Auftrags. Dabei kommen Heuristiken wie 'kaufe nächstes' und 'verkaufe entferntestes' zum Einsatz. Am Ende jeder Runde versucht der Firmenagent die Kauf- und Verkaufangebot so abzugleichen, dass die Gesamtlösung verbessert wird. Dies resultiert in einem *Hill Climbing*-Algorithmus. Indem der Firmenagenten ebenso schlechtere Lösungen akzeptiert, die Wahrscheinlichkeit für das Akzeptieren von schlechteren Lösungen von Runde zu Runde jedoch reduziert wird, erhält man einen *Simulated Annealing*-Algorithmus. Ein Verharren in lokalen Maxima wird dadurch möglichst reduziert. Maxima, die verlassen werden, werden gespeichert. Wenn der Algorithmus abbricht ohne eine bessere Lösung gefunden zu haben, kann auf das beste bisher gefundene Maximum zurückgegriffen werden. Daher ist ST ein *Anytime*-Algorithmus.

Der ursprüngliche ST-Algorithmus wurde erweitert, um die Neukombination von Komponenten zu ermöglichen. Begründung dafür ist, dass ein Wegplan zwar gut, die Ressourcenverteilung innerhalb des Plans jedoch schlecht sein kann. Dies kann dann der Fall sein, wenn ein großer Lastwagen nicht voll beladen wird, während ein kleiner Lastwagen zusätzliche Ladekapazität benötigt. Eine Handelsrunde wird daher in drei Phasen unterteilt. In der ersten Phase werden Aufträge, wie oben beschrieben verhandelt. In der zweiten Phase können Holone den Austausch von Komponenten anbieten. In der dritten Phase werden wieder Aufträge gehandelt. Nach der dritten Phase gleicht der Firmenagent die Gebote zum Kaufen und Verkaufen und zusätzlich die Angebote zum Tauschen von Komponenten ab. Diese letzte Phase wird benötigt, um zu entscheiden, ob der Austausch von Komponenten in der mittleren Phase die globale Ressourcenverteilung verbessert hat.

In der Einleitung zu diesem Abschnitt wurde erwähnt, dass MAS-Lösungen manchmal in direktem Wettbewerb mit zentral generierten Lösungen, z.B. durch OR, stehen. TeleTruck ist ein solches System. Es ist daher interessant die Qualität der von TeleTruck erzeugten Lösungen mit denen von OR-Algorithmen generierten zu vergleichen. Zu diesem Zweck wurden Vergleichstests verwendet, die von Desrochers et al. [De92] für das Testen deren zentralen Planers entwickelt wurden. Diese Vergleichstests erfordern eine statische Situation, z.B. müssen alle Aufträge im Voraus bekannt sein und dürfen sich nicht mehr ändern. Die Vergleichstests wurden aufgrund der vorhandenen Daten gewählt, die Vergleiche mit zentralen Planern erlauben. Sie zeigten, dass TeleTruck ohne die Optimierung mit ST, d.h. allein unter Verwendung von ECNP 3% bis 74% schlechter als die optimale Lösung ist. Die Qualität der Lösungen befindet sich somit im Bereich von heuristischen OR-Algorithmen. Weitere Experimente zeigten, dass die Verwendung von ST das Ergebnis im Durchschnitt um ca. 12% verbesserte.

Zusammenfassend konnte gezeigt werden, dass die Kombination von ECNP und ST bessere Lösungen generierte als heuristische OR-Algorithmen. Es sollte betont werden, dass die Vergleichstests die Fähigkeiten des TeleTruck-Systems mit dynamischen Umgebungen umzugehen, nicht nutzten. In dynamischen Umgebungen ist von einer weiteren Verbesserung der von TeleTruck generierten Lösungen gegenüber einer OR-Lösung auszugehen.

## 3.7.2 Holonenorientiertes Programmieren

*Although some object-oriented software is reusable, what makes it reusable is its bottom-upness, not its object-orientedness.*
--Paul Graham

Es wurde verschiedentlich angemerkt (z.B. [Sz98] und [WJ00]), dass das Paradigma der objektorientierten Programmierung (OOP) an seine Grenzen gekommen ist, wenn es um das Design und die Implementierung von komplexen Systemen in großem Umfang geht. Da Softwaresysteme in Zukunft immer komplexer werden, scheint ein Wechsel zu anderen Paradigmen angebracht. Ein solches, auf OOP aufbauendes Paradigma ist die komponentenorientierte Programmierung (*component-oriented programming*) (COP) [Sz98]. Es wird eine kurze Einführung in die Idee der Komponenten in der Softwareentwicklung gegeben werden, da dieses Konzept auch im holonischen Umfeld von Interesse ist.

Eine **Komponente** ist eine Sammlung von kooperierenden Objekten. Die Komponente umschließt die Objekte in einer klar definierten Grenze und bietet eine Schnittstelle für die Interaktion mit ihnen durch andere Komponenten oder Objekte. Komponenten kapseln Daten bzw. Zustand und Operationen, die diese Daten in der Art einer Blackbox verändern. Die Objekte innerhalb einer Komponente sind eng miteinander verflochten, während Interaktionen über die Komponentengrenzen hinweg schwach sind. Softwarekomponenten fügen normalerweise das Konzept der dynamisch feststellbaren Schnittstellen, Eigenschaften und Operationen hinzu. In der Programmiersprache Java kann dies durch die Introspektion von JavaBeans erreicht werden. Eine Alternative besteht in der Verwendung der vollen Reflektionsmöglichkeit (*reflection*). In der komponentenbasierten Programmierung geht es darum, auf die Fähigkeiten einer Komponente nur durch ihre Schnittstelle zuzugreifen.

Der wichtigste Vorteil des komponentenbasierten Ansatzes der Softwareentwicklung besteht in der Wiederverwendbarkeit von Komponenten. Diese Eigenschaft wird noch mehr als beim objektorientierten Ansatz betont. Dadurch, dass Komponenten auf ihre Schnittstelle reduzierbar sind und wegen der Möglichkeit diese zur Laufzeit eines Systems festzustellen, kann man Komponenten als kleinste Einheiten der Auslieferung (*single units of deployment*) bezeichnen [WJ00]. Durch die Gruppierung von zusammengehörenden Objekten erlauben Komponenten eine gröbere Wiederverwendung als die von Grund auf neue Kombination von einzelnen Klassen [Tv00]. Ein weiterer Vorteil von Komponenten ist, dass sie alleinstehend entwickelt werden können. Wiederverwendbarkeit verlangt, dass implizite Abhängigkeiten explizit gemacht werden. Definierte Schnittstellen sollen die einzigen Berührungspunkte einer Komponente mir ihrer Umgebung sein. Dadurch kann eine Komponente entwickelt werden ohne das bekannt sein muss, in welchem genauen Umfeld sie später zum Einsatz kommen wird. In einer Testumgebung (*testbed*) kann die Komponente eingehend geprüft und verbessert werden, bevor sie ausgeliefert wird. Desweiteren kann analog zu den Modulen in „Die Parabel von den Uhrmachern“ (S. 17) durch das Zusammenfügen von Komponenten, die als stabile Zwischenstufen dienen, ein großes, komplexes System leichter gebildet werden.

Interaktionen zwischen Softwarekomponenten finden über Methodenaufrufe statt und sind daher rein syntaktisch. Im Gegensatz dazu findet Interaktion zwischen Agenten in einer Agentenkommunikationssprache (*agent communication language*) (ACL) statt. Solche Sprachen, wie beispielsweise KQML, beruhen

gewöhnlich auf der Sprechakttheorie und funktionieren auf der Wissensebene, d.h. semantisch. Der Einsatz einer Hochsprache fördert die Entkopplung der Agenten untereinander. Diese Entkopplung ist in offenen Systemen von Vorteil und kommt dort zum Einsatz.

Es ist weithin anerkannt, dass Interaktion der wichtigste Aspekt eines komplexen Systems ist [WC00]. Softwaresysteme bestehen oft aus vielen miteinander interagierenden Komponenten, die in Threads, Subprozesse oder auf separaten Prozessoren nebenläufig ausgeführt werden. Solche Komponenten koordinieren sich untereinander durch komplexe Protokolle. Die daraus entstehenden Systeme sind typischerweise um Größenordnungen komplexer als Systeme, die eine Funktion in einem einzelnen Thread berechnen. Die meisten realen Anwendung weisen diese komplexen Eigenschaften auf. Erst in den 1990er Jahren (vgl. [Sh93]) hat man realisiert, dass der Agentenbegriff sehr gut auf die Idee der *nebenläufig und autonom interagierenden Einheiten* passt. Dieser Erkenntnis folgend schlug Yoav Shoham [Sh93] das **agentenorientierte Programmieren** (AOP) vor. In diesem Zusammenhang kapseln Agenten ähnlich wie Objekte Daten und Operation auf diesen Daten. Sie unterscheiden sich von Objekten darin, dass Objekte als Abstraktionen für *passive* Einheiten (z.B. ein Haus) gesehen werden können, wohingegen Agenten *aktive* Einheiten abstrahieren. Die Beziehungen zwischen Objekten und Agenten lässt sich durch den Ausspruch „Objekte machen es umsonst. Agenten machen es, weil sie es wollen. [WC00]“ zusammenfassen. Zusätzlich kapseln Agenten mentale Aspekte, z.B. Annahmen und Verpflichtungen und bieten Interaktion auf hohem Niveau [Tv00]. In Abbildung 3.23 werden diese Beziehungen graphisch dargestellt.

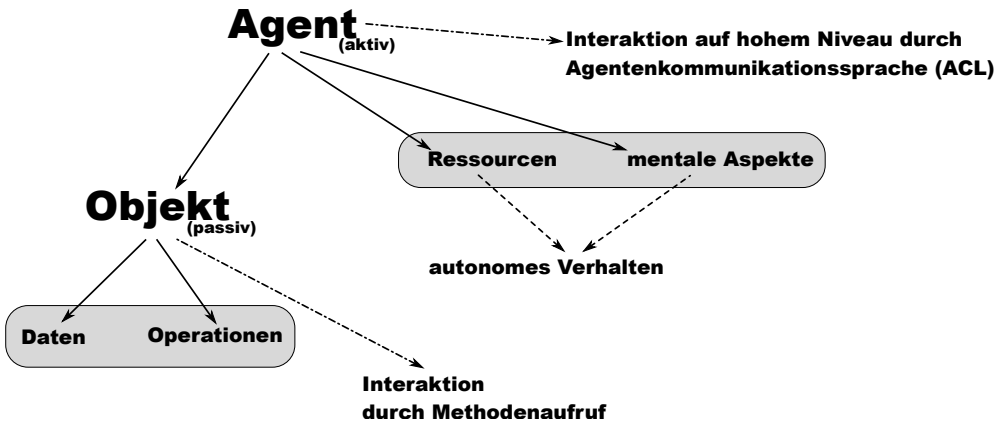



**Abbildung 3.23. Agenten als Erweiterung von Objekten**

In [WJ00] charakterisieren Jennings und Wooldridge einen Agenten im Kontext der agentenorientierten Softwareentwicklung (AOSE), einem Ansatz, der AOP mit einschließt. Tabelle 3.1 stellt komplexe Systeme agentenbasierten Systemen gegenüber [Je00]. Holonen sind ein Hilfsmittel Organisationsformen darzustellen. Daher wurden in der Tabelle entsprechende Veränderungen vorgenommen. Damit wird klar, dass im Umfeld der AOSE Holone in derselben Beziehung zu Agenten stehen wie Komponenten im Sinn des COP zu Objekten. Holone sind sehr gut dazu geeignet **komponentenbasierte agentenorientierte Softwaresysteme** zu erstellen.

**Tabelle 3.1. Vergleich zwischen komplexen und agentenbasierten Systemen**

| **komplexe Systeme** | **agentenbasierte Systeme** |
|---|---|
| Subsysteme | Agentenorganisationen (**Holone**) |
| Komponenten von Subsystemen | Agenten |
| Interaktionen zwischen Subsystemen und Subsystem-Komponenten | Kooperation um gemeinsame Ziele zu erreichen; Koordination der Aktivitäten; Verhandlungen um Konflikte aufzulösen |
| Beziehungen zwischen Subsystemen und Subsystem-Komponenten verändern sich mit der Zeit. Gruppen (*collections*) werden als einzelne zusammenhängende Einheiten behandelt. | Explizite Mechanismen für die Repräsentation und die Leitung von organisatorischen Beziehungen. Strukturen für das Modellieren von Gruppen. (**Holone**) |

Die Idee des **holonenorientierten Programmierens** (HOP) bietet sich daher an und wurde von Gerber et al. [Ge99] bereits umrissen. Es konnten keine aktuellen Referenzen gefunden werden, was darauf hindeutet, dass der Ansatz gegenwärtig nicht weiter erforscht wird. Das mag an der Neuheit des gesamten Feldes des AOSE liegen. Da einige der Ideen aus [Ge99] in der hier vorliegenden Arbeit verwendet werden, soll dieser Ansatz kurz beschrieben werden. Das implementierte System basiert auf der InteRRaP-Architektur von Müller et al. [Mü93] und wurde in der Programmiersprache Oz realisiert. Die holonische Struktur des Systems wird als gerichteter Graph in der obersten Schicht der Architektur, dem *cooperative planning layer* (CPL) repräsentiert. Dieser Graph ist zwischen allen Subholonen eines Superholonen verteilt. Der CPL ist verantwortlich für Kommunikation, Verhandlung und die holonischen Strukturen und stellt unterstützende Funktionen bereit. Jeder Holon wird durch ein Holonenobjekt dargestellt und im CPL des Kopfholonen verwaltet.

# 4 Realisierung

## 4.1 Vorgehensmodell für Analyse und Entwurf

*Our ability to imagine complex applications will always exceed our ability to develop them.*
--Grady Booch

Die Analyse- und Entwurfsphase beim Entwickeln komplexer Systeme dient der Erstellung einer Abstraktion des zu implementierenden Systems. Bisher gibt es keine Methodik, in der explizit Holonen eingesetzt werden, um Teile des Systems umzusetzen. Es wurde lediglich eine Vielzahl an Methoden vorgeschlagen, die sich auf das Konzept des Agenten stützen. Bekannte Methoden sind u.a. Gaia, AAII, SODA und Tropos [Da04], auch MaSE [De01] ist hier zu nennen.

Von diesen Methoden wurden die verbreitetsten, nämlich MaSE, Gaia und SODA näher betrachtet. Es sollte eine Methode ausgewählt werden, die für die Verwendung von Holonen erweitert werden konnte. An die zu erweiternde Methode wurden folgende **Anforderungen** gestellt: (1) es soll einfach sein das holonische Konzept in die bereits vorhandene Methodik einzugliedern, (2) andere in der Entwicklung holonischer Systeme hilfreichen Techniken sollen einsetzbar sein (beispielsweise Zielgraphen), (3) die beiden Phasen Analyse und Entwurf sollen integriert sein, damit kein Paradigmenwechsel stattfinden muss, (4) die Methode soll ausreichend, aber nicht übermäßig komplex sein und (5) die Methode soll in SeSAm umsetzbar sein.

MaSE ist ein sehr umfangreiches Vorgehensmodell, das explizit Zielgraphen verwendet. Auf Grund seines Umfangs ist es jedoch sehr komplex, was es für einen Einsatz im Kontext dieser Arbeit nicht geeignet macht. Bei Gaia handelt es sich um eine Vorgehensweise, die von Wooldridge et al. für die agentenorientierte Entwicklung von Systemen vorgeschlagen wurde [Wo00]. Es ist die erste Methode, die Agentengesellschaften bzw. Agentenorganisationen berücksichtigt hat [Da04]. Die wichtigste Abstraktion ist hier das Konzept der *Rolle*. SODA ist sehr ähnlich zu Gaia. Im Unterschied zu Gaia berücksichtigt SODA auch die nicht-agentische Umwelt. Dies ist für die Betrachtung hier nicht wesentlich. Ein weiterer Unterschied besteht in der Verwendung der **Aufgabe** als wesentliches Konzept bei der Systemanalyse. Dies ermöglicht es, einen Zielgraphen in der Analysephase einzusetzen, um das zu modellierende System zu untersuchen. Der Einsatz eines Zielgraphen wird in SODA nicht explizit erwähnt. Die mögliche

Integration der Verwendung eines Zielgraphen in SODA bringt die Vorteile der Intuitivität und Einfachheit, die die Verwendung dieser Technik für die Entwicklung von holonischen Modellen nahelegt. Zusätzlich kann das in SODA vorhandene Konzept der Gruppe fast direkt auf das Konzept des Holonen abgebildet werden. Durch ein Übergangsmodell zwischen Analyse und Entwurf sind die beiden Phasen auch weitgehend integriert. Die oben angeführten Anforderungen an eine Entwurfsmethodik für holonische Systeme sind durch SODA hinreichend erfüllt.

Im Nachfolgenden wird SODA zu Analyse und Entwurf von agentenbasierten Systemen beschrieben und im Anschluss die Abbildung auf das holonische Umfeld erläutert.

### 4.1.1 SODA

**SODA** (*Societies in Open and Distributed Agent spaces*) ist eine von Omicini [Om00] entwickelte Methodik, die die Beziehungen *zwischen* Agenten (*inter-agent*) im Gegensatz zu deren internen Strukturen betont. Zwei Aspekte sind für diesen Ansatz wichtig: (1) die Gesellschaft, die durch die Interaktionen der einzelnen Agenten entsteht und (2) die Umwelt, in die die Agenten eingebettet sind. Das Verhalten von Agentengesellschaften kann dabei oft nicht auf die Verhaltensweisen der einzelnen Agenten zurückgeführt werden (vgl. *erhöhte Gruppenfähigkeiten*). Daraus erwächst der Wunsch, Agentengesellschaften als vollwertige Teile im Entwicklungsprozess abzubilden. Es sei hier bereits angemerkt, dass diese Agentengesellschaften später als Holone umgesetzt werden. Durch den Bezug auf Aufgaben kann schon sehr bald in der Analysephase eine Unterscheidung zwischen Aufgaben des Individuums und einer Gesellschaft vollzogen werden.

In der **Analysephase** wird das Anwendungsgebiet studiert und modelliert. Die zur Verfügung stehenden Berechnungsressourcen und die technologischen Beschränkungen werden erfasst. Die grundsätzlichen Ziele und die zu bearbeitenden Aufgaben der Anwendung werden herausgearbeitet. Das Ergebnis der Analysephase ist eine Abstraktion auf hohem Niveau, welche gegenseitige Abhängigkeiten im System wiedergibt. In SODA entstehen zu diesem Zweck drei unterschiedliche Modelle.

Das erste in der Analyse entstehende Modell ist das **Rollenmodell**. Die im System ausgemachten Aufgaben sind durch die Verantwortlichkeiten, durch Fähigkeiten, die zu ihrer Bearbeitung notwendig sind und durch benötigte Ressourcen bestimmt. *Verantwortlichkeiten* werden durch Weltzustände, die aus der Bearbeitung einer Aufgabe resultieren, beschrieben. Sie bezeichnen die Funktionen, die eine Rolle zur Verfügung stellt. Es gibt sowohl individuelle, als auch soziale Aufgaben. Für die Bearbeitung von sozialen Aufgaben werden unterschiedliche Fähigkeiten und Ressourcen benötigt. Individuelle Aufgaben benötigen zu ihrer Bearbeitung nur beschränkt Ressourcen und Fähigkeiten. Individuelle Aufgaben werden einer einzelnen Rolle zugeschrieben, wohingegen soziale Aufgaben einer *Gruppe* zugewiesen werden. Eine Gruppe ist eine Menge von sozialen Rollen, die jeweils die Rolle eines einzelnen in einer Gesellschaft beschreiben. Eine soziale Rolle kann dabei durch eine bereits existierende einzelne Rolle erfüllbar sein, oder es wird eine neue Rolle erstellt. Die Aufgabe dieser neuen Rolle ist dann die Teilaufgabe aus der Gruppe, die der sozialen Rolle zugewiesen wurde. Sowohl einzelne Rollen als auch Gruppen werden durch ihre Verantwortlichkeiten definiert. Gruppen werden zusätzlich durch die sozialen Rollen definiert, die in ihnen anzutreffen sind. Rollen und Gruppen stellen zusammenfassend *funktionale Einheiten* dar.

Für das zweite in der Analyse zu definierende Modell, dem **Ressourcenmodell**, werden die Dienste identifiziert, die eine Umgebung der in ihr agierenden Agenten zur Verfügung stellt. Diese Dienste werden für dieses Modell mit einer abstrakten Ressource assoziiert. Mit jeder Ressource werden abstrakte *Zugriffsmodi* verbunden, die die Zugriffsmöglichkeiten auf die Dienste der Ressource angeben. Auf eine Ressource dürfen nur Rollen oder Gruppen mit einer vorgegebenen *Befugnis* zugreifen.

Als letztes in der Analysephase erstelltes Modell dient das **Interaktionsmodell** dazu, die Interaktionen zwischen Rollen, Gruppen und Ressourcen zu definieren. Dabei werden *Interaktionsprotokolle* für Rollen und Ressourcen, und *Interaktionsregeln* für Gruppen festgelegt. Das einer Rolle zugewiesene Interaktionsprotokoll beschreibt die von der Rolle benötigten und zur Verfügung gestellten Informationen, um eine Aufgabe zu bearbeiten. Im Gegensatz dazu definiert das einer Ressource zugewiesene Interaktionsprotokoll die Informationen, die für das Aufrufen des Dienstes einer Ressource notwendig sind. Außerdem beschreibt es die Informationen, die vom Dienst zurückgegeben werden. Interaktionsregeln werden mit Gruppen assoziiert und bestimmen die Interaktionsmöglichkeiten

zwischen den sozialen Rollen und Ressourcen. Diese Herangehensweise ermöglicht eine Entkopplung zwischen den an der Interaktion beteiligten Einheiten. Es wird nur angegeben, welche Information benötigt wird und nicht woher diese Information stammt. D.h. es können z.B. verschiedene Agentenklassen als eine Rolle abstrahiert werden, die dann miteinander interagieren können. Dabei wird nicht auf die konkrete Agentenklasse Bezug genommen.

Nach der Erstellung der vorher beschriebenen drei Modelle ist die Analysephase abgeschlossen. Die anschließende **Entwurfsphase** beschäftigt sich mit der Übersetzung der in der Analysephase gewonnenen Abstraktionen in konkretere Repräsentationen. Diese Repräsentationen sollen dann idealerweise eins-zu-eins in die konkreten Einheiten des fertigen Systems umgesetzt werden können.

Analog zur Analysephase werden in der Entwurfsphase der SODA-Methodologie drei Modelle erstellt. Im **Agentenmodell** werden die individuellen und sozialen Rollen auf zu implementierende Agentenklassen abgebildet. Eine Agentenklasse kann dabei für die Umsetzung mehrerer Rollen zuständig sein und wird definiert durch die von ihr zu bearbeitenden Aufgaben, den Befugnissen für Ressourcenzugriffe und den Interaktionsprotokollen, die den von ihr umgesetzten Rollen zugewiesen wurden.

Im **Gesellschaftsmodell** werden die im Rollenmodell identifizierten Gruppen auf Agentengesellschaften, bzw. -organisationen abgebildet. Wie die Agentenklasse wird auch eine Organisation definiert durch die von ihr zu bearbeitenden (sozialen) Aufgaben, die Menge der Befugnisse für Ressourcenzugriffe, die teilnehmenden sozialen Rollen und den Interaktionsregeln. Um eine soziale Aufgabe bestmöglich zu bearbeiten, ist eine gute Koordination zwischen den einzelnen sozialen Rollen bzw. den Agentenklassen, die diese umsetzen, notwendig. Es ist eine Organisationsform zu wählen, die die notwendigen Koordinationsmechansimen zur Verfügung stellt und unterstützt. Die in Abschnitt 3.2.4 vorgestellten Organisationsformen können hier einen Anhaltspunkt liefern.

Als letztes Modell in der Entwurfsphase von SODA wird das **Umgebungsmodell** erläutert. Hier werden Ressourcen, die in der Analyse identifiziert wurden, auf Infrastrukturklassen abgebildet. Eine Infrastrukturklasse wird definiert durch (1) die Dienste, (2) die Zugriffsmodi, (3) die den Rollen und Gruppen garantierten Befugnissen und (4) den Interaktionsprotokollen, die von den der Infrastrukturklasse zugeordneten Ressourcen herrühren. Das Ergebnis dieses Modells sind

die Dienste, die die Infrastrukturklasse anbieten muss und die Schnittstellen, die aus den Interaktionsprotokollen hergeleitet werden.

Abbildung 4.1 gibt die wichtigsten Elemente der SODA-Methodologie wieder. Das Ressourcen- und das Umgebungsmodell sind für die hier zu erarbeitende Vorgehensweise beim Entwickeln von holonischen Systemen nicht interessant, da diese Teile außerhalb der holonischen Erweiterung und somit im Zuständigkeitsbereich von SeSAm selbst liegen. Der Vollständigkeit halber wurden sie jedoch in der Grafik wiedergegeben.

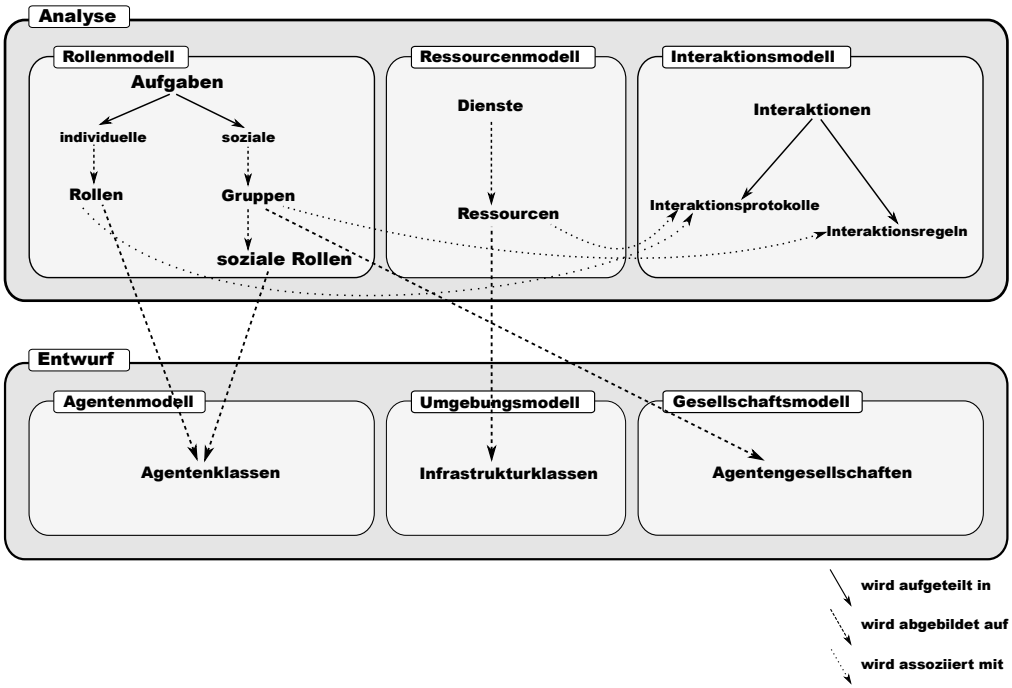



**Abbildung 4.1. Die SODA-Methodologie im Überblick**

Holonen sind eine Kapselung von Organisationsformen. Mit ihnen kann, wie in Abschnitt 3.2 gezeigt wurde, jede beliebige Organisation nachgebildet werden. Somit entsprechen sie den Agentengesellschaften der SODA-Terminologie. In der Analysephase sind Gruppen gleichbedeutend mit Superholonen. Anders als bei SODA kann im holonischen Umfeld eine soziale Rolle wiederum durch eine Gruppe umgesetzt werden. Dies entspricht der rekursiven Struktur der Holarchie.

### 4.1.2 Erweiterung von SODA

Wie bereits erwähnt ist die *Aufgabe* das wichtigste Konzept der SODA-Methodologie. Die Identifikation von Aufgaben kann graphisch komfortabel mit Hilfe eines **Zielgraphen** erarbeitet werden [Je96]. Außerdem unterstützt ein Zielgraph das Finden von Abhängigkeiten zwischen verschiedenen Aufgaben und kann dem Systemanalytiker die Arbeit erleichtern. Diese Abhängigkeiten erfordern Koordination zwischen den an diesen Aufgaben arbeitenden Problemlösern. Anhand des Zielgraphen ist es leicht möglich eine holonische Struktur des betrachteten Systems zu extrahieren. Es ist sinnvoll die Erstellung eines Zielgraphen der restlichen Analysephase voranzustellen. Das nachfolgende Beispiel (Abbildung 4.2) zeigt die Erstellung eines (vereinfachten) Zielgraphen für die TeleTruck-Domäne.

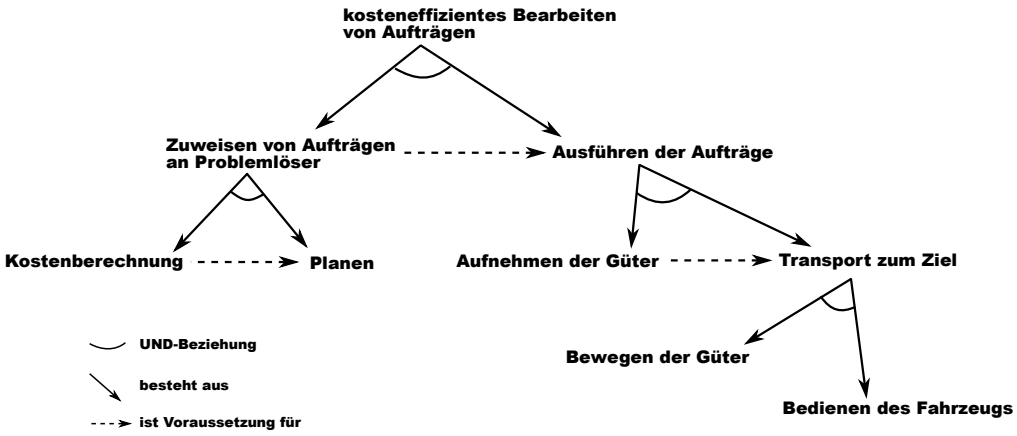



**Abbildung 4.2. Zielgraph für einen Teil der TeleTruck-Domäne**

Gruppen (im Folgenden als *Holone* bezeichnet) sind eine angebrachte Abstraktion für das Zusammenfassen von Problemlösern, wenn zwischen Teilbäumen des Zielgraphen Abhängigkeiten bestehen. Eine solche Abhängigkeit existiert zwischen den Knoten 'Zuweisen von Aufträgen an Problemlöser' und 'Ausführen der Aufträge' in der Abbildung. Es muss in der Analysephase erarbeitet werden, welche Abhängigkeiten als Holon abgebildet und welche durch einzelne Agenten gekapselt werden sollen. Diese Entscheidungen hängen nicht unwesentlich vom Systementwickler ab.

### 4.1.3 Zusammenfassung

Es wird die folgende Vorgehensweise, als Erweiterung der SODA-Methode, für die Analyse und den Entwurf holonischer Systeme vorgeschlagen. In der Aufzählung sind das Ressourcen- und das Umgebungsmodell nicht enthalten, da sie nicht speziell für die Entwicklung von holonischen Systemen, sondern von Systemen im Allgemeinen von Interesse sind.

1. Identifikation von Aufgaben und Abhängigkeiten zwischen diesen mit Hilfe eines Zielgraphen.

2. Unterteilung der Aufgaben in individuelle und soziale Aufgaben. Individuelle Aufgaben werden auf Rollen abgebildet, soziale Aufgaben auf Gruppen. Eine solche Gruppe ist gleichbedeutend mit einem Holonen und besteht aus mehreren sozialen Rollen. Soziale Rollen werden entweder auf bereits existierende Rollen oder neue Rollen abgebildet.

3. Für Interaktionen zwischen Rollen werden Interaktions*protokolle* festgelegt, für die Interaktion zwischen *sozialen* Rollen werden Interaktions*regeln* bestimmt.

4. Die für die Bearbeitung sozialer Aufgaben zuständigen Gruppen werden durch Holonentypen wiedergegeben.

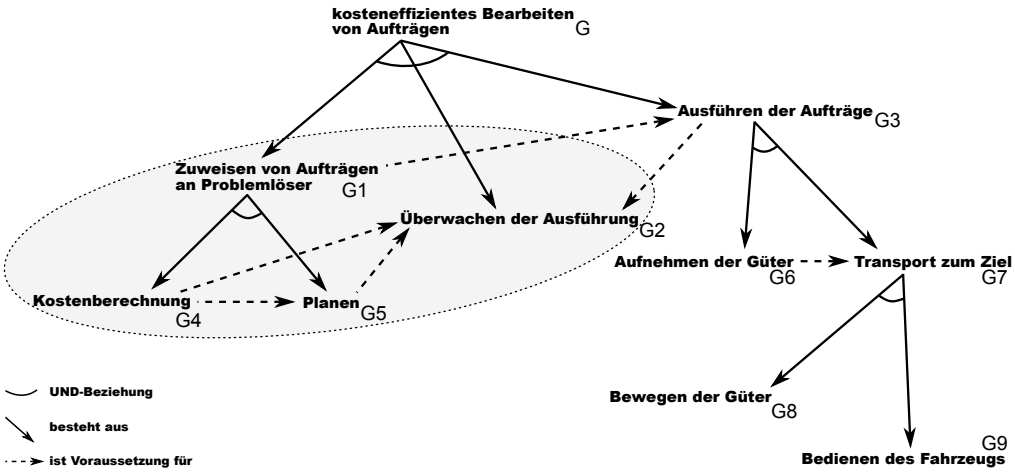
5. Rollen (individuelle und soziale) werden auf Agentenklassen und Holonentypen abgebildet. Dabei können Rollen akkumuliert werden. Die Abbildung von Rollen auf Klassen oder Typen kann anhand der für eine Rolle benötigten Fähigkeiten geschehen. Die Fähigkeiten können statisch zur Modellierzeit, als auch dynamisch zur Laufzeit, festgelegt werden. Letzteres dient bei Superholonen zur Darstellung von erhöhten Gruppenfähigkeiten. Dieselben Fähigkeiten können von unterschiedlichen Agentenklassen und Holonentypen bereitgestellt werden.

### 4.1.4 Beispiel: TeleTruck

Analyse und Entwurf eines holonischen Systems sollen anhand TeleTruck (siehe Abschnitt 3.7.1) beispielhaft gezeigt werden.

## Analyse

In der Analysephase wird ein Zielgraph wie der in Abbildung 4.3 erstellt. Aus diesem lässt sich die Struktur des Problems `kosteneffizientes Bearbeiten von Aufträgen` (G) ablesen. Da das Problem aus der Bearbeitung von drei Teilproblemen besteht, handelt es sich um eine soziale Aufgabe und wird einem Holonen (in SODA entsprechend einer *Gruppe*) zur Bearbeitung zugewiesen. Dieser Holon wird in TeleTruck *Firmenagent* genannt. Er repräsentiert ein Speditionsunternehmen und sieht von außen betrachtet wie ein einzelner Agent aus.




Der markierte Bereich stellt das Aufgabengebiet des PnEU-Holonen dar.

**Abbildung 4.3. Ausführlicher Zielgraph der TeleTruck-Domäne**

Das Teilziel `Überwachen der Ausführung` (G2) hängt durch die Unterziele `Kostenberechnung` (G5) und `Planen` (G6) stark mit `Zuweisen von Aufträgen an Problemlöser` (G1) zusammen. Die beiden Teilziele G1 und G2 von G ergeben daher eine soziale Aufgabe. In TeleTruck können genügend Ressourcen und Fähigkeiten in einem Agenten gebündelt werden, so dass die soziale Aufgabe auch als individuelle Aufgabe umgesetzt werden kann. Tatsächlich wird in TeleTruck dieser Aufgabenbereich durch das PnEU-Holon bearbeitet. Der PnEU-Holone akkumuliert die Rollen (und somit die Fähigkeiten), die für die Bearbeitung von G4, G5, G1 und G2 benötigt werden Falls die notwenigen Fähigkeiten nicht in einem Agenten gebündelt werden können, läge eine Modellierung durch soziale Rollen in einem Holonen nahe.

Für `Ausführen der Aufträge` (G3) ist G1 Voraussetzung. Selbst ist es Voraussetzung für das Teilziel G2. Es existieren somit starke Abhängigkeiten zwischen diesen Teilzielen, die als soziale Aufgabe (S1) zusammengefasst werden. Der dieses soziale Aufgabe bearbeitende Holon besitzt zunächst zwei soziale Rollen: Zum einen die Rolle, die durch den PnEU-Holon besetzt werden kann und die Teilziele G1 und G2 abdeckt. Zum anderen ist es die Rolle, die für das Bearbeiten von G3 benötigt wird. Da die dafür notwendigen Fähigkeiten nicht in einem Holonen allein vorhanden sind, entspricht auch G3 einer sozialen Aufgabe, die aus `Aufnehmen der Güter` (G6) und `Transport zum Ziel` (G7) besteht. G7 kann als individuelle Rolle realisiert werden. Die für diese Rolle notwendigen Fähigkeiten können mit `Ladefläche für Güter` und `Transport von Gütern` bezeichnet werden. G7 ist wiederum eine soziale Aufgabe, da sie nicht von einem Holonen allein bearbeitet werden kann. Die für G8 notwendige Fähigkeit kann als `Zugmaschine für Anhänger` bezeichnet werden. Für G9 ist `Fahren eines Lastwagens` notwendig. G8 und G9 werden als individuelle Rollen modelliert.

Da G7 eine soziale Aufgabe ist, muss überlegt werden, ob die Bearbeitung dieser Aufgabe einem Holonen zugewiesen wird. Die Granularität der Aufgabenteilung ist bei G7 jedoch noch so hoch, dass es nicht sinnvoll erscheint, diese beiden Aufgaben in einem eigenen Holonen zusammenzufassen. Auch bei G3 ist eine Zusammenfassung noch nicht sinnvoll. Auf der Stufe von S1 ergeben sich somit die sozialen Rollen, die zum einen vom PnEU-Holonen ausgefüllt werden, und zum anderen aus den Rollen und Fähigkeiten für G6, G8 und G9 bestehen. Auf dieser Ebene scheint die Etablierung eines Holonen begründet zu sein, da genügend Funktionalität gekapselt wird. In TeleTruck wird dieser Holon *Vehikel* genannt. Er besteht aus dem PnEU (G4, G5 und G2) als Kopf und den Rollen Anhänger (G6), Lastwagen (G6 und G8) und Fahrer (G9).

Diese Herangehensweise erlaubt eine **top-down Modellierung** des Systems. Ausgehend von der abstraktesten Sichtweise eines Systems (im Zielgraph die abstraktesten Ziele), werden die Aufgaben, die das System bearbeiten muss, immer mehr konkretisiert. Im Zielgraphen entspricht dies einem Hinabsteigen zu konkreteren Zielen. Beim top-down Modellieren wird so vorgegangen, dass das globale Verhalten vom Systementwickler vorgegeben und solange verfeinert wird, bis es direkt in Agenten umgesetzt werden kann. Da die Verhaltensweisen der einzelnen Agenten an dieser Vorgabe orientiert sind, müssen sie neu entwickelt werden, wenn sich die Ziele eines top-down entworfenen Systems ändern.

Das ist auf der einen Seite ein Nachteil, auf der anderen Seite ist die Zielorientiertheit aber auch ein Vorteil, weil so ein System entsteht, dass nahe an seinen Anforderungen entwickelt werden kann.

Beim **bottom-up Modellieren** wird das Verhalten der einzelnen Individuen (Agenten) zuerst entworfen. Dieses wird solange angepasst, bis man auf globalerer Ebene das gewünschte Verhalten erreicht. Das Hervorgehen des globalen Verhaltens aus dem Verhalten der Individuen wird **Emergenz** genannt.

Beim Entwickeln holonischer Systeme kann eine *Mischung aus beiden Ansätzen* verwendet werden. Die hierarchischen Organisationsstrukturen werden top-down entwickelt. Das Verhalten der einzelnen Agenten wird weiterhin bottom-up modelliert [Fi03].

Nachdem die benötigten Rollen und Holone identifiziert wurden, werden die **Interaktionen** im System untersucht. Auch hier ist der Zielgraph hilfreich. Abhängigkeiten im Zielgraph machen oft Koordination und somit Interaktion notwendig. Die Analyse der Interaktionen beinhaltet das Festlegen von Interaktionsprotokollen und Interaktionsregeln.

Der oben identifizierte Firmenagent kommuniziert mit externen Auftraggebern und mit den PnEU-Holonen, die die Vehikel-Holonen als Kopf repräsentieren. Wie in Abbildung 3.22 bereits dargestellt wird für die Kommunikation mit den PnEUs das ECNP verwendet. Die Subholone des Firmenagenten kommunizieren nur bezüglich der Zusammenstellung eines neuen oder der Ergänzung eines unvollständigen Vehikels miteinander (= Interaktionsregel). Für Zusammenstellung oder Ergänzung wird das ECNP verwendet. Wie in Abbildung 3.22 auch zu sehen ist, wird innerhalb eines Vehikel-Holonen das in Abschnitt 3.7.1 beschriebene einfache Protokoll zur Ressourcenverteilung verwendet. Weitere Interaktionen betreffen Lenkbefehle des Fahrers an Lastwagen und Anhänger und ähnliches.

## Entwurf

Nach dem Abschluss der Analysephase folgt die Entwurfsphase (siehe Abschnitt 4.1.3). Die identifizierten Rollen werden auf Agentenklassen und Holonen abgebildet. Die Holonen werden auf konkrete Organisationsformen (siehe Abschnitt 3.2.4) abgebildet. Der Vehikel-Holon entspricht im Beispiel

einer Gruppe, da es eine Organisationsform ist, bei der Mitglieder nicht in weiteren Holonen teilnehmen können und eine autoritäre Hierarchie vorhanden ist. Der Firmenagent realisiert eher eine Allianz, da einzelne Lastwagen oder Anhänger nicht der Firma gehören müssen, die er vertritt. Solche Vehikel-Bestandteile können auch von kooperierenden Speditionen oder von Kunden stammen.

Da Holonen nicht direkt modelliert werden, sondern eine erst zur Laufzeit des Systems entstehende Struktur sind, ist es nicht möglich, Rollen direkt auf Holone abzubilden. Für die Abbildung wird das Konzept der **Fähigkeit** verwendet. Anstatt einer Agentenklasse eine Rolle zuzuweisen, werden ihr Fähigkeiten zugeordnet. Besitzt ein Agent alle für eine Rolle geforderten Fähigkeiten, so kann er die Rolle einnehmen. Bei der Bildung eines Superholonen werden die Fähigkeiten des Kopfholonen, der den Superholonen repräsentiert, entsprechend den Fähigkeiten des Superholonen erweitert (siehe Abschnitt 3.1.6). Der Kopfholon kann nun, stellvertretend für den Superholonen, Rollen einnehmen, die diese Fähigkeiten voraussetzen.

In realen Systemen müssen Information über die Fähigkeiten von Agenten erst angefordert werden. Für die Modellierung in SeSAm sind Fähigkeiten von außen sichtbare Attribute einer Agentenklasse bzw. eines Agenten. Auf diese Weise werden die eigentlich notwendigen Kommunikationsvorgänge abgekürzt.

Die Verwendung von Fähigkeiten, zusammen mit einem Wert für die Qualität der Fähigkeit (bzw. des angebotenen Dienstes), erlaubt neben der Abbildung von Rollen auf Agentenklassen und Holonen auch die Darstellung von *Lernen*, *Verbessern* und *Degeneration*. Bei einem System, dessen Verhalten und Leistung optimiert werden soll, kann das Konzept der Fähigkeit daher auch Verwendung finden.

Mit dem Abschluss der Entwurfsphase können die gewonnenen Erkenntnisse in SeSAm umgesetzt werden.

## 4.2 Anforderungen an die Implementierung

Aus den in Kapitel 3 und Abschnitt 4.1 erarbeiteten Konzepten und Methoden können die folgenden Anforderungen an die Implementierung einer Erweiterung

für das Entwickeln und Modellieren von holonischen Systemen in SeSAm extrahiert werden.

- Das Vorgehen in der Modellierung sollte dem Vorgehen in Analyse und Entwurf entsprechen. Damit ist insbesondere das Beginnen der Modellierung auf einer hohen Abstraktionsebene des Systems und das schrittweise Konkretisieren gemeint (**top-down Modellierung**). Dadurch soll ein nahtloser Übergang von Analyse und Entwurf hin zur Modellierung ermöglicht werden.

- Eine **Zuweisung von Rollen** aus dem Vorgehensmodell für Analyse und Entwurf an Agentenklassen und Holonentypen soll leicht möglich sein. Hierfür wird das Konzept der *Fähigkeit* verwendet.

- **Interaktionsregeln** (siehe Abschnitt 4.1.3) zwischen Subholonen sollen definierbar sein.

- Die Einführung von Holonen soll **keine weitreichenden Veränderungen** im SeSAm-Code nach sich ziehen. U.a. ist dies bei einer zukünftigen Erweiterung der holonischen Unterstützung von Vorteil. Die Modellierung von Holonen soll sich nicht von der Modellierung normaler Agenten unterscheiden, damit der Modellierer nicht umlernen muss.

- Es ist von Vorteil im Modell **zwischen normalen Agenten, Subholonen und Kopfholonen unterscheiden** zu können. Z.B. kann dadurch das Finden von bereits existierenden Superholonen vereinfacht werden. Aber auch das Entdecken von Holonen, die noch eine Rolle in einem Superholonen einnehmen können, kann so erleichtert werden.

- Die **Verwaltung eines Superholonen** durch den Kopfholonen soll unterstützt werden.

- Die **Realisierung verschiedener Organisationsformen** (siehe Abschnitt 3.2.4) soll unterstützt werden. Hier ist eine Erstellung der später im System möglichen Formen bereits zur Modellierzeit hilfreich. Eine schablonenhafte Vorgabe von Organisationsformen, die zur Laufzeit mit konkreten Holonen ausgefüllt wird, ermöglicht die Verwendung unterschiedlicher Organisationsformen, ohne diese zur Laufzeit unter den Agenten verhandeln zu müssen.

- Die **hierarchische und rekursive Struktur** eines holonischen Systems soll vollständig umsetzbar sein. Subholonen können für andere Subholonen als Superholon fungieren. Der Zustand eines Superholonen, der sich aus den Zuständen seiner Subholonen herleitet, soll einsehbar sein.

- Die Beziehungen der einzelnen Holonen (**Holarchie**), die sich zur Laufzeit des Systems ergeben, sollen beobachtbar sein. Dies ist bei der Fehlerfindung und der Optimierung hilfreich.

- Das **Gründen, Verlassen und Beenden eines Superholonen** (siehe Abschnitt 3.3 und 3.5) soll unterstützt werden. Die dafür nötige Kommunikation soll leicht modellierbar sein. Der Zustandsautomat für das Rollenspiel soll leicht umsetzbar sein.

- Ein einzelner Holon soll **Mitglied in mehreren Superholonen** sein können. Die für die Realisierung der Rollen in den unterschiedlichen Superholonen nötigen Verhaltensweisen sollen übersichtlich modelliert werden können.

- Die gleichzeitige Mitgliedschaft in verschiedenen Superholonen verlangt unterschiedliches Verhalten pro Mitgliedschaft. Das Modellieren von **parallelen Verhaltensweisen** soll unterstützt werden.

- Für die Kommunikation zwischen Subholonen sollen **Kommunikationsprotokolle** zur Verfügung gestellt werden.

- Um die hierarchische Struktur und die Möglichkeit der abstrahierten Ausführung eines holonischen Systems besser ausnutzen zu können, wäre es hilfreich **Pläne zerlegen** und einzelnen Subholonen zuweisen zu können.

Es sei bereits hier erwähnt, dass die beiden letzten Punkte nicht umgesetzt wurden. Sie sind für einen zukünftigen Ausbau der holonischen Erweiterung von SeSAm vorgemerkt (siehe Kapitel 6).

## 4.3 Das Holonic Augmentation Plugin

Die holonische Erweiterung von SeSAm wird mit Hilfe eines Plugins erreicht (*Holonic Augmentation Plugin*) (HAP). Nach Einbindung des Plugins stehen in SeSAm drei Features zur Verfügung, die in den nachfolgenden Abschnitten beschrieben werden. Die durch das Plugin bereitgestellten Datentypen (siehe

Abschnitt A.1) und Primitive (siehe Abschnitt A.2 und A.3) werden im Anhang aufgelistet und erläutert.

### 4.3.1 Das HolonAdminFeature

Dieses Feature wird der *Welt* zugewiesen. Es ist zentral für die **Persistenz** von Daten, die nicht einzelnen Agentenklassen zugeordnet werden können. Beim Speichern eines Modells, das holonische Funktionalität verwendet, muss eine Welt existieren und diese das HolonAdminFeature besitzen. Ansonsten gehen die globalen Daten beim Speichern verloren, was ein holonisches Modell für gewöhnlich unbrauchbar macht. Außerdem ist beim Erstellen eines holonischen Modells darauf zu achten, dass nur eine Weltklasse definiert wird.

Die angesprochenen globalen Daten, die **Informationen der Holonentypen** (ein Holonentyp entspricht einer vorgegebenen Organisationsform), können über das HolonAdmin-Panel bearbeitet werden. Dieses Panel wird in der Weltklasse nach Zuweisen des Features angezeigt. Über einen Button gelangt man dann zur Auswahl der Holonentypen (Abbildung 4.4). Hier können neue Typen erstellt und vorhandene entfernt oder bearbeitet werden.

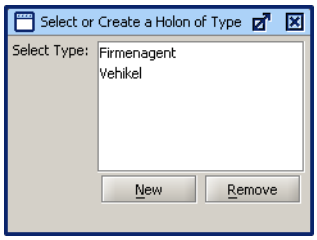
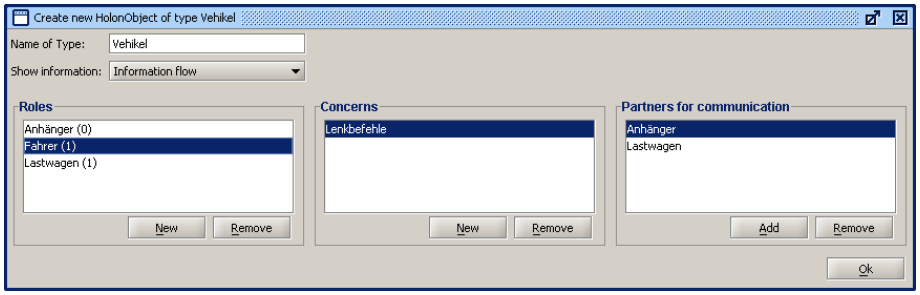



**Abbildung 4.4. Auswahldialog für Holonentypen**

Für die Erstellung eines neuen Holonentypen erscheint ein Dialogfeld (der **HolonTypeEditor**), in dem Rollen, ihre Beziehungen untereinander (Abbildung 4.5) und die benötigten Fähigkeiten (Abbildung 4.6) definiert werden können. Der Kopfholon wird gegenwärtig nicht explizit modelliert. Standardmäßig wird ein Name für den Typ gewählt, der systemweit eindeutig ist. In der linken Liste (`Roles`) werden alle im Holonentyp deklarierten Rollen angezeigt. Die Zahlen in Klammern hinter den Rollennahmen geben die zulässige Höchstzahl an Agenten an, die diese Rolle einnehmen dürfen (*Kardinalität*). Dabei steht 0 für eine unbeschränkte Anzahl. Sobald eine Rolle erstellt oder selektiert wird,

werden zusätzliche Optionen in den beiden rechten Listen zugänglich. Die in diesen beiden Listen angezeigten Informationen beziehen sich immer auf die in der ersten Liste selektierte Rolle. Die in den drei Listen anzuzeigenden Informationen werden durch die Auswahlbox *`Show information`* selektiert. Es stehen *`Information flow`* und *`Needed Skills`* zur Auswahl.




Der Agent, der die Fahrer-Rolle einnimmt kommuniziert mit den Agenten, die die Anhänger- und Lastwagen-Rollen einnehmen, wegen der Umsetzung der Lenkbefehle.

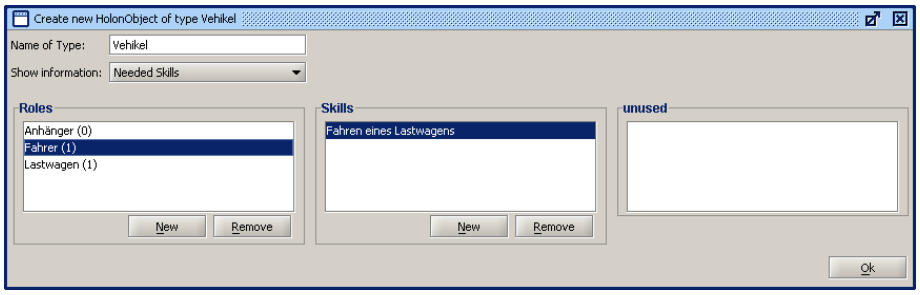
**Abbildung 4.5. Spezifizieren von Interaktionsregeln im HolonTypEditor**

In Abbildung 4.5 ist ersichtlich, wie *Interaktionsregeln* definiert werden können. Durch Auswahl des Eintrags *`Information flow`* wird dieser Eingabemodus aktiviert. Zur ausgewählten Rolle werden im mittleren Feld 'Concerns' erstellt. Ein Concern gibt den *Zweck* eines Kommunikationsvorgangs wieder. Für jedes Concern können im rechten Feld Rollen ausgewählt werden, die bei der Kommunikation wegen dem Concern angesprochen werden sollen. Der Kopfholon unterliegt gegenwärtig nicht diesen Beschränkungen. Er kann unter Angabe eines beliebigen Concerns mit den angesprochenen Subholonen kommunizieren.

Es ist zum gegenwärtigen Stand der Implementierung von HAP nicht möglich, den verschiedenen Interaktionsvorgängen innerhalb eines Holonen Protokolle zuzuweisen. In Kapitel 6 wird auf diese Möglichkeit eingegangen.

In Abbildung 4.6 ist zu sehen, dass durch die Auswahl von *`Needed Skills`* die Möglichkeit gegeben wird, die Fähigkeiten anzugeben, die von einem Agenten verlangt werden, wenn er die Rolle einnehmen möchte. Es ist zu beachten, dass die Fähigkeiten `oder`-Verknüpft sind. Das Vorhandensein *einer* der für eine Rolle benötigten Fähigkeit im Agenten oder Holonen reicht aus, um für das Einnehmen der Rolle legitimiert zu sein. Wird keine Fähigkeit angegeben, so kann jeder Agent die Rolle einnehmen. In einer späteren Version von HAP

soll die Modellierung der benötigten Fähigkeiten in einem Und-Oder-Graph erfolgen und somit komplexere Bedingungen möglich sein.




Um die Rolle des Fahrers einzunehmen, wird die Fähigkeit einen Lastwagen fahren zu können, benötigt.

**Abbildung 4.6. Angeben von Fähigkeiten im HolonTypeEditor**

Die auf diese Weise definierten Holonentypen können zur Laufzeit des Systems durch Kopfholone mit Hilfe des Primitives `CreateHolonObjectForHolonType` instanziiert werden (siehe Abschnitt A.2).

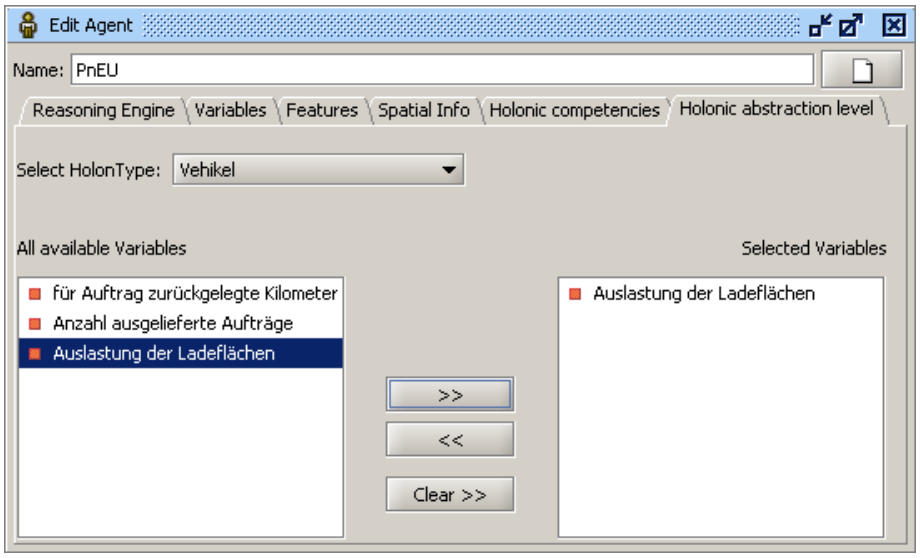
Der Holarchie-Browser steht ebenfalls nach der Zuweisung des HolonAdminFeatures zur Verfügung. Er wird in Abschnitt 4.3.4 gesondert beschrieben.

### 4.3.2 Das HeadholonFeature

Das HeadHolonFeature muss Agentenklassen zugeordnet werden, deren Instanzen potentiell Kopfholone werden können. Da alle Kopfholone auch immer Subholone sind, wird einer Agentenklasse bei Zuordnung des HeadHolonFeatures automatisch das SubHolonFeature zugeordnet, wenn dies noch nicht geschehen ist.

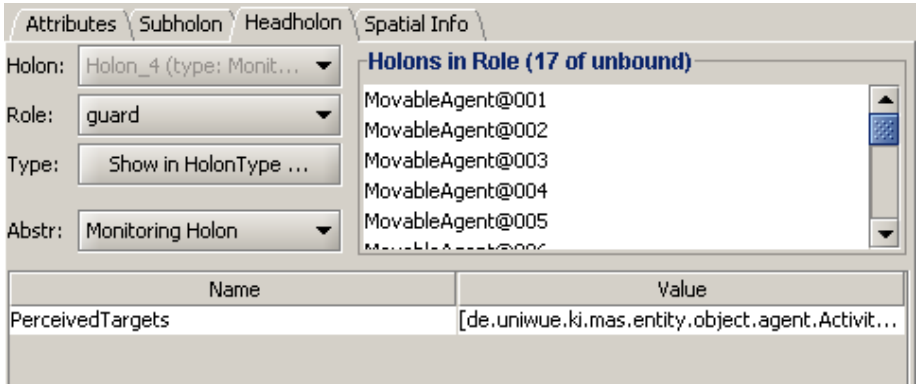
Nach der Zuordnung des Features erscheint in dem Fenster der Agentenklasse das *`Holonic abstraction level`*-Panel (siehe Abbildung 4.7). Hier können in der Agentenklasse definierte Variablen ausgewählt werden, die den Zustand des Superholonen, in dem eine Instanz dieser Klasse Kopf sein kann, wiedergeben. Diese Liste von Variablen wird Abstraktionsebene genannt, weil in ihr Daten aus den Subholonen aggregiert und abstrahiert werden können. Dieses Zusammenfassen ist Aufgabe des Modellierers. Zur Laufzeit kann die Abstraktionsebene eines Superholon angezeigt werden (siehe Abbildung 4.8). So können wichtige Informationen über den Zustand eines Superholonen beob-

achtet werden. Es ist möglich, für jeden Typen eine separate Liste von Variablen anzulegen. Hierfür wird in der Auswahlbox *Select HolonType* der entsprechende Typ ausgewählt.




Das *Holonic abstraction level*-Panel erlaubt das Zuweisen und Entfernen von Variablen einer Abstraktionsebene.

**Abbildung 4.7. Das *Holonic abstraction level*-Panel**




Die angezeigten Daten stammen von dem unter Abschnitt 5.1.1 vorgestellten Modell.

**Abbildung 4.8. Das Headholon-Panel**

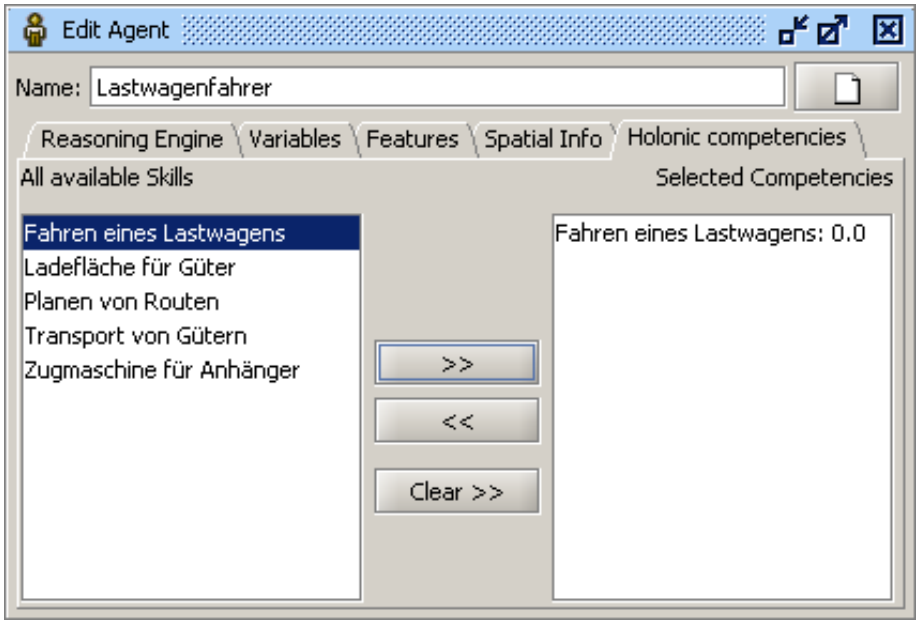
Im **Headholon-Panel** (siehe Abbildung 4.8) werden Informationen über einen selektierten Kopfholonen zur Laufzeit angezeigt. Wenn er Kopf von mehreren Superholonen ist, so bietet die Dropdown-Box *Holon* die Möglichkeit einen Superholonen auszuwählen, dessen Informationen angezeigt werden sollen. Die

Dropdown-Box *Role* erlaubt die Auswahl einer Rolle. Die Subholonen, die diese Rolle gegenwärtig einnehmen werden in der Liste rechts (*Holons in Role*) angezeigt. Durch Betätigen des Buttons *Show in HolonTypeEditor* wird die interne Struktur des Superholonen im HolonTypeEditor angezeigt. Mit der darunterliegenden Dropdown-Box ist es möglich, die anzuzeigende Abstraktionsebene zu wählen. Die Variablen der Abstraktionsebene und deren Werte werden im darunterliegenden Bereich angezeigt. *Holons in Role* gibt als zusätzliche Information an, wieviele Agenten die Rolle einnehmen und welche Kardinalität für die Rolle vorgegeben wurde.

Die von diesem Feature bereitgestellten Primitive werden in Abschnitt A.2 aufgeführt und erläutert.

### 4.3.3 Das SubholonFeature

Im SubholonFeature wird Funktionalität bereitgestellt, die für die Eingliederung einer Agentenklasse in ein holonisches System notwendig sind. Agentenklassen, die nicht die Aufgaben eines Kopfholonen übernehmen müssen, benötigen nur das SubholonFeature.

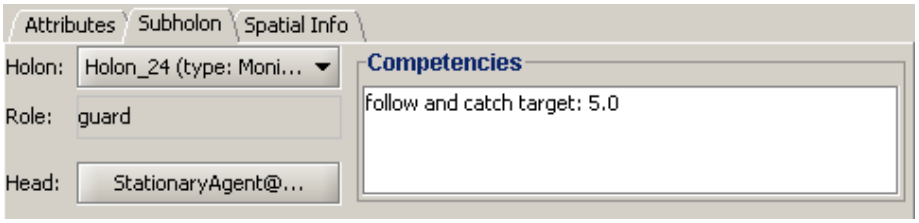



Das *Holonic competencies*-Panel bietet die Möglichkeit Fähigkeiten an eine Agentenklasse zuzuweisen.

**Abbildung 4.9. Zuweisung von Fähigkeiten an eine Agentenklasse in SeSAm**

Nach der Zuweisung des Features an eine Agentenklasse wird das *Holonic competencies*-Panel angezeigt (siehe Abbildung 4.9). Hier können der Agentenklasse Fähigkeiten zugewiesen werden. Die zur Verfügung stehenden Fähigkeiten (*Skills*) wurden im HolonTypeEditor deklariert. Im linken Bereich des Panels wird eine Liste aller Skills angezeigt. Im Beispiel wird der Agentenklasse *Lastwagenfahrer* die Fähigkeit `Fahren eines Lastwagens` zugewiesen. In der Agentenklasse wird der Skill mit einem Wert assoziiert, der die Qualität (*Quality*) der Fähigkeit angibt (siehe Abschnitt 4.1.4). Ein solches Skill/Quality-Tupel wird **Competency** genannt. Der Wert für die Qualität einer Competency kann durch einen Doppelklick geändert werden.

In einem Multiagentensystem sollten die einzelnen Agenten wissen, welche Fähigkeiten sie besitzen. Dies kann durch Nachdenken bzw. Schließen erfolgen, oder vom Systementwickler vorgegeben werden. In dem hier beschriebenen Modellierungsansatz werden die Fähigkeiten der Agentenklassen zunächst statisch vorgegeben. Mit den Primitiven `AddOrChangeCompetency` und `RemoveCompetency` können *Competencies* zur Laufzeit hinzugefügt, verändert oder entfernt werden. Dies ist für die Modellierung der erhöhten Gruppenfähigkeit eines Holonen wichtig (siehe Abschnitt 3.1.6).




Das Subholon-Panel zeigt Laufzeitinformationen zu einem Subholonen an. Die angezeigten Daten stammen von dem unter Abschnitt 5.1.1 vorgestellten Modell.

**Abbildung 4.10. Das Subholon-Panel**

Zur Laufzeit einer Simulation bietet das **Subholon-Panel** Informationen zum gegenwärtigen Zustand eines Subholonen (siehe Abbildung 4.10). Die Dropdown-Box *Holon* erlaubt die Selektion des Superholonen, über den Informationen zur Mitgliedschaft des aktuellen Subholonen angezeigt werden sollen. Das *Role*-Feld zeigt an, welche Rolle der Subholon in dem unter *Holon* selektierten Superholonen einnimmt. Durch Drücken des *Head*-Buttons werden Informationen zum Kopfholonen des selektierten Superholonen angezeigt. In der Liste rechts

werden die Competencies des Subholonen aufgelistet. Wegen der Möglichkeit einer Änderung ist diese Information auch zur Laufzeit interessant.

Die vom SubholonFeature bereitgestellten Primitive werden unter Abschnitt A.3 erläutert.

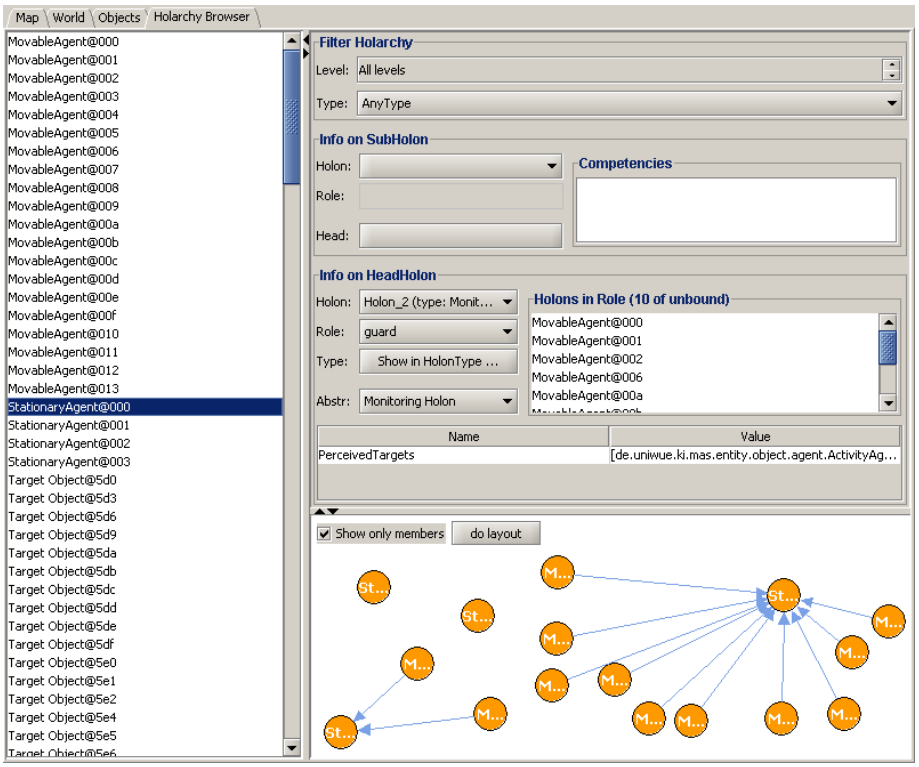
### 4.3.4 Laufzeitanalyse

Neben dem Headholon-Panel und dem Subholon-Panel, die in den vorangehenden Abschnitten erläutert wurden, ist der **Holarchie-Browser** (siehe Abbildung 4.11) das wichtigste Werkzeug für die Untersuchung und Darstellung der Holarchie zur Laufzeit einer Simulation. In der großen Liste links werden standardmäßig alle Agenten im System angezeigt. Durch die Bedienelemente unter *Filter Holarchy* kann dies verändert werden. Der Spinner *Level* erlaubt die Auswahl der Holarchieebene, auf der sich Holone befinden müssen, damit sie in der Liste angezeigt werden. Ebene 0 ist dabei die Ebene, auf der sich nur Subholonen befinden. Ab Ebene 1 bestehen alle Ebenen aus Kopfholonen, die einen Superholonen repräsentieren, der aus Holonen der vorhergehenden Ebene gebildet wurde. Die Dropdown-Box *Type* erlaubt das Filtern der Liste anhand des Holonentyps. Dann werden nur die Kopfholonen des ausgewählten Typs in der Liste angezeigt.

Darunter befinden sich Anzeigeelemente für Daten eines Subholonen (*Info on SubHolon*) und eines Kopfholonen (*Info on HeadHolon*). Dabei handelt es sich um das Subholon-Panel und das Headholon-Panel. Diese beiden Panels werden in den Abschnitten 4.3.2 bzw. 4.3.3 erläutert. Im Holarchie-Browser weisen diese Panels erweiterte Funktionalität auf, die das Traversieren der Holarchie erlaubt. Durch Betätigen des *Head*-Buttons im Subholon-Panel wird zur Ansicht des Kopfholonen des angezeigten Subholonen gewechselt. Umgekehrt kann durch die Auswahl eines Eintrags in der Liste *Holons in Role* des Headholon-Panels zur Ansicht des entsprechenden Subholonen gewechselt werden.

Im unteren rechten Bereich wird die Holarchie als Graph dargestellt. Pfeile gehen von Subholonen zu Kopfholonen. Durch Auswahl der Option *Show only members* werden im Graphen nur Agenten dargestellt, die Mitglieder in einem Superholonen sind. Die Knoten im Graphen können neu angeordnet werden, um die Übersichtlichkeit zu erhöhen. Durch Selektion eines Knotens wird der ent-

sprechende Agent in der Liste links ausgewählt und seine Daten angezeigt. Für die Erzeugung und Darstellung des Graphen wurde auf [JG05] zurückgegriffen.




Der Holarchie-Browser ermöglicht die Untersuchung der Holarchie zur Laufzeit. Es werden Daten der Sub- und Kopfholonen angezeigt und die Beziehungen der Holonen untereinander in einem Graphen visualisiert. Die dargestellten Daten stammen aus einer Simulation des Modells, das unter Abschnitt 5.1.1 vorgestellt wird.

**Abbildung 4.11. Der Holarchie-Browser**

## 4.4 Modellierung eines holonischen Systems

Nachdem die für die Modellierung eines holonischen Systems in SeSAm zur Verfügung stehenden Werkzeug vorgestellt wurden, soll nun die Entwicklung des Beispielszenarios 'TeleTruck' fortgesetzt werden. Wie in Abschnitt 4.3.1 gezeigt wurde, können die in Abschnitt 4.1.4 ermittelten Informationen zur Struktur des Systems mit Hilfe von HAP modelliert werden. Dies geschieht hauptsächlich durch den *HolonTypeEditor* für die Definition von Interaktionsregeln und Fähigkeiten und das *Holonic competencies*-Panel für das Zuweisen von Fähigkeiten an Agentenklassen.

Die Beschreibung der Vorgehensweise bleibt abstrakt und soll die bei der Modellierung holonischer Systeme gewonnenen Einsichten vermitteln. Das System wird zuerst weiter top-down entwickelt. Die konkrete Ausarbeitung der Verhalten der einzelnen Agentenklassen, d.h. die bottom-up Phase, wird hier nicht erläutert.

In den Beispielen oben wurden bereits zwei Agentenklassen des Systems eingeführt: *PnEU* und *Lastwagenfahrer*. Die Agentenklassen *Anhänger* und *Lastwagen* sind weitere logische Bestandteile des Systems. Desweiteren wird die Klasse *Speditionsagent* als Kopf des Firmenagent-Holonen eingeführt.

Jeder PnEU-Agent ist automatisch Subholon im Firmenagent-Holonen. Der Speditionsagent ist immer Kopf des Firmenagent-Holonen. Beide Agentenklassen müssen daher nicht darüber nachdenken können, ob sie in einem Superholonen Mitglied werden möchten. Für diese Klassen ist ein **Zustandsautomat für das Rollenspielen** (siehe Abschnitt 3.3) nicht notwendig. Instanzen der übrigen Klassen können dynamisch Mitglied in einem Vehikel-Holonen werden oder ihn wieder verlassen (wenn die Aufträge ausgeführt wurden). Daher muss bei ihnen ein Zustandsautomat modelliert werden. Als Vorlage kann Abbildung 3.17 dienen. Durch die Modellierung des Automaten in einem eigenen Verhaltensgraphen (VG) werden die im Zustandsautomaten nötigen Überlegungen von den unterschiedlichen Verhaltensweisen in den verschiedenen Zuständen getrennt. Ist z.B. ein Lastwagenfahrer nicht Mitglied in einem Vehikel-Holonen, so wartet er auf einen Auftrag, um Teil eines Vehikel-Holonen zu werden. Wenn er allerdings Mitglied ist, so ist sein Verhalten das Fahren des Vehikels an dessen Bestimmungsort. Diese beiden Verhaltensweisen können in zwei VG modelliert werden.

In SeSAm werden alle VG gleichzeitig abgearbeitet. Auch die beiden Verhalten des Lastwagenfahrers, Warten und Fahren, würden gleichzeitig stattfinden, wenn sie in unterschiedlichen VG modelliert wurden. Um dies zu verhindern ist es notwendig, einen VG aktivieren zu können, der dann ein Verhalten produziert. Der aktivierte VG soll nach seiner Abarbeitung den aktivierenden VG benachrichtigen und auf eine erneute Aktivieren warten. Eine solche Möglichkeit ist im gegenwärtigen SeSAm nicht explizit vorhanden. Es ist jedoch möglich für die Kante, die aus der Startaktivität herausführt, eine Bedingung zu definieren, die die Abarbeitung des VG unterbricht und erst nach der Erfüllung der Bedingung mit der Abarbeitung fortfährt. Um das Aktivieren eines VG durch einen anderen VG, also beispielsweise des Fahren-Verhaltens durch den Zustandsau-

tomaten, zu unterstützen, werden die Primitive `ActivateSemaphore`, `IsSemaphoreActivated` und `DeactivateSemaphore` bereitgestellt. Abbildung 4.12 zeigt einen minimalen VG für das Fahren, in dem eine Semaphore eingesetzt wird. Um auf die Aktivierung der Semaphore zu warten und zum aufrufenden VG zurückzukehren, wird eine Aktivität *`Warte`* eingefügt. In Abschnitt A.3.3 werden die Primitive anhand eines UML Diagramms erläutert.

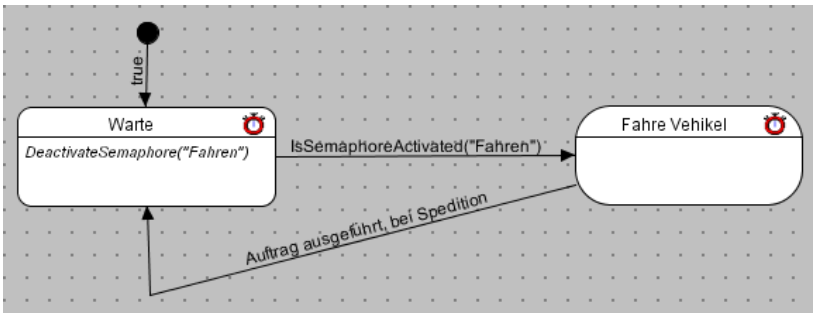



**Abbildung 4.12. Beispiel für eine Semaphore**

Durch die Verwendung mehrerer VG sind Mitgliedschaften in mehreren Superholonen (verschiedener Holonentypen) realisierbar. Für jede Rolle, die ein Agent in einem Superholonen einnehmen kann, kann so ein Verhalten modelliert werden. Je nach der eingenommenen Rolle kann der entsprechende VG aktiviert werden. Mit `IsMemberOfHolon` kann zusätzlich geprüft werden, ob überhaupt eine Mitgliedschaft besteht.

Die Realisierung von mehreren Mitgliedschaften in Superholonen desselben Holonentyps scheitert an dem Problem, dass zur Modellierzeit nicht feststeht, wieviele solche Mitgliedschaften zur Laufzeit entstehen werden. Um für jede Mitgliedschaft in einem Superholonen desselben Typs jedoch individuelles Verhalten darstellen zu können, muss auch hier für jede potentielle Mitgliedschaft ein eigener VG modelliert werden. Dies trifft auch dann zu, wenn die VG für die Mitgliedschaft im selben Holonentypen identisch sind. Dadurch, dass die Anzahl der Mitgliedschaften zur Modellierzeit nicht bekannt ist, kann auch nicht sichergestellt werden, dass genügend VG zur Laufzeit vorhanden sind. Daher wurde in der gegenwärtigen Version von HAP festgelegt, dass **jeder Holon nur Mitglied in einem Superholonen eins Holonentyps** sein kann. In Kapitel 6 werden Wege erläutert, diese Restriktion aufzuheben.

Alternativ zur Verwendung verschiedener VG, um unterschiedliches Verhalten zu realisieren, besteht auch die Möglichkeit der Abfrage des Status eines Holonen

innerhalb eines größeren VG. Dies ist dann sinnvoll, wenn sich die verschiedenen Verhalten nur an wenigen Stellen unterscheiden. So kann mit `IsMemberOfHolon` und `IsRoleInHolon` ein Holon selektiv seinen Zustand in Erfahrung bringen (siehe Abbildung 4.13). Mit `GetRoleInHolon` kann er die Rolle, die er in einem gegebenen Holonen einnimmt, ermitteln und entsprechend sein Verhalten verändern.

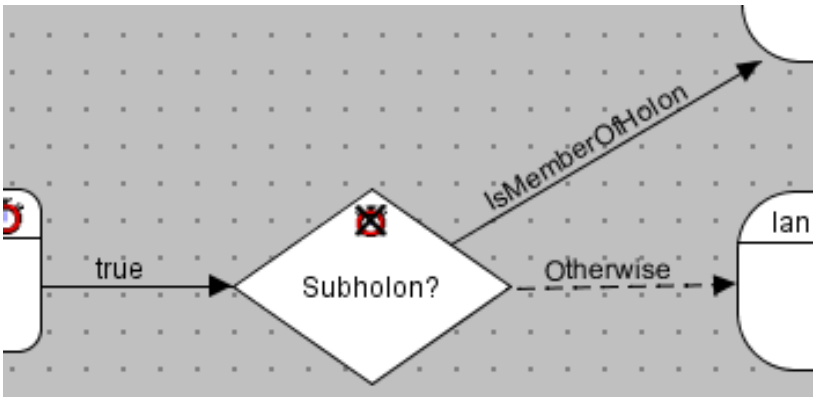



**Abbildung 4.13. Beispiel für die Verwendung von IsMemberOfHolon**

Nachdem eine generelle Vorgehensweise beim Modellieren von holonischen Systemen aufgezeigt wurde, sollen nun konkretere Modellierungsaufgaben umgesetzt werden.

Die **Gründung eines Superholonen** entspricht der Instanziierung eines Holonentypen. Wie in Abschnitt 4.3.1 beschrieben, erfolgt die Definition eines Holonentypen mit Hilfe des HolonTypeEditors. Das Primitiv `CreateHolonObjectForHolonType` erzeugt eine Instanz vom angegebenen Typen und liefert dessen `HolonId` (siehe Abschnitt A.1) zurück. So kann ein PnEU-Agent einen Superholonen vom Typ Vehikel erzeugen, dem er als Kopf vorsteht.

Um seine Aufgabe erfüllen zu können, muss ein Superholon **Subholonen finden, die die notwendigen Rollen ausfüllen**. Mit `GetHolonsWithSkillNeededForRole` kann ein beliebiger Agent eine Liste von Agenten danach filtern, ob die für eine Rolle benötigten Fähigkeiten im jeweiligen Agenten vorhanden sind. Mit `GetFreeAgents` kann eine Liste erstellt werden, die nur Agenten enthält, die nicht Mitglied in einem Superholonen sind. Erhält der PnEU-Agent eines ansonsten noch unbesetzten Vehikel-Holonen durch den Speditionsagenten ein Angebot für einen Auftrag, so kann er durch die Verwendung dieser Primitive eine Liste von Agenten erhalten, die eine Ladefläche zur Verfügung stellen und

die fähig sind, Güter zu transportieren. Im System stellen *Lastwägen* diese Fähigkeiten zur Verfügung. Durch die Verwendung von Fähigkeiten ist diese Tatsache jedoch unwichtig. Es ist nur wichtig, dass die für eine Rolle benötigten Fähigkeiten im Agenten vorhanden sind. Hier wird deutlich, dass durch die Verwendung der Fähigkeiten zur Abbildung von Rollen auf Agenten (bzw. Holone) eine Abstrahierung erreicht wird. Das Besitzen von Fähigkeiten entspricht somit der Implementierung eines *Interfaces* in einer objektorientierten Programmiersprache wie Java.

Soll eine Suche nach einem **Agenten mit einer bestimmten Fähigkeit unabhängig von einer Rolle** durchgeführt werden, so kann das Primitiv `GetHolonsWithSkill` eingesetzt werden. Um die Systemleistung zu optimieren ist es wichtig zu wissen, wie gut ein bestimmter Agent eine Aufgabe bearbeiten kann. Durch die Verwendung von `GetQualityOfSkill` kann die Qualität einer Fähigkeit bei einem Agenten abgefragt werden. Solche Primitive sind Abkürzungen für Kommunikationsvorgänge, denn in Wirklichkeit müsste ein Agent nach dessen Fähigkeiten und deren Qualität gefragt werden.

Wurde ein passender Agent gefunden, so kann er mit `CastAgentForRoleInHolon` **zum Subholonen in einer Rolle gemacht** werden. Zusätzlich liefert `CanAgentBeCastToRoleInHolon` auf bequeme Weise Rückschluss darüber, ob ein beliebiger Agent (oder ein Kopfholon, stellvertretend für einen Superholonen) die Fähigkeiten besitzt, eine Rolle einzunehmen. Der PnEU-Agent kann auf diese Weise einen Lastwagen und den benötigten Fahrer in den Vehikel-Holonen eingliedern.

Nachdem sich Subholonen in einem Superholonen zusammengefunden haben, soll die **Kommunikation** zwischen ihnen erleichtert werden. Wie in Abschnitt 3.2 erläutert, soll sie auch eingeschränkt werden, d.h. Subholonen sollen nur aus bestimmten Gründen miteinander kommunizieren. Diese Interaktionsregeln wurden im HolonTypeEditor spezifiziert (siehe Abschnitt 4.3.1). Mit `GetAgentsConcerningCommunication` kann ein Subholon die ihm für eine intraholonische Kommunikation zur Verfügung stehenden Partner ermitteln. In einem realen System entspricht dies einer Anfrage beim Kopfholonen. Im Beispiel kann der Lastwagenfahrer mit dem Primitiv `GetAgentsConcerningCommunication` die Empfänger seiner Lenkbefehle ermitteln und anschließend übermitteln.

Die Kommunikation an sich muss der Modellierer selbst umsetzen. Mit Hilfe eines UserFeatures oder des Communication-Plugins [CP05] kann eine Entkopplung der konkreten Agentenklasse und der Kommunikationsfähigkeit erreicht werden. Es ist somit nicht wichtig zu wissen, mit welchem Agenten kommuniziert wird, solange die Kommunikationsfähigkeit gegeben ist.

Die **Kommunikationsvorgänge und -protokolle** in TeleTruck (RAP, ECNP) sind komplex und spezialisiert. Für viele Szenarien, wie z.B. dem in Abschnitt 5.1.1 beschriebenen *verteilte Überwachungs*-Szenario, sind einfache Protokolle ausreichend. HAP unterstützt zwei einfache Protokolle durch Primitive. Zum einen ist dies die Anfrage eines Holonen bei einem Kopfholonen wegen der Mitgliedschaft in einem vom Kopfholonen verwalteten Superholonen. Zum anderen handelt es sich um das Angebot eines Kopfholonen an einen Holonen, einem Superholonen beizutreten. Im ersten Protokoll initiiert ein potentieller Subholone den Vorgang, im zweiten Fall tut dies der Kopfholon. Beide Protokolle und die verwendeten Primitive werden in Abschnitt 4.4.1 anhand von UML Diagrammen eingeführt und erläutert.

Will ein Agent oder Holon in einem Superholonen eines bestimmten Typs Mitglied werden, so kann er mit `GetHeadsOfHolonType` vom System eine Liste beziehen, die alle Kopfholonen des angegebenen Typs enthält.

Ein Subholon kann zu dem Schluss kommen, dass er einen **Superholonen verlassen** möchte. Mit `LeaveHolon` benachrichtigt ein Subholon den entsprechenden Kopfholonen von seinem Austritt. Der Kopfholon bekommt dies nicht direkt mitgeteilt, kann jedoch durch den Vergleich des Wertes von `GetNumberOfAgentsInRole` mit einem vorher gespeicherten Wert oder mit dem Wert `GetRequiredNumberOfAgentsInRole` (entspricht der *Kardinalität*, siehe Abschnitt 4.3.1) das Verlassen eines Subholonen feststellen. In TeleTruck kann ein Subholone, z.B. ein Anhänger, den Vehikel-Holonen bei einem Schadensfall verlassen. Der Kopfholone, das PnEU, muss dann entscheiden, wie dieser Ausfall zu kompensieren ist (siehe Abschnitt 3.5).

Die **Kardinalität** kann zur Laufzeit mit `SetRequiredNumberOfAgentsInRole` geändert werden. Dadurch kann die Anzahl der Subholonen, die in einer Rolle benötigt werden, an konkrete Umstände angepasst werden.

Wie in Abschnitt 3.5 beschrieben kann es manchmal notwendig werden, dass ein **Superholon beendet** wird. Dabei handelt es sich um eine Entscheidung des

Kopfholonen, der mit `TerminateHolon` die Beendigung durchführen kann. Die Subholonen bekommen diesen Umstand nicht direkt mitgeteilt, die Abfrage von `IsMemberOfHolon` fällt nach der Auflösung des Superholonen jedoch negativ aus. Wurde im TeleTruck-Beispiel ein Auftrag erfolgreich ausgeführt und ist der Vehikel-Holon zum Ausgangspunkt, der Spedition, zurückgekehrt, so löst der PnEU den Vehikel-Holonen mit `TerminateHolon` auf.

## 4.4.1 Unterstützte Kommunikationsprotokolle

Generelle Kommunikation wird durch HAP nicht unterstützt, sondern muss vom Modellierer selbst umgesetzt werden. Wie oben erwähnt können hier UserFeatures oder das Communication-Plugin hilfreich sein. Zwei spezielle Kommunikationsvorgänge werden jedoch durch Primitive direkt unterstützt. Dies ist zum einen, wenn ein Subholon sich bei einem Kopfholonen für eine Rolle bewirbt und zum anderen, wenn ein Kopfholone einem Agenten bzw. Holonen eine Rolle anbietet.

### Ein Subholon bewirbt sich für eine Rolle

In Abbildung 4.14 initiiert ein Holon, der Mitglied in einem Superholon werden will, den Kommunikationsvorgang. Um sich für eine Rolle zu bewerben sendet ein Holon mit `RequestRole` eine Anfrage an einen Kopfholonen. Ein Kopfholon kann mit `IsRequestAvailable` seinen 'Posteingang' kontrollieren. Befindet sich dort eine oder mehrere Anfragen, so kann er sie mit `PopRequestersForRole` abholen. Die Ergebnisliste enthält alle anfragenden Holone.

Für jede Anfrage entscheidet der Kopfholon, ob der anfragende Holon Mitglied werden kann oder nicht. Falls die Mitgliedschaft erlaubt wird, so wird der Holon mit `CastAgentForRole` in den Superholonen eingebunden und mit `SendAnswerToRequest` die `HolonId` des Superholonen übermittelt. Bei einer Ablehnung enthält `SendAnswerToRequest` eine `null`. Der anfragende Holon hat durch regelmäßigen Aufruf von `WasAnswerToRequestReceived` auf eine Antwort auf seine Anfrage gewartet. Wenn schließlich eine Antwort eintrifft, kann er sie mit `PopAnswerToRequest` abrufen und so feststellen, ob er als Mitglied akzeptiert wurde oder nicht.

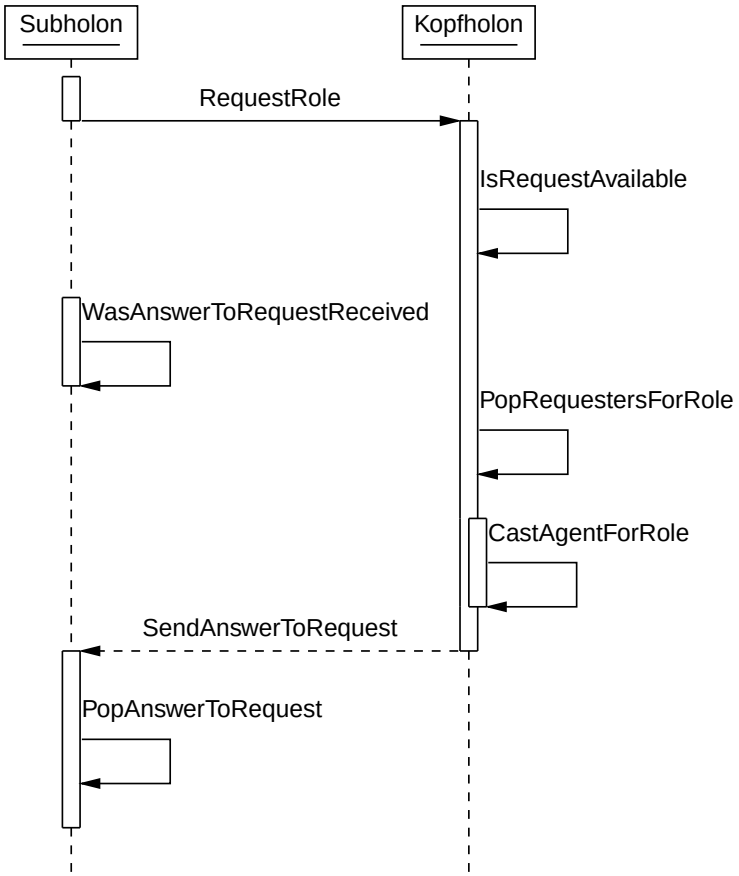



**Abbildung 4.14. Kommunikationsvorgang: Subholon bewirbt sich für eine Rolle.**

## Ein Kopfholon bietet eine Rolle an

In Abbildung 4.15 wird der Kommunikationsvorgang durch einen Kopfholonen angeregt. Dazu sendet er ein Rollenangebot mit `OfferRole` an einen potentiellen Subholonen. Diese können z.B. mit `GetHolonsWithSkillNeededForRole` bestimmt werden. Potentielle Subholonen prüfen regelmäßig den Eingang von Angeboten mit `IsOfferAvailable`. Wenn ein oder mehrere Angebote eingegangen sind, können diese mit `PopOffersForRole` abgerufen werden. Ein solches Angebot besteht im Regelfall aus dem Anbieter und einem optionalen Wert, dessen Bedeutung festgelegt wurde. Mit `GetOffererOfOffer` kann der Anbieter ermittelt werden.

`SendAnswerToOffer` benachrichtigt den Anbieter von der Annahme oder der Ablehnung des Angebots. Nach dem Verschicken eines oder mehrerer Angebote hat der Anbieter durch regelmäßiges Aufrufen mit `WasAnswerToOfferReceived` den Eingang einer Antwort auf sein Angebot kontrolliert. Trifft

eine Antwort ein, so kann er sie mit `PopAnswerToOffer` abfragen. Ist die Antwort zustimmend, so wird der Holon mit `CastAgentForRole` als Mitglied eingebunden. Andernfalls Antwortet der Kopfholon mit einer `null` in `ConfirmOffer`. Eine `null` wird auch dann zurückgeschickt, wenn das Angebot für eine Rolle vom Kopfholonen zurückgezogen wurde. Ein gecasteter Agent bekommt mit `ConfirmOffer` die `HolonId` des Superholonen, in dem er jetzt Mitglied ist, mitgeteilt. Mit `WasConfirmOfferReceived` überprüft der Subholon den Eingang einer Bestätigung und kann diese dann mit `PopOfferConfirmation` abfragen.

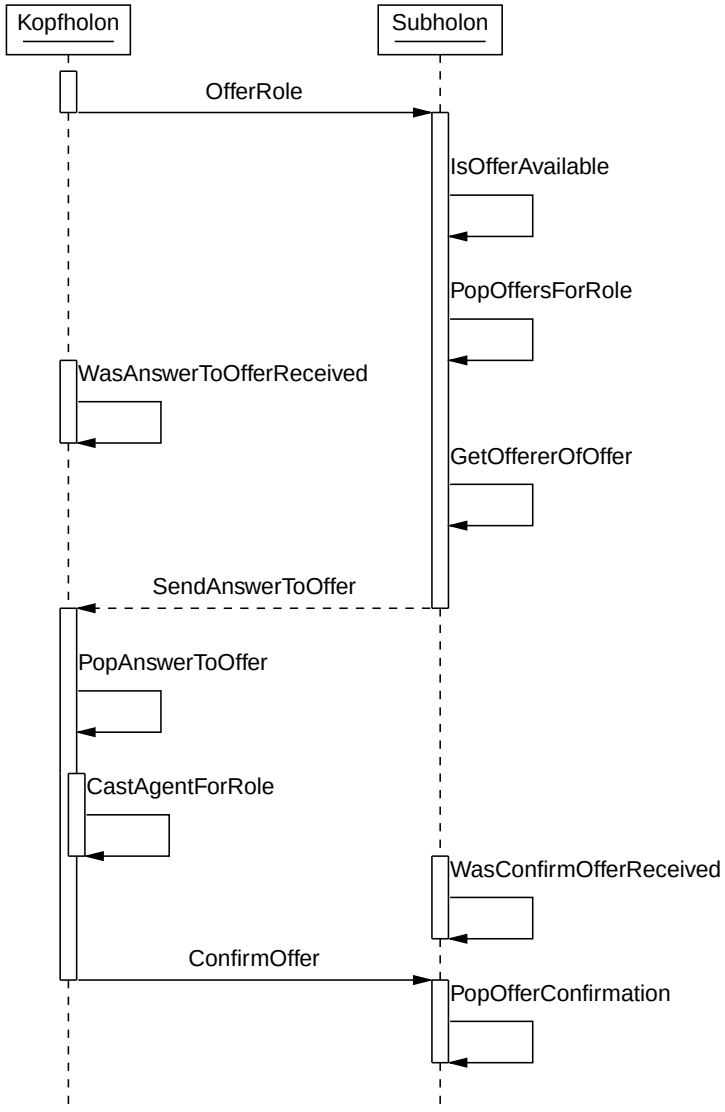



**Abbildung 4.15. Kommunikationsvorgang: Kopfholon bietet eine Rolle an.**

# 5 Beispielanalyse und Diskussion

Die Evaluation der holonischen Erweiterung von SeSAm (*Holonic Augmentation Plugin*) (HAP) wird in zwei Bereiche aufgeteilt. Zum einen ist das der Bereich der Effizienz von holonischen Systemen. Es wurde untersucht, unter welchen Umständen ein holonisches System besser ist als alternative Lösungen und wie stark sich eine Verbesserung bemerkbar macht. Zum anderen wurde der Bereich der Umsetzung von holonischen Systemen betrachtet. Fragen, die in diesem Teil der Evaluation geklärt wurden, waren, inwieweit sich holonische Systeme mit HAP einfacher modellieren lassen als mit den regulären Mitteln von SeSAm und wie HAP das Beobachten des Systems zur Laufzeit unterstützt.

## 5.1 Effizienz von holonischen Systemen

Nach der Implementierung der holonischen Erweiterung von SeSAm war es interessant herauszufinden, wie sich die mit der Erweiterung modellierten Holone zu normalen Agenten im Bezug auf die Effizienz des Problemlösens verhalten. Zu diesem Zweck wurde ein Szenario ausgewählt und sowohl holonisch, als auch mit normalen Agenten modelliert. In den nachfolgenden Abschnitten wird das Modell vorgestellt, die gewonnenen Ergebnisse präsentiert und schließlich diskutiert.

### 5.1.1 Beschreibung des Evaluationsmodells

Das zur Evaluierung der Effizienz holonischer Systeme ausgewählte Szenario stammt aus dem Bereich der verteilten Überwachung. Es wurde gewählt, weil am Lehrstuhl 6 für Künstliche Intelligenz und Angewandte Informatik der Universität Würzburg bereits ein SeSAm-Modell für dieses Szenario existiert. Dieses Modell würde von Credner im Rahmen einer Studienarbeit erstellt [Cr04]. Ferner ist das Szenario wegen seiner Verteiltheit und einer hierarchischen Kommandostruktur, deren Teile sich jedoch selbst organisieren können, für den Einsatz von Holonen prädestiniert.

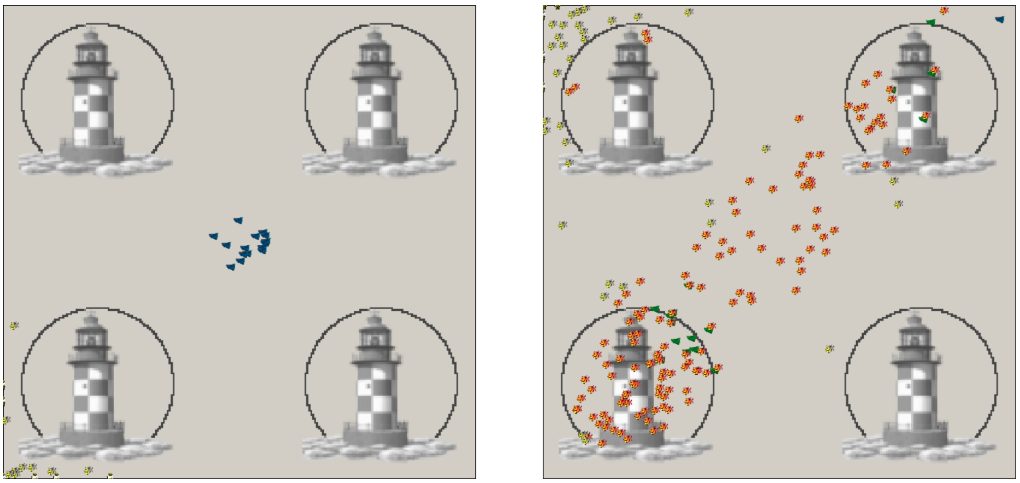
In der verteilten Überwachung (VÜ) geht es darum, ein Gebiet flächendeckend zu überwachen. In diesem Gebiet bewegen sich die zu überwachenden Einheiten, die im folgenden *Ziele* genannt werden. Im Modell betreten die Ziele den zu

überwachenden Bereich mit Kantenlänge `edge` auf einer Seite an einem zufällig gewählten Punkt und bewegen sich dann geradlinig auf einen zufällig gewählten Punkt auf der anderen Seite zu. Für die Überwachung der Ziele werden räumlich verteilte Überwachungseinheiten mit beschränktem Sichtradius `sightS` eingesetzt, die untereinander kommunizieren, um so die Effizienz der Überwachung zu erhöhen. Es soll beispielsweise kein Ziel von mehreren Überwachungseinheiten, im nachfolgenden *Stationen* genannt, gleichzeitig überwacht werden. Die vier Stationen im System sind dabei wie in Abbildung 5.1 zu sehen positioniert.

Dieses Szenario wurde von Credner um aktive Einheiten erweitert, welche die von den Stationen beobachteten Ziele verfolgen und einfangen. Diese Einheiten werden im folgenden *Mobile* genannt. Im ursprünglichen Modell wurden, analog zu biologischen Systemen mit ähnlichen Aufgaben, **Pheromone** für die indirekte Kommunikation verwendet. Die Stationen markieren die Ziele mit Pheromonen, die von den Mobilen von einer größeren Entfernung wahrgenommen werden können, als die Ziele selbst. Die Sichtweite der Mobilen beträgt `sightM`.

Ziel der verteilten Überwachung, wie sie hier beschrieben wurde, ist, alle zu entdeckenden Einheiten möglichst früh aufzuspüren. Daher wurden als **Effizienzmaße** für diese Erweiterung zwei Werte ermittelt: (1) das *durchschnittliche Alter der Ziele*, zum Zeitpunkt des Fangens und (2) die *Fangquote* der Mobilen. Das Alter eines Zieles entspricht der Taktanzahl, die das Ziel in der Monitoring-Welt verbracht hat. Nur wenn ein Ziel gefangen wird, wird sein Alter in den Altersdurchschnitt der gefangenen Ziele eingerechnet. Die Fangquote berechnet sich aus dem Verhältnis der gefangenen zu den insgesamt jemals vorhandenen Zielen. Eine Umsetzung des *verteilte Überwachung*-Szenarios ist umso besser, je jünger die Ziele sind, wenn sie gefangen werden und je höher die Fangquote der Mobilen ist.

Eine alternative Implementierung erlaubt die **direkte Kommunikation** zwischen Stationen und Mobilen. Eine Station weist jedes Ziel dem ihm nächsten Mobilen zu. Dieser verfolgt dann das ihm zugewiesene Ziel. Jede Station kann dabei jedem Mobilen ein Ziel zuweisen. Da die alternative Implementierung sich als leistungsfähiger (im Sinne der Effizienzmaße) als die Implementierung unter Verwendung von Pheromonen erwies, wurde die alternative Implementierung für den Vergleich mit der holonischen Realisierung herangezogen. Um die Bezugnahmen zu erleichtern, wird diese Implementierung im Weiteren mit `VÜ_ohneH` (verteiltes Überwachen ohne Holonen) bezeichnet.



**Linke Abbildung:** Die vier Leuchttürme in den Ecken Stellen die *Stationen* dar. Die Kreise markieren deren Sichtfeld. In der Mitte der Welt befinden sich alle *Mobilen*. In der linken unteren Ecke betreten die ersten *Ziele* die Welt und streben zur rechten oberen Ecke.

**Rechte Abbildung:** Das System hat sich bereits mehrmals nach Richtungswechseln neu organisieren müssen. Bis kurz vor der Aufnahme dieses Bildes bewegten sich die Ziele von unten links nach oben rechts. Die Mobilen haben sich bei den entsprechenden Stationen versammelt und die Systemleistung verbessert. Für die nächsten 500 Takte werden sich die Ziele von oben links nach unten rechts bewegen.

**Abbildung 5.1. Die Monitoring-Welt**

In der **holonischen Realisierung** des Szenarios fungieren die Stationen jeweils als Köpfe eines Monitoring-Holonen. Ein solcher Holon besteht aus einer einzelnen Rolle, der Wächter-Rolle. Diese Rolle kann von beliebig vielen Mobilen eingenommen werden. Eine Station teilt nur ihren Subholonen die Position von Zielen mit, d.h. nur Subholone können den Vorteil der erhöhten Sichtweite einer Station nutzen. Zu Beginn der Simulation sind alle Mobilen frei, d.h. nicht Mitglied in einem Monitoring-Holonen. Sie bewegen sich dann zufällig in dem zu überwachenden Gebiet (der *Monitoring World*). Wenn sie dabei ein Ziel entdecken, so verfolgen sie es und fangen es schließlich, da sie sich schneller bewegen können als die Ziele. Durch das Fangen eines Ziels erhöht sich die Zufriedenheit des Mobilen `sat` um den Wert `add`. Die Zufriedenheit nimmt ansonsten kontinuierlich in jedem Takt der Simulation um den Wert `dec` ab. Unterschreitet sie einen vorgegebenen Wert `threshold`, so ist der Agent unzufrieden mit seinem Gegenwärtigen Zustand im Zustandsautomaten für das Rollenspiel (siehe Abbildung 3.17). Er will dann Subholon werden, wechselt in den Fusion-Zustand und fragt bei einem zufällig ausgewählten Kopf einer der Monitoring-Holone nach, ob er die Wächter-Rolle einehmen kann (das hierfür eingesetzte Kommu-

nikationsprotokoll wird in Abschnitt 4.4.1 erklärt). Stimmt dieser zu, wird der anfragende Agent Subholon, ansonsten bleibt er alleinstehend. Ein Kopfholon lehnt eine Anfrage allerdings nur dann ab, wenn ihm mehr Subholone unterstehen als er Ziele ausmachen kann.

Wie ein freier Agent kann auch ein Subholon unzufrieden werden. Dann beendet der Subholon seine Mitgliedschaft im Monitoring-Holon, indem er seinen Entschluss dem Kopfholonen mitteilt. Diese Implementierung wird im Folgenden mit `VÜ_mitH_1` bezeichnet.

Die **zweite Ausbauphase** dieses Ansatzes besteht darin, Stationen zu erlauben, Subholonen anzuwerben. Stellt eine Station fest, dass sie weit mehr Ziele beobachtet, als Subholone bei ihr Mitglied sind, so bietet sie freien Mobilen die Mitgliedschaft an. Das Angebot enthält das Verhältnis der beobachteten Ziele zu der Anzahl der gegenwärtigen Subholone (der genaue Kommunikationsablauf wird in Abschnitt 4.4.1 dargestellt). Dies ermöglicht es einem Mobilen, der mehrere Angebote von verschiedenen Stationen empfangen hat, das dringendste Angebot auszuwählen. Auf diese Implementierung wird mit `VÜ_mitH_2` Bezug genommen.

In beiden holonischen Modellen erhöht der Agent beim Wechsel zwischen den beiden Zuständen 'Alleinstehend' und 'Subholon' seinen `sat`-Wert auf `sat-Start`. Dies geschieht, damit der Agent nicht sofort wieder unzufrieden ist.

Um dem System eine bessere Struktur zu geben, wurde die **Bewegung der Ziele** weniger zufällig modelliert als dies oben beschrieben wurde. Anstatt sich von einer beliebigen Stelle einer Seite an eine beliebige Stelle an der gegenüberliegenden Seite des Überwachungsbereichs zu bewegen, bewegen sich die Ziele 500 Takte lang von einer Ecke zur diagonal gegenüberliegenden Ecke. Dabei sind die Start- und Zielpunkte um die jeweilige Ecke mit einer Varianz von 5000 (`length_edge * 10`) *normalverteilt* und werden unter Verwendung des SeSAm-Primitivs `RandomNormal` erzeugt. Es wird nur zwischen den beiden linken Ecken abgewechselt, d.h. 500 Takte bewegen sich die Ziele von der unteren linken Ecke zur oberen rechten und die anschließenden 500 Takte von der oberen linken zur unteren rechten Ecke. Die Ziele selbst ändern ihre Richtung nicht. Es ändert sich nur die Richtung, in der neue Ziele den Bereich durchqueren. In Abschnitt 5.1.4 wird der Grund für diese Veränderungen erläutert.

## 5.1.2 Modellparameter

Im Folgenden eine Aufstellung aller wichtigen Modellparameter.

**Tabelle 5.1. Parameter aller Modelle**

| | |
|---|---|
| **length_edge** | Die Kantenlänge des überwachten Bereichs (der Monitoring-Welt) beträgt 500 Einheiten. |
| **Anzahl der Stationen** | Im System existieren vier Stationen. |
| **Anzahl der Agenten** | Im System gibt es 20 Mobile. |
| **sightM** | Das Sichtfeld eines Mobilen beträgt 10 Einheiten. |
| **sightS** | Das Sichtfeld einer Station beträgt length_edge/6, also ca. 80 Einheiten, im Umkreis. |
| **Geschwindigkeit Ziele** | Die Ziele bewegen sich um 5 Einheiten pro Takt. |
| **Geschwindigkeit Mobile** | Die Mobilen bewegen sich um 7 Einheiten pro Takt. |
| **Ziele pro Takt** | Pro Takt werden drei neue Ziele erzeugt. |
| **Richtungswechsel** | Alle 500 Takte ändert sich die Richtung, in der sich die Ziele durch den Überwachungsbereich bewegen. |

**Tabelle 5.2. Parameter der holonischen Modelle**

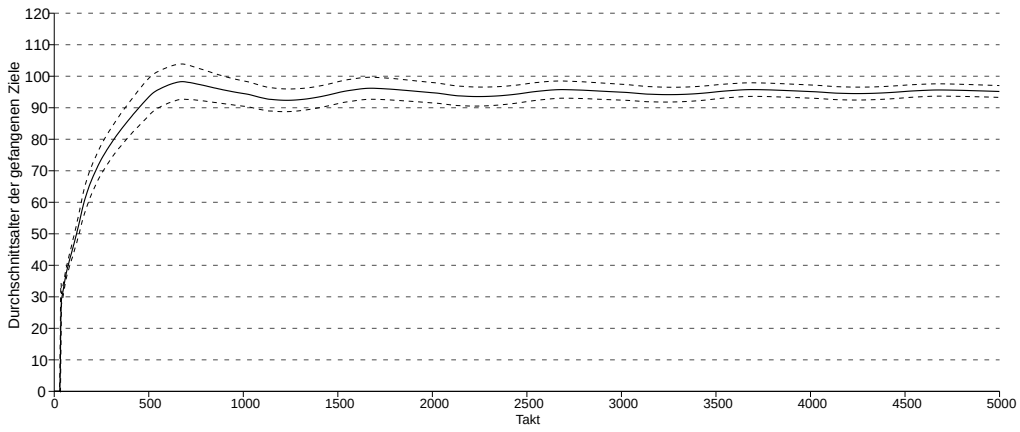
| | |
|---|---|
| **sat** | Die Zufriedenheit eines Agenten. Sie kann maximal 25 betragen. |
| **satStart** | Beim Wechsel zwischen den Zuständen 'Alleinstehend' und 'Subholon' erhält der Agent einen Zufriedenheitswert von 15. |
| **add** | Für jedes gefangene Ziel wird ein Wert von 6 zu `sat` hinzuaddiert. |
| **dec** | Pro Takt wird von `sat` eine Einheit abgezogen. |
| **threshold** | Sinkt der Wert von `sat` unter 0, so wird ein Agent unzufrieden. |

### 5.1.3 Ergebnisse

Um die Werte für die Effizienzmaße zu ermitteln, wurden die erstellten SeSAm-Modelle mit den entsprechenden Analysen ausgestattet. Um statistisch aussagekräftige Werte zu erhalten, wurde jedes der drei Modelle *VÜ_ohneH*, *VÜ_mitH_1* und *VÜ_mitH_2* mit je 100 Wiederholungen simuliert. Jede Simulation bestand dabei aus 30.000 Takten.

Nachfolgend werden die ermittelten Werte nach einer graphischen Aufbereitung dargestellt. Um die Graphen übersichtlich zu halten, wurden nicht die Daten der 30.000 Takte eingearbeitet, sondern nur die der ersten 5000 Takte. Die y-Achse der Graphen gibt das durchschnittliche Alter (DA) der Ziele bei ihrem Fang an. Zu den Zeitpunkten der Richtungswechsel befinden sich Markierungen mit der Angabe des Takts an der x-Achse. Jeder Graph gibt den Mittelwert und die Standardabweichung der 100 Simulationsläufe wieder.

Im Modell **VÜ_ohneH**, in dem klassische Agenten eingesetzt wurden, ergab sich eine Fangquote von 79,5% am Ende der Messung. Das DA steigt schnell an und erreicht bei 600 Takten (direkt nach dem ersten Richtungswechsel) das Maximum. Danach singt das DA wieder und steigt nach jedem Richtungswechsel erneut an. Diese Schwankungen konzentrieren sich um einen Wert von 95. Auch die Standardabweichung erreicht bei Takt 600 ihr Maximum von sechs Takten und nimmt dann kontinuierlich ab. Am Ende der Messung beträgt sie zwei Takte.

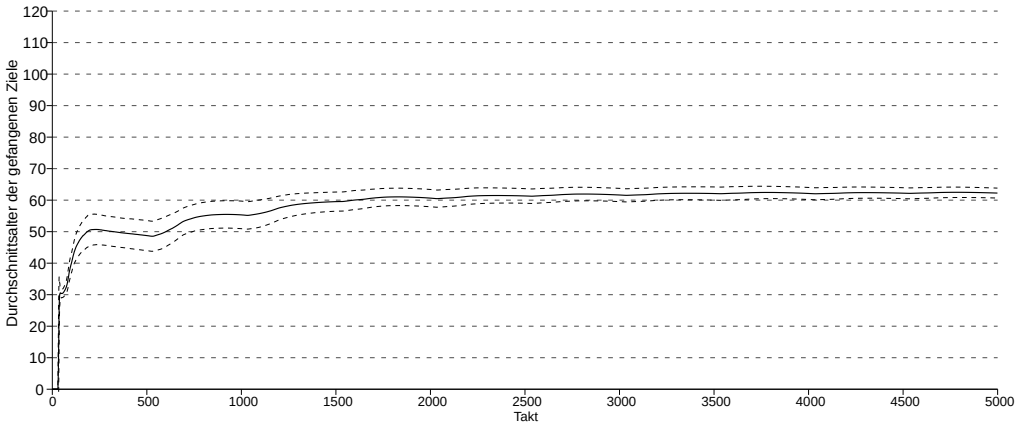



Die Graphik zeigt das Durchschnittsalter der gefangenen Ziele und dessen Standardabweichung (gestrichelte Linie). Die Fangquote in diesem Modell mit klassischen Agenten beträgt 79,5%.

**Abbildung 5.2. Ergebnisgraphik für VÜ_ohneH**

In den beiden Modellen, in denen Holonen verwendet werden, ist die Fangquote jeweils ca. 11 Prozentpunkte höher als im Modell mit klassischen Agenten. Das Durchschnittsalter der Ziele beim Fang liegt in beiden Modellen bei ungefähr 60, d.h. die holonischen Modelle unterbieten das Modell mit klassischen Agenten um ca. 30%.

Im ersten holonischen Modell, **VÜ_mitH_1**, in dem die Mobilen nach einer Station suchen, beträgt die Fangquote am Ende der Messung 91,9%. Auffällig ist das stufenweise Ansteigen des DA. Ein erstes Plateau wird nach 200 Takten bei einem Wert von 49 Takten erreicht. Nach dem ersten Richtungswechsel hebt sich das Niveau auf 55 Takte. Nach dem dritten Wechsel bei 1500 Takten, beträgt das DA der Ziele ca. 63 Takte. Auf diesem Wert pendelt sich das System ein und bleibt nach 2500 Takten weitgehend stabil. Die Standardabweichung des DA erreicht gleich zu Beginn der Simulation, bei Takt 700 ein Maximum von fünf Takten. Danach nimmt sie kontinuierlich ab. Das Einpendeln des Systems beschleunigt den Vorgang. Am Ende der Messung beträgt die Standardabweichung weniger als zwei Takte.

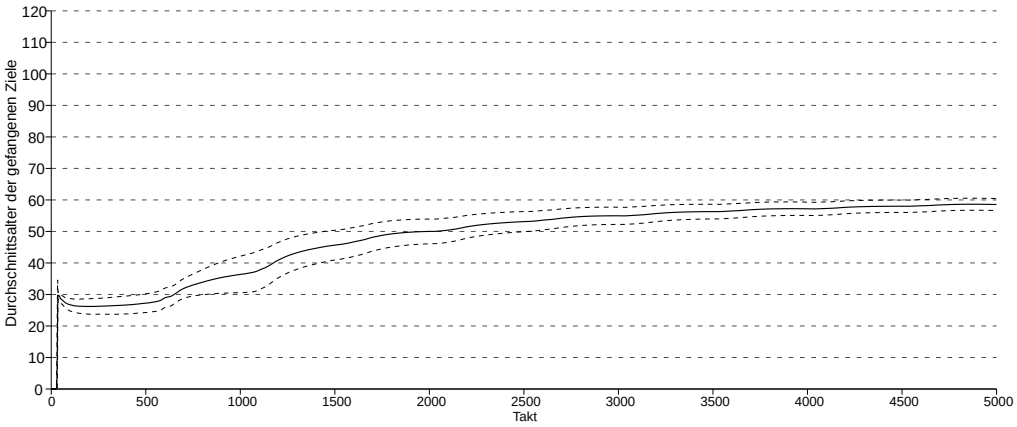



Die Graphik zeigt das Durchschnittsalter der gefangenen Ziele und dessen Standardabweichung (gestrichelte Linie). Die Fangquote in diesem holonischen Modell beträgt 91,9%.

**Abbildung 5.3. Ergebnisgraphik für VÜ_mitH_1**

Im zweiten holonischen Modell, **VÜ_mitH_2**, suchen nicht nur die Mobilen eine Station, sondern die Stationen suchen auch nach Mobilen. Die Fangquote beträgt hier 90,9%. Wie im ersten holonischen Modell ist eine asymptotische Annäherung des DA der Ziele von unten an den Wert zu beobachten, bei dem sich das System schließlich einpendelt. Auffällig an den Ergebnissen dieses Modells ist, dass gleich nach Simulationsstart das DA auf einem niedrigen Niveau

von unter 30 Takten stagniert. Erst nach dem ersten Richtungswechsel der Ziele beginnt das DA weiter anzusteigen und erreicht mit jedem Wechsel ein weiteres Plateau. Schließlich, ab ca. 4000 Takten, hat sich das System eingependelt und weitere Richtungswechsel ändern das DA von 58 Takten nicht mehr wesentlich. Die Standardabweichung erreicht nach langsamem Anstieg bei 1100 Takten ihr Maximum von 6 und fällt bis zum Ende der Messung auf unter zwei Takten ab.




Die Graphik zeigt das Durchschnittsalter der gefangenen Ziele und dessen Standardabweichung (gestrichelte Linie). Die Fangquote in diesem holonischen Modell beträgt 90,9%.

**Abbildung 5.4. Ergebnisgraphik für VÜ_mitH_2**

## 5.1.4 Diskussion

Die Ergebnisse aus Abschnitt 5.1.3 sind in der folgenden Tabelle zusammengefasst. Anschließend erfolgt eine Betrachtung verschiedener Aspekte der gewonnenen Analysewerte.

**Tabelle 5.3. Ergebnisse der Evaluation zusammengefaßt**

| **Modell** | **Fangquote** | **DA** | **Standardabweichung (Maximum/Ende)** |
|---|---|---|---|
| VÜ_ohneH | 79,5% | 94 | 6/2 |
| VÜ_mitH_1 | 91,9% | 62 | 5/2 |
| VÜ_mitH_2 | 90,9% | 58 | 6/2 |

## Unterschiede im Durchschnittsalter

Die maximale Dauer einer vollständigen Durchquerung des überwachten Bereichs durch ein Ziel beträgt $\frac{\sqrt{length\_edge^2 * 2}}{Geschwindigkeit\ Ziele} \approx 140$ Takte. Ein Ziel kann somit maximal 140 Takte alt werden. Je näher das DA der Ziele an diesen Wert heranreicht, desto später werden die Ziele gefangen und desto schlechter ist das jeweilige Modell in Bezug auf dieses Effizienzmaß. Es kann hier festgestellt werden, dass die beiden holonischen Modelle um 30% besser sind als das Modell mit klassischen Agenten.

Die Mobilen in den holonischen Modellen fangen die Ziele früher, weil sie in der Nähe einer Station, ihrem jeweiligen Kopfholonen, bleiben. Die Station versorgt ihre Mobilen mit Informationen darüber, wo sich das ihnen nächste Ziel befindet. Da sich der Mobile an diesen Informationen orientiert, bleibt er im Sichtfeld der jeweiligen Station, da er nur hier diese Informationen erhalten kann. Die Station, die die meisten Ziele beobachtet, kann deshalb viele Mobile als Subholonen mit solchen Informationen versorgen. Die gleichmäßige Verteilung der Stationen im Überwachungsbereich führt dazu, dass die Station, die dem Eintrittspunkt der Ziele in den Überwachungsbereich am nächsten liegt, die meisten Ziele beobachtet.

Das Durchschnittsalter im Modell mit klassischen Agenten ist auch deswegen hoch, weil Mobile durch das verfolgen der Ziele sich in deren Richtung, weg vom Eintrittspunkt, bewegen. Da es auch weiter weg vom Eintrittspunkt Ziele zu fangen gibt, besteht für die Agenten in diesem Modell kein Anlass, sich zurück in Richtung Eintrittspunkt zu bewegen.

## Einpendelverhalten

Die Modelle zeigen unterschiedliches Verhalten beim Einpendeln um einen stabilen Wert für DA. Das Modell mit klassischen Agenten steigt sofort auf den maximalen Wert seines DA und schwingt sich dann bei einem festen Wert ein. Leichte Schwankungen bleiben hier jedoch bestehen. Die beiden holonischen Modelle nähern sich dagegen von unten asymptotisch an den stabilen Wert an. Nach anfänglichen starken Anstiegen im DA in Folge eines Richtungswechsels, sind kaum mehr Änderungen nach dem Erreichen des stabilen Wertes bei holonischen Modellen zu registrieren.

Die holonischen Modelle nähern sich deshalb von unten an ihre stabilen Werte an, weil sie, wie im vorherigen Abschnitt erläutert, Ziele früher fangen als das Modell mit klassischen Agenten. Die stärkeren Reaktionen auf Richtungswechsel in Modell VÜ_ohneH liegen in der Trägheit der klassischen Agenten in diesem Modell begründet. Es dauert länger bis ausreichend viele Agenten feststellen bzw. mitgeteilt bekommen, dass die Ziele aus einer anderen Richtung kommen. Die Holonen registrieren Wechsel schneller. Wenn ein Mobiler in einem der holonischen Modelle feststellt, dass er nur noch wenige Ziele fängt, so sucht er sich eine Station, bei der er mehr Ziele fangen kann.

Im zweiten holonischen Modell, in dem die Stationen nach Mobilen suchen, geschieht der Informationsaustausch über einen Richtungswechsel noch schneller. Daher ist hier das DA geringer.

## Fangquote

Die Fangquote liegt bei den holonischen Modellen 11 Prozentpunkte höher als im Modell mit klassischen Agenten. Die Trägheit der klassischen Agenten beim Feststellen eines Richtungswechsels führt dazu, dass viele Ziele aus der neuen Richtung ungehindert den Überwachungsbereich durchqueren können.

Ein ähnlicher Grund liegt beim leicht schlechteren Fangverhalten des zweiten holonischen Modells, bei dem Stationen nach Mobilen fragen (VÜ_mitH_2), gegenüber dem ersten (VÜ_mitH_1) vor. Im späteren Stadium der Simulation stehen einem Agenten Anfragen von verschiedenen Stationen zur Wahl. Der Agent begibt sich dann zunächst zu der Station, die das größte Verhältnis beobachteter Ziele zu Mobilen (Subholonen) angegeben hat. Es ist dabei durchaus möglich, dass die Station, die das Angebot abgegeben hat, nur noch die letzten Ziele der alten Bewegungsrichtung beobachtet hat. Bis die gerufenen Agenten an der Station angelangt sind, sind kaum mehr Ziele vorhanden und es dauert eine Zeit, bis die Agenten als Subholone dieser Station wieder unzufrieden werden. In der Zwischenzeit durchqueren einige Ziele in der neuen Bewegungsrichtung ungehindert den überwachten Bereich.

### Tendenzen

Die Abbildungen 5.2 bis 5.4 zeigen die Werte für 5000 Takte der Simulation. Für die Datengenerierung wurden jedoch 30.000 Takte pro Modell und Lauf simuliert. Die restlichen Daten dienten zur Auswertung von Tendenzen. Es konnte festgestellt werden, dass sich die Werte aller drei Modelle um einen festen Wert einpendeln und mit zunehmender Taktzahl weniger Schwankungen unterliegen.

### Chaos und Richtungswechsel

Bei der Evaluation der Modelle ist aufgefallen, dass Holone eine gewisse Ordnung in der Umgebung benötigen, um sich gut anpassen zu können. Zu Beginn der Evaluierung betraten die Ziele an einer zufälligen Stelle einer Seite der Welt das überwachte Gebiet und strebten einen zufällig ausgewählten Punkt der gegenüberliegenden Seite an, um die Welt wieder zu verlassen. Es stellte sich heraus, dass bei einem so hohen Maß an Zufälligkeit die holonische Lösung keinen Vorteil gegenüber der nicht-holonischen Lösung aufweist. Dies liegt daran, dass in einem chaotischen System sinnvolle Organisation kaum möglich ist, da keine Strukturen vorhanden sind, an die sich angepasst werden könnte.

Um die Umgebung weniger chaotisch zu gestalten, wurde die Bewegung der Ziele so abgeändert, dass sie in einer der linken Ecken die Welt betreten und zur diagonal gegenüberliegenden Ecke streben. Alle 500 Takte wird dabei die Eintrittsecke von unten nach oben oder von oben nach unten gewechselt. Die Wechsel sollen eine Neuorganisierung des Systems anregen.

### Zusammenfassung

Es lässt sich feststellen, dass die holonischen Modelle in Bezug auf die untersuchten Effizienzmaße besser sind als das nicht-holonische Modell. Dies liegt hauptsächlich daran, dass in den holonischen Modellen mehr Informationen und diese gezielter (feste Zuweisung der Kommunikationspartner innerhalb eines Superholonen) übermittelt werden. Die Mobilen in den holonischen Modellen sind proaktiver, weil sie selbständig einen Überwachungsbereich mit vielen Zielen suchen. Aus dem Zusammenspiel von *Autonomie* und *Kooperation*

erwächst ein sich sehr gut selbstorganisierendes System, das sich sehr erfolgreich an seine Umgebung anpasst.

Die bessere Leistung eines holonischen Systems wird durch einen **höheren Modellieraufwand** erkauft. Der Mehraufwand entfällt dabei hauptsächlich auf die Kommunikationsprotokolle und auf die Modellierung verschiedener Verhaltensweisen in den Zuständen 'Alleinstehend' und 'Subholon'.

Die **Entscheidung zwischen einem System mit Holonen und einem System mit normalen Agenten** muss daher auf Basis der Abschätzung erfolgen, inwieweit die Vorteile des Einsatzes von Holonen den Mehraufwand beim Entwickeln eines holonischen Systems kompensieren oder übertreffen. Im VÜ-Szenario muss beispielsweise abgewägt werden, ob einfache und in der Entwicklung günstige Agenten eingesetzt werden sollen, deren Effizienz weit niedriger als die von Holonen ist. Oder, ob eine hohe Systemeffizienz gefordert wird, für die sich der Mehraufwand bei der Entwicklung einer holonischen Lösung lohnt.

## 5.2 Entwicklung von holonischen Systemen

Die Benutzbarkeit von HAP bei der Entwicklung holonischer Modelle in SeSAm war ein weiterer Punkt in der Evaluation der Erweiterung. Eine aussagekräftige Evaluation einer Benutzerschnittstelle, wie sie HAP darstellt, kann nur durch Rückmeldungen einer möglichst großen Anwendergruppe erstellt werden. Da der Entwickler von HAP der bisher einzige Anwender der Erweiterung ist, stützen sich die im folgenden dargestellten Ergebnisse ausschließlich auf seine Erfahrungen. Da HAP weiterentwickelt werden soll, sind auch diese Erfahrungen nur ein Zwischenstand. Es wird erwartet, dass mit der Zunahme der Anwender die Erweiterung an deren Bedürfnisse angepasst und somit verbessert werden wird.

Die in Abschnitt 4.3 vorgestellten grafischen Bedienelemente unterstützen den Modellierer weitestgehend bei der Entwicklung von holonischen Strukturen. Besonders der HolonTypeEditor (siehe Abbildung 4.5) sei hier erwähnt. Ohne die durch ihn realisierbare Kapselung holonischer Strukturen wäre die Verwendung von generellen SeSAm-Datentypen zur Speicherung dieser Informationen notwendig. Die durch HAP bereitgestellten Primitive erlauben desweiteren den unkomplizierten Zugriff auf und die Verwendung der Strukturen, die mit dem HolonTypeEditor definiert wurden.

Ein weiterer Gewinn bei der Umsetzung von holonischen Systemen im Vergleich zur Umsetzung ohne HAP wird bei Kommunikationsvorgängen erreicht. Die in Abschnitt A.2.2 und A.3.2 beschriebenen Primitive kapseln intern ansonsten explizit zu modellierende Variablen und Verwaltungsmethoden für diese. Dadurch, dass der Modellierer nur mit den Primitiven, nicht jedoch mit den Datenstrukturen in Kontakt kommt, vermindert sich die Gefahr von Fehlern in der Modellierung eines Kommunikationsvorgangs. Gerade bei Kommunikationsvorgängen, die sehr fehleranfällig und daher schwierig zu modellieren sind, ist die Reduktion auf das Wesentliche ein Vorteil.

Eine weitere Reduktion wurde bei den Primitiven realisiert. Es wurden nach Möglichkeit einfache Primitive gewählt, die zusammengesetzt komplexere Funktionen anbieten. So werden die Primitive `GetHolonsWithSkillNeededForRole` und `GetFreeAgents` bereitgestellt, die unabhängig voneinander eingesetzt werde können. Zusammengesetzt in der Form `GetHolonsWithSkillNeededForRole(GetFreeAgents)` ersetzen sie ein Primitiv wie `GetFreeAgentsWithSkillNeededForRole`. Die komplexere Funktion kann der Modellierer selbst, in einer UserFunction gekapselt, nachbilden. Auf diese Weise lassen sich auf Basis weniger, mächtiger Primitive Bibliotheken komplexerer Funktionen erstellen.

Für das Wechseln zwischen verschiedenen VG, das bei der Modellierung von Holonen in SeSAm sehr hilfreich ist, mussten vor der Erweiterung von SeSAm mit HAP für jeden Wechsel boolesche Variablen deklariert werden. Dies bläht die Variablenliste auf, was unnötig ist, weil die Variablen nur für diesen einen Zweck benötigt werden und ansonsten uninteressant sind. Der Zustand dieser Variable lässt sich implizit an den aktivierten VG ablesen, deshalb ist die Information in den booleschen Variablen redundant. Mit Hilfe der in Abschnitt 4.4 und A.3.3 beschriebenen Semaphoren-Primitive wird eine Kapselung dieser booleschen Variablen erreicht. Die Verwendung der Primitive macht Aus- und Eintrittspunkt der verschiedenen VG explizit, und somit das Modell lesbarer.

Als letzte Verbesserung der Unterstützung von holonischen Modellen in SeSAm durch HAP soll die Möglichkeit der Laufzeitanalyse angeführt werden. Die in Abschnitt 4.3 erläuterten Anzeigeelemente versetzen den Modellierer in die Lage, Informationen zu einzelnen Holonen und deren Beziehungen untereinander anzuzeigen. Vor allem der Holarchie-Browser (siehe Abschnitt 4.3.4), aber auch das Subholon-Panel und Headholon-Panel zeigen diese Informationen an. Das

Traversieren der Holarchie ist eine weitere Möglichkeit die Systemstruktur zu beobachten.

Durch das nachträgliche Hinzufügen weniger holonischer Attribute zu einem bestehenden nicht-holonischen Modell kann auch dessen hierarchische Struktur mit dem Holarchie-Browser untersucht werden: Die in Abschnitt 4.3 beschriebenen Features müssen den entsprechenden Agentenklassen zugewiesen werden. Ein primitiver Holonentyp mit einer Rolle, für die keine Fähigkeiten benötigt werden, schafft eine minimale Struktur. Agenten aus höheren Hierarchieebenen fungieren dann als Kopfholonen, instanziieren den Holonentyp und binden die ihnen in der Hierarchie untergebenen Agenten als Subholonen ein.

**Zusammenfassend** erlaubt HAP die Modellierung holonischer Systeme in einer Weise, wie sie in SeSAm ohne Erweiterung nicht möglich wäre. Der Entwickler wird in vielfacher Hinsicht bei der Modellierung und Laufzeitanalyse des Systems unterstützt.

Nach Rückmeldungen von Anwendern muss HAP jedoch sicherlich angepasst und erweitert werden.

# 6 Zusammenfassung und Ausblick

## 6.1 Zusammenfassung

In einer holonischen Multiagentensimulation werden Holonen als grundlegende Einheiten des simulierten Systems eingesetzt. Die Verwendung von Holonen in SeSAm wird durch die holonische Erweiterung (HAP) unterstützt. Das Konzept des Kopfholonen erlaubt es, einen Superholonen durch einen Agenten zu repräsentieren. Es müssen daher keine neuen Strukturen in das System eingeführt werden. Der Kopfholon stellt die Schnittstelle des Superholonen gegenüber seiner Umwelt dar.

Durch die Kapselung der Funktionalität mehrerer Agenten in einem Superholonen können Holone bei der komponentenorientierten Entwicklung von Multiagentensystemen und -simulationen eingesetzt werden. Hier dienen sie als Bausteine komplexer Systeme. Ihre Verwendung erlaubt eine abstrahierende Sicht auf das entwickelte System. Eine abstrakte Aufgabe kann an einen Superholonen vergeben werden. Dieser konkretisiert sie durch Zerteilung und verteilt die Teilaufgaben an seine Subholonen zur weiteren Bearbeitung. Das Akkumulieren von Informationen aus den Subholonen in einem Superholonen ist eine weitere Möglichkeit dieser Sichtweise.

Für die Entwicklung holonischer Systeme wurde ein bereits existierendes Vorgehensmodell für die Analyse und den Entwurf agentenbasierter Systeme (SODA) um das Konzept des Holonen erweitert und anhand eines Beispiels vorgestellt. Die Erweiterung besteht im expliziten Einsatz eines Zielgraphen zur Systemanalyse und der Verwendung von Fähigkeiten zur Abstraktion von Agentenklassen und Holonentypen.

Basierend auf den vorher erarbeiteten theoretischen Grundlagen wurde eine Erweiterung von SeSAm implementiert, welche das Modellieren holonischer Modelle unterstützen soll. Die graphische Analyse eines holonischen Systems zur Laufzeit ist ebenfalls möglich. Hier ist besonders der Holarchie-Browser zu erwähnen.

Schließlich wurde die Effizienz und die Unterstützung des Modellierers beim Erstellen von holonischen Systemen in SeSAm untersucht. In beiden Punkten

sind die Ergebnisse vielversprechend: (1) Im Beispielszenario des verteilten Monitorings wurden in den holonischen Modellen die Bereiche umfassender überwacht und Ziele früher entdeckt als im Modell mit klassischen Agenten. (2) Die Entwicklung eines holonischen Systems wird durch HAP ermöglicht und unterstützt. Die Möglichkeit der Laufzeitanalyse hilft bei der genaueren Analyse des Systems und beim Finden von Fehlern in den intraholonischen Strukturen.

## 6.2 Ausblick

*It's better to make something extendable by somebody else than to waste your time making it so complete that simplicity goes away.*
--Scott McKay

Für die zukünftige Entwicklung der holonischen Erweiterung von SeSAm sollen einige Anhaltspunkte gegeben werden. Sie betreffen hauptsächlich die Implementierung der Unterstützung für den Modellierer.

Im HolonTypeEditor können bereits Interaktionsregeln spezifiziert werden. Für die Modellierung wäre es hilfreich, wenn hier auch das verwendete Interaktionsprotokoll angegeben werden könnte. Dies würde die Verwendung von Kommunikationsschnittstellen als Abstraktion von konkreten Agentenklassen erlauben. So könnte ein Subholone durch die Angabe des Kommunikationszwecks (concern) bei seinen Kommunikationspartnern das entsprechende Protokoll aktivieren. Der Subholon braucht keine weiteren Kenntnisse über seine Partner. Es muss nur sichergestellt sein, dass das Kommunikationsprotokoll und der Zweck unterstützt werden. Dies könnte beim Einbinden eines Holonen in einen Superholonen überprüft werden.

Um die Verwendung von Interaktionsprotokollen zu unterstützen, wäre es von Vorteil, wenn dem Modellierer eine Bibliothek von Protokollen zur Verfügung steht. Es könnten dann einzelnen Agentenklassen bestimmte Rollen aus den vorgegebenen Protokollen (z.B. die Rolle 'Participant' im CNP, siehe Abschnitt 3.4.3) zugewiesen und mit dem Zweck der Kommunikation versehen werden, damit bei mehreren gleichen Protokollen die richtige Instanz ausgewählt werden kann. Jede Protokollinstanz könnte als separater Verhaltensgraph modelliert werden. Die Zuweisung eines Protokolls würde dann einen schablonenhaften Verhaltensgraphen in der Agentenklasse erzeugen, den der Modellierer anpassen kann.

Eine weitere Ausbaustufe dieser Idee ist das Hinzufügen (und Entfernen) von Verhaltensgraphen zur Laufzeit. Dies würde es ermöglichen, die Beschränkung auf eine Mitgliedschaft pro Holonentyp aufzuheben. Der Modellierer müsste dann angeben, welche Verhaltensgraphen für die jeweilige Mitgliedschaft notwendig sind. SeSAm könnte dann zur Laufzeit diese Gruppe von Verhaltensgraphen dem Agenten hinzufügen. Die oft engmaschigen Abhängigkeiten zwischen unterschiedlichen Verhaltensgraphen lässt die Implementierung dieser Technik jedoch schwierig erscheinen.

In einer zukünftigen Version von HAP sollte der Kopfholon ebenfalls im HolonTypeEditor deklarierbar sein. Dies ist vor allem für die Angabe der benötigten Fähigkeiten des Kopfholonen notwendig. Bisher ist für die Erzeugung eines Holonen durch einen Agenten nur das Vorhandensein des Headholon-Features notwendig.

Die für die Einnahme einer Rolle in einem Superholonen notwendigen Fähigkeiten sollten eine komplexere Struktur aufweisen können. Zur Zeit genügt es für die Einnahme einer Rolle, wenn *eine* der benötigten Fähigkeiten im Agenten vorhanden ist. Eine Modellierung in Form eines Und-Oder-Baums würde komplexere Bedingungen erlauben.

Interessante Einsatzgebiete von Holonen sind solche, in denen fast zerlegbare Probleme existieren (siehe Abschnitt 3.6). Hierfür wäre es notwendig, dass Agenten in SeSAm mit der Fähigkeit der Planrepräsentation und Planzerlegung ausgestattet werden. Dann wäre die abstrahierte Ausführung von Plänen (siehe Abschnitt 3.6.1) durch Holone ebenfalls möglich.

Erst durch den regelmäßigen Einsatz von HAP lassen sich weitere Defizite und verbesserungsfähige Punkte der gegenwärtigen Implementierung erkennen und in zukünftigen Versionen ausbessern.

# Datenstrukturen und Primitive

Für die Beschreibung der Primitive, die durch die einzelnen Features zur Verfügung gestellt werden, wurde die Syntax `Primitivname( Arg_1, ..., Arg_n) retVal` gewählt. Dabei steht *Primitivname* für den Namen des Primitives, wie er in der Primitivliste von SeSAm angezeigt wird. *Arg_i* steht für das i-te Argument des Primitives und stellt Argumenttyp und Argumentname für die Bezugnahme im beschreibenden Text gleichzeitig dar, wenn dies eindeutig ist. *retVal* steht für den Rückgabewert des Primitives. Er kann void (kein Rückgabewert), bool (ein boolescher Wert) oder ein SeSAm-Typ sein.

## A.1 Datentypen

Das HAP stellt die folgenden SeSAm-Datentypen bereit, die für das Arbeiten mit den durch die Features des HAP zur Verfügung gestellten Primitiven notwendig sind.

**`Competency`**

Eine **Competency** ist ein SeSAm-ComposedType, ein Datentyp, der aus anderen Datentypen zusammengesetzt ist. Competency besteht aus einem *Skill* und einer *Quality*, d.h. einem numerischen Wert für die Güte des Skills.

**`Concern`**

Dient zur Umsetzung der Interaktionsregeln von SODA (siehe Abschnitt 4.1.3). Intern handelt es sich um einen *String*.

**`HolonId`**

Eine HolonId identifiziert einen Holonen eindeutig im System. Sie wird von Kopf- und Subholonen in Primitiven verwendet, damit Daten einer konkreten Holonen-Instanz bearbeitet werden können.

**`HolonType`**

Der Name einer zur Modellierzeit definierten Organisationsstruktur.

**`Offer`**

Das Angebot für eine Rolle von einem Kopfholonen an einen potentiellen Subholonen. Es besteht aus dem Anbieter und einer Zahl, deren Bedeutung durch den Modellierer festgelegt wird.

**`Role`**
Steht für eine Rolle im Holonen (siehe Abschnitt 4.1.3).

**`Skill`**
Der Name einer Fähigkeit. Intern handelt es sich um einen *String*.

# A.2 Primitive des Headholon-Features

## A.2.1 Primitive für die Verwaltung eines Superholonen

Mit den hier beschriebenen Primitiven kann ein Kopfholon neue Holone erstellen und beenden. Er kann Agenten zu Subholonen machen, indem er ihnen Rollen zuweist und sie aus dem Holonen wieder entlassen.

**`CreateHolonObjectForHolonType(HolonType) HolonId`**
Erzeugt einen Holon vom angegebenen Typ und liefert ein HolonId-Objekt zurück. Der aufrufende Agent wird automatisch Kopf des neu erzeugten Superholonen.

**`TerminateHolon(HolonId) void`**
Beendet den Superholon, der durch die angegebene HolonId identifiziert wird. Alle Subholonen werden darüber informiert. Nur der Kopf des Superholonen kann ihn beenden.

**`GetAgentsInRole(Role, HolonId) List<SimObject>`**
Liefert eine Liste von Agenten, die im angegebenen Holon die vorgegebene Rolle einnehmen.

**`CanAgentBeCastToRoleInHolon(SimObject, Role, HolonId) bool`**
Kontrolliert, ob der Agent (SimObject) eine der notwendigen Fähigkeiten für die Rolle (Role) im Holon (HolonId) besitzt.

**`IsMemberOfHolon(SimObject, HolonId) bool`**
Kontrolliert, ob der Agent (SimObject) bereits Mitglied in einem Holon (HolonId) ist.

**`IsMemberOfHolonOfHolonType(SimObject, HolonType) bool`**
Überprüft, ob der Agent (SimObject) bereits in einem Holon vom Typ HolonType Mitglied ist.

**`CastAgentForRoleInHolon(SimObject, Role, HolonId) void`**
Nimmt den Agenten (SimObject) in der Rolle (Role) im Holon (HolonId) auf.

**`RemoveAgentForRoleInHolon(SimObject, Role, HolonId) void`**
Entfernt den Agenten (SimObject) aus der Rolle (Role) im Holon (HolonId).

**`GetNumberOfAgentsInRole(Role, HolonId) Number`**
Liefert die Anzahl der Agenten, die gegenwärtig die Rolle *Role* im Holon *HolonId* einnehmen.

**`SetRequiredNumberOfAgentsInRole(HolonId, Role, Number) void`**
Legt die Anzahl (Number) der benötigten Agenten in einer Rolle für den Holon fest. Hiermit kann die Kardinalität einer Rolle zur Laufzeit geändert werden. Diese Änderung bezieht sich nur auf einen Holon, nicht auf alle Holonen dieses Typs.

**`GetRequiredNumberOfAgentsInRole(HolonId, Role) Number`**
Liefert die Anzahl der Agenten, die eine Rolle (Role) im Holon (HolonId) besetzen müssen.

## A.2.2 Primitive für die Kommunikation mit Subholonen

Durch die Verwendung der folgenden Primitive kann die Kommunikation zwischen Kopfholon und potentiellem Subholon zwecks des Beitritts in einen Holon vereinfacht werden. Der erste, größere Teil der Primitive dient dabei der Bearbeitung eines Angebots zum Beitritt in einen Holon. Dieses Angebot (*offer*) wird durch den Kopfholon initiiert. Die letzten Primitive in der Liste dienen für den Fall, dass ein potentieller Subholon beim Kopfholon für eine Rolle anfragt (*request*).

## Der Kopfholone bietet eine Rolle an

**`OfferRole(SimObject, HolonType, Role) void`**
Biete dem Agenten SimObject die Rolle Role in einem Holon vom Typ HolonType an.

**`OfferRole(SimObject, Number, HolonType, Role) void`**
Wie das vorherige OfferRole-Primitiv mit einem zusätzlichen numerischen Parameter, der beispielsweise die Dringlichkeit eines Angebots ausdrücken kann.

**`WasAnswerToOfferReceived(HolonType, Role) bool`**
Stellt fest, ob eine Antwort auf ein Angebot eingegangen ist. HolonType und Role können detailiert vorgegeben werden. Durch die Übergabe des Wertes **AnyType** für HolonType können Antworten auf ein Angebot für eine Rolle in einem beliebigen Typ abgefragt werden. Bei Verwendung des vordefinierten **AnyRole**-Wertes für Role, können Antworten für eine beliebige Rolle abgefragt werden. Eine Kombination ist möglich, d.h. Antworten auf ein beliebiges Angebot können mit `WasAnswerToOfferReceived(AnyType, AnyRole)` abgefragt werden.

**`WasAnswerToOfferReceived(SimObject) bool`**
Stellt fest, ob vom vorgegebenen Agenten SimObject eine Antwort eingegangen ist.

**`PopAnswerToOffer(SimObject, HolonType, Role) SimObject`**
Liefert die Antwort des Agenten *SimObject* auf ein Angebot der Rolle *Role* in einem Holon vom Typen *HolonType*. Die Liste der Antworten wird intern gelöscht.

Eine **Antwort** besteht im Falle einer Zustimmung aus einer Referenz auf den antwortenden Agenten (SimObject) und im Falle einer Ablehnung aus einer `null`. Dieser Ansatz wurde gewählt, weil er die bequeme Aufnahme des antwortenden Agenten in den Holonen mit `CastAgentForRole(SimObject, Role, HolonId)` ermöglicht.

**`PopAnswersToOffer(HolonType, Role) List<SimObject>`**
Liefert alle Antworten (in Form von Referenzen auf den Antwortenden) auf Angebote für die Rolle *Role* im Typ *HolonType*. Die Liste der Antworten wird intern gelöscht.

**`GetAnswerToOffer(SimObject) SimObject`**

Gibt die Antwort eines einzelnen Agenten (SimObject) zurück. Wenn die Antwort nicht `null` ist, so ist es eine Referenz auf den Antwortenden.

**`GetAffirmativeAnswersToOffer(HolonType, Role) List<SimObject>`**

Liefert eine Liste von Agenten, die dem Angebot der Rolle *Role* in einem Holon vom Typ *HolonType* zugestimmt haben.

**`ClearAnswerToOffer(SimObject, HolonType, Role) void`**

Entfernt alle Antworten von Agent *SimObject* für die Rolle *Role* im Typen *HolonType* aus der Liste der Antworten.

**`ClearAnswersToOffer(HolonType, Role) void`**

Entfernt alle Antworten betreffend einer Rolle in einem HolonType aus der Liste der Antworten.

**`ConfirmOffer(SimObject, HolonId) void`**

Sendet an *SimObject* die Bestätigung zur Aufnahme in den Holon *HolonId*. Falls der Wert von HolonId `null` ist, so wurde die Aufnahme verweigert.

## Ein Agent fragt nach einer Rolle

**`IsRequestAvailable(HolonType, Role) bool`**

Kontrolliert, ob eine Anfrage für eine Rolle in einem bestimmten Typen eingegangen ist. Wie bei `WasAnswerToOfferReceived(HolonType, Role)` beschrieben können AnyType und AnyRole verwendet werden.

**`PopRequestersForRole(HolonType, Role) List<SimObject>`**

Liefert die Liste von Agenten, die für eine Rolle in einem HolonType angefragt haben. Danach wird die interne Liste der Anfragen gelöscht.

**`SendAnswerToRequest(SimObject, HolonId) void`**

Sendet dem anfragenden Agenten *SimObject* eine Antwort. Ist *HolonId* nicht `null`, ist die Antwort eine Zustimmung, ansonsten eine Ablehnung.

# A.3 Primitive des Subholon-Features

## A.3.1 Primitive für die Selbstverwaltung von Subholonen

Primitive, die allgemeine Funktionalitäten für Subholone bereitstellen werden nachfolgend beschrieben. Dazu gehört das Finden von Kopfholonen, das ändern von Fähigkeiten (Competencies) zur Laufzeit um die erhöhte Gruppenfähgikeit von Superholonen abbilden zu können, die Kommunikation mit anderen Subholonen im selben Superholon und schließlich Primitive um rollenspezifisches Verhalten realisieren zu können.

**`GetHeadsOfHolonType(HolonType) List<SimObject>`**
: Liefert eine Liste von Agenten im System, die Kopfholonen in einem Holon vom angegebenen Typ sind.

**`GetFreeAgents(HolonType, List<SimObject>) List<SimObject>`**
: Filtert aus der übergebenen Liste von Agenten die aus, die in einem Holon vom angegebenen *HolonType* Mitglied sind und gibt die Agenten zurück, die frei sind. Ein Agent ist dann frei, wenn er weder Körper- noch Kopfholon ist.

**`GetHolonsWithSkill(Skill, List<SimObject>)`**
: Filtert aus der Liste diejenigen Agenten aus, die die angegebene Fähigkeit nicht besitzen.

**`GetHolonsWithSkillNeededForRole(Role, List<SimObject>) List<SimObject>`**
: Filtert aus der übergebenen Liste von Agenten die aus, die keine der Fähigkeiten besitzen, die für die angegebene Rolle nötig sind.

**`AddOrChangeCompetency(Competency) void`**
: Die übergebene Competency (Fähigkeit mit Qualitätswert) wird dem Agenten hinzugefügt oder der Qualitäts-Wert einer vorhandenen Competency geändert. Durch Doppelklick auf das Skill-Argument des ComposedTypes 'Competency' wird ein Dialog geöffnet, in dem aus den bereits vorhandenen Skills einer ausgewählt werden, oder ein Neuer erstellt werden kann.

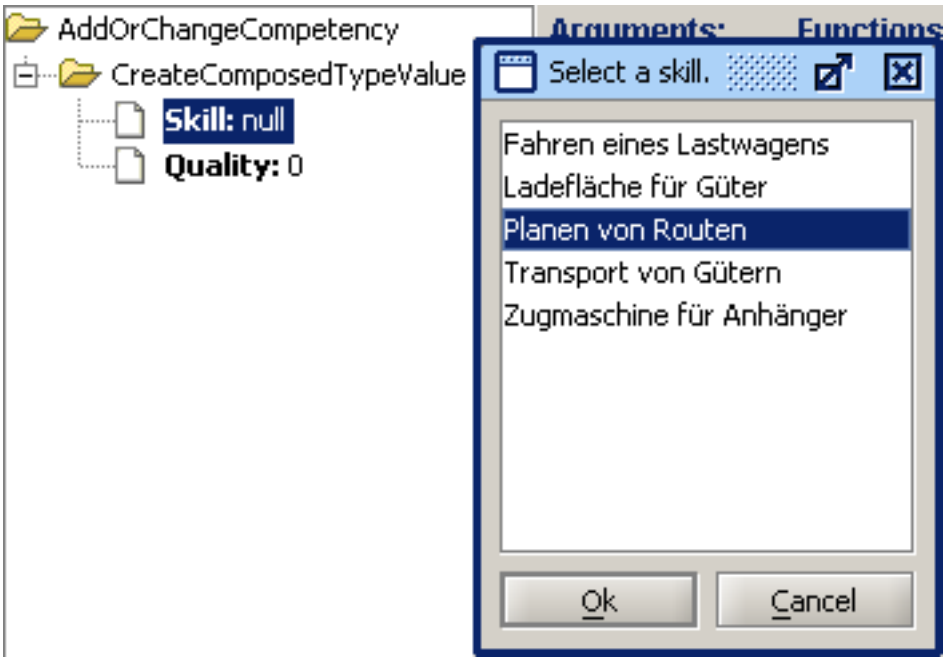



**Abbildung A.1. Auswahl eines Skills für den ComposedType *Competency*.**

**`RemoveCompetency(Skill) void`**

Entfernt die Competency, die durch den angegebenen Skill identifiziert ist.

**`GetAgentsConcerningCommunication(Concern, HolonId) List<SimObject>`**

Liefert eine Liste von Agenten, die im angegebenen Holonen für Interaktion bezüglich *Concern* zur Verfügung stehen. Mit diesem Primitiv werden die Strukturen, wie sie in Abbildung 4.5 definiert werden, abgefragt.

**`IsMemberOfHolon(HolonId) bool`**

Beantwortet die Frage, ob der gegenwärtige Agent im angegebenen Holon Mitglied, entweder Kopfholon oder normaler Subholon, ist.

**`IsRoleInHolon(HolonId, Role) bool`**

Bestimmt, ob der gegenwärtige Agent im angegebenen Holon die Rolle *Role* einnimmt.

**`GetRoleInHolon(HolonId) Role`**

Liefert die Rolle, die der gegenwärtige Agent im Holon *HolonId* einnimmt. Falls er dort keine Rolle einnimmt, wird `null` zurückgegeben.

**`GetHeadOfHolon(HolonId) SimObject`**
: Liefert zum angegebenen Holon den Kopfholon. Falls dieser nicht vorhanden ist, wird `null` zurückgegeben.

**`GetQualityOfSkill(SimObject, Skill) Number`**
: Gibt den Qualitätswert des angegebenen Skill beim Agenten *SimObject* zurück.

**`LeaveHolon(HolonId) void`**
: Benachrichtigt den Kopfholon des Holonen *HolonId* über den Austritt des gegenwärtigen Agenten. Da der Kopfholon nicht widersprechen kann, ist keine weitere Aktivität nötig.

## A.3.2 Primitive für die Kommunikation mit Kopfholonen

Die nachfolgend beschriebenen Primitive erleichtern die Kommunikation zwischen Subholon und Kopfholon für das Anfragen nach Rollen oder das Bearbeiten von Angeboten. Der erste Teil der Primitive eignet sich für eine vom Subholon initiierte Anfrage nach einer Rolle (*request*). Die Primitive im zweiten Teil dienen der Bearbeitung von Angeboten durch Kopfholone (*offer*).

### Der Subholon fragt nach einer Rolle

**`RequestRole(SimObject, HolonType, Role) void`**
: Fragt beim Kopfholon *SimObject* nach einer Rolle in einem Holon vom Typ *HolonType* an.

**`WasAnswerToRequestReceived(SimObject) bool`**
: Kontrolliert, ob eine Antwort auf eine Anfrage beim Kopfholon *SimObject* vorliegt.

**`PopAnswerToRequest(SimObject) HolonId`**
: Liefert die Antwort des Kopfholonen *SimObject* in Form einer HolonId. Falls diese nicht `null` ist, handelt es sich um eine zustimmende Antwort. Danach wird die Antwort intern gelöscht.

**`GetAnswerToRequest(SimObject) HolonId`**
: Wie `PopAnswerToRequest`, jedoch ohne die Antwort zu löschen.

**`ClearAnswerToRequest(SimObject) void`**

Löscht die Antwort des Kopfholonen *SimObject* auf die Anfrage nach Mitgliedschaft in einem Holon, den SimObject als Kopfholon repräsentiert.

**`IsAnswerAffirmative(HolonId) bool`**

Gibt zurück, ob die Antwort *HolonId* auf eine Anfrage positiv war. Diese Primitiv ist eine Abkürzung für `Not(Equals(HolonId, null))` und kann mit Primitiven verwendet werden, die eine HolonId zurückliefern. Die HolonId wird als Antwort eines Kommunikationsvorgangs betrachtet und wird erzeugt, wenn ein Kopfholon mit `SendAnswerToRequest(SimObject, HolonId)` auf eine Anfrage antwortet. Die Verwendung der HolonId als Antwort hat den Vorteil, dass bei einer positiven Antwort die HolonId direkt weiterverwendet werden kann.

## Ein Kopfholon bietet eine Rolle an

**`IsOfferAvailable(HolonType, Role) bool`**

Kontrolliert, ob ein Angebot für eine Rolle im angegebenen HolonType vorliegt.

**`PopOffersForRole(HolonType, Role) List<Offer>`**

Liefert eine Liste von Angeboten, für die Rolle *Role* in Holonen vom Typ *HolonType*. Danach werden die Angebote intern gelöscht.

**`PopOfferersForRole(HolonType, Role) List<SimObject>`**

Wie PopOffersForRole nur, dass anstatt der Angebote die Kopfholonen zurückgegeben werden, die die Angebote geschickt haben. Dies ist eine Abkürzung für den Fall, das der Wert der Angebote nicht von Interesse ist. Da die Liste der Angebote nach Aufruf des Primitives intern gelöscht wird, sind die Werte verloren. Ein nachträgliches Abfragen der Werte ist nicht möglich.

**`GetOffererOfOffer(Offer) SimObject`**

Liefert den Kopfholon, der das Angebot *Offer* geschickt hat.

**`GetValueOfOffer(Offer) Number`**

Liefert den Wert eines Angebots.

**`ClearOffersForRole(HolonType, Role) void`**
Entfernt alle Angebote für eine Rolle in einem HolonType.

**`SendAnswerToOffer(SimObject, Boolean) void`**
Sendet eine Antwort auf das Angebot vom Kopfholonen *SimObject*. *Boolean* gibt an, ob die Antwort zustimmend ist.

**`WasConfirmOfferReceived(SimObject) bool`**
Kontrolliert, ob eine Bestätigung für ein Angebot von Kopfholon *SimObject* empfangen wurde.

**`PopOfferConfirmation(SimObject) HolonID`**
Liefert die Bestätigung für ein Angebot von Kopfholon *SimObject* als HolonId. Falls diese nicht `null` ist, wurde das Angebot bestätigt. Danach wird die Bestätigung intern gelöscht.

**`GetOfferConfirmation(SimObject) HolonId`**
Wie `PopOfferConfirmation`, jedoch ohne die Antwort zu löschen.

**`ClearOfferConfirmation(SimObject) void`**
Entfernt die Bestätigung für ein Angebot vom Kopfholonen *SimObject*.

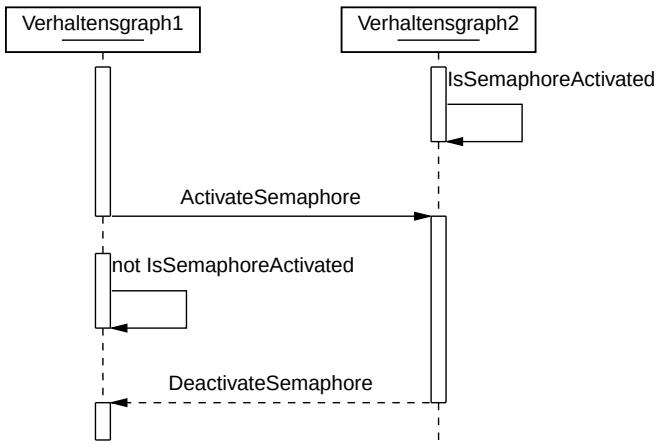
### A.3.3 Primitive für den Wechsel zwischen Verhaltensgraphen

Man kann das Verzweigen in unterschiedliche Verhaltensgraphen als ein Mittel zur Entwirrung komplexer Verhaltensgraphen sehen. Die nachfolgenden Primitive erleichtern das Verzweigen. Die Bezeichnung Semaphore ist etwas irreführend, da strenggenommen keine Semaphoren verwendet werden. Vielmehr handelt es sich um ein einfaches Signaling-Konzept.

**`ActivateSemaphore(String) void`**
Aktiviert die Semaphore, die mit dem String *String* identifiziert wird.

**`DeactivateSemaphore(String) void`**
Deaktiviert die Semaphore *String*.

**`IsSemaphoreActivated(String) bool`**
Kontrolliert, ob die angegebene Semaphore aktiviert ist.




Verhaltensgraph2 (VG2) wartet darauf, dass eine Semaphore aktiviert wird. Diese wird von Verhaltensgraph1 (VG1) schließlich aktiviert. VG1 wartet nun auf die Deaktivierung der Semaphore durch VG2, um mit seinen Berechnungen fortzufahren.

**Abbildung A.2. Verwendung der Semaphore**

# Danksagung

Ich möchte mich bei Prof. Dr. Frank Puppe für das Zustandekommen dieser Diplomarbeit und für das Aufbringen interessanter Fragestellungen bedanken.

Meinem Betreuer Dipl.-Inf. Manuel Fehler möchte ich danken, dass er mir bei Problemen half und mich ab und zu auf den rechten Pfad zurückführte.

Bei Dr. Franziska Klügl-Frohnmeyer bedanke ich mich dafür, dass sie mir die Tür in die interessante Welt der Multiagentensysteme geöffnet hat.

Desweiteren möchte ich mich bei denen bedanken, die mich durch anregenden Unterhaltungen, Diskussionen und Monologe auf interessante Aspekte der Modellierung in SeSAm, der Programmierung in Java und der Holonik aufmerksam machten: Hans Peter Maurer, Clemens Mühlberger, Reinhard Hatko, Tanja Credner und Marcus Scholz.

Bedanken möchte ich mich auch bei meinen Eltern, ohne die ich nicht hätte studieren können und die mich immer vollends unterstützt haben.

Besonders möchte ich mich aber bei Barbara bedanken, ohne die ich das alles nicht geschafft hätte.

# Abkürzungsverzeichnis

**ACL**
Agent Communication Language

**ADS**
Agent Directory Service

**AOP**
Agent-Oriented Programming

**AOSE**
Agent-Oriented Software Engineering

**CFP**
Call For Proposal

**CNP**
Contract Net Protocol

**COP**
Component-Oriented Programming

**CPL**
Cooperative Planning Layer; Eine Schicht in der InterRRaP-Architektur von Müller et al. [Mü93].

**DA**
Durchschnittliches Alter

**ECNP**
Extended Contract Net Protocol

**FIPA**
Foundation for Intelligent Physical Agents

**HAP**
Holonic Augmentation Plugin; Die holonische Erweiterung von SeSAm

**HMS**
Holonic Manufacturing Systems

**HOP**
Holon-Oriented Programming

**IMS**
Intelligent Manufacturing Systems

**KQML**
Knowledge Query and Manipulation Language

**MAS**
Multiagentensystem

**OOP**
Object-Oriented Programming

**OR**
Operation Research

**PnEU**
Plan'n'Execute Unit

**RAP**
Resource Allocation Protocol

**SeSAm**
Shell for Simulated Agent Systems; Eine am Lehrstuhl 6 für Künstliche Intelligenz und Angewandte Informatik entwickelte Entwicklungs- und Simulationsumgebung für Multiagentensysteme.

**SN**
Strategisches Netzwerk

**SODA**
Societies in Open and Distributed Agent spaces

**ST**
Simulated Trading

**UML**
Unified Modelling Language

**VG**

Verhaltensgraph

**VU**

Virtuelles Unternehmen

**VÜ**

Verteiltes Überwachen

# Literaturverzeichnis


[Ba94] Achim Bachem, Winfried Hochstättler und Martin Malich. *Simulated Trading - A New Parallel Approach For Solving Vehicle Routing Problems*. Erschien in Advances in Parallel Computing. 9(125): 471-475. 1994.

[Ba98] Albert Baker. *A Survey of Factory Control Algorithms which Can be Implemented in a Multi-Agent Heterarchy*. Dispatching, Scheduling, and Pull. Journal of Manufacturing Systems. 1998.

[BF99] S. Bussmann und D.C. McFarlane. *Rationales for Holonic Manufacturing Control*. In Proc. of Second Int. Workshop on Intelligent Manufacturing Systems. 177 - 184. 1999.

[BR05] *Bounded Rationality: A Response to Rational Analysis*. http://ai.eecs.umich.edu/cogarch0/common/theory/boundrat.html. (01/2005).

[Bu97] B. Burmeister, A. Haddadi und G. Matylis. *Application of multi-agent systems in traffic and transportation*. Erschienen in Software Engineering. IEE Proceedings. 144(1): 51-60. 1997.

[CN05] *FIPA Contract Net Interaction Protocol Specification*. http://fipa.org/specs/fipa00029/. (02/2005).

[CP05] *Communication-Plugin von SeSAm*. http://ki.informatik.uni-wuerzburg.de/~sesam/wiki/pmwiki.php/Main/CommunicationPlugin. (03/2005).

[Cr04] Tanja Credner. *Emergentes Monitoring - Untersuchungen in einer Multiagentensimulation*. Studienarbeit am Lehrstuhl 6 der Informatik an der Universität Würzburg. 2004.

[Da04] M. Dastani, J. Hulstijn, F. Dignum und J.-J. C. Meyer. *Issues in multiagent system development*. In Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04). 2004.

[De01] Scott DeLoach. *Analysis and Design using MaSE and agentTool*. In 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS). 2001.

[De92] Martin Desrochers, Jacques Desrosiers und Marius Solomon. *A new optimization algorithm for the vehicle routing problem with time windows*. Erschien in Operations Research. 40(2): 342-354. 1992.

[FG96] Stan Franklin und Art Graesser. *Is it an Agent, or just a Program?*. A Taxonomy for Autonomous Agents. Springer-Verlag. Erschienen in *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*. 1996.

[Fi03] Klaus Fischer, Michael Schillo und Jörg Siekmann. *Holonic Multiagent Systems*. A Foundation for the Organisation of Multiagent Systems. In *Proceedings of International Conference on Applications of Holonic and Multiagent Systems (HoloMAS)*, Prag, Tschechische Republik, September 2003.. 2003.

[Fi97a] Klaus Fischer, Hans-Jürgen Bürckert und Gero Vierke. *TeleTruck: A Holonic Fleet Management System*. Technical Memo TM-97-03, DFKI Saarbrücken. 1997.

[Fi97b] Klaus Fischer, Jörg P. Müller, Markus Pischel und Darius Schier. *A model for Cooperative Transportation Scheduling*. The MIT Press. Proceedings of the First International Conference on Multiagent Systems, June, 1995, San Francisco, California, USA. 1997.

[Ge99] Christian Gerber, Jörg Siekmann und Gero Vierke. *Holonic Multi-Agent Systems*. Research Report RR-99-03, DFKI Saarbrücken. 1999.

[GR91] Anand S. Rao und Michael P. Georgeff. *Modeling Rational Agents within a BDI-Architecture*. appeared in Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91): 473-484. 1991.

[HH05] Eberhard von Goldammer, Joachim Paul und Joe Newbury. *Heterarchy - Hierarchy*. Two complementary categories of description. http://www.vordenker.de/heterarchy/a_heterarchy-e.pdf. (01/2005).

[HI04] Vincent Hilaire. *Holonic Multi-Agent Systems*. Präsentation auf AgentLink III in Ljubljana, Slowenien. 2004.

[IS92] Toru Ishida, Les Gasser und Makoto Yokoo. *Organization Self-Design of Distributed Production Systems*. IEEE Transactions on Knowledge and Data Engineering. 4(2): 123-134. 1992.

[Je00] Nicholas R. Jennings. *Agent-Oriented Software Engineering*. Keynote-presentation given at ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). 2000.

[Je94] T. Wittig, Nicholas R. Jennings und E. H. Mamdani. *ARCHON - A Framework for Intelligent Cooperation*. IEE-BCS Journal of Intelligent Systems Engineering - Special Issue on Real-time Intelligent Systems in ESPRIT. 3(3): 168-179.. 1994.

[Je96] Nicholas R. Jennings. G. M. P. O'Hare and N. R. Jennings. *Coordination Techniques for Distributed Artificial Intelligence*. John Wiley & Sons, New York. Erschienen in *Foundations of Distributed Artificial Intelligence*. 187-210. 1996.

[JG05] *JGraph und JGraphT, Bibliotheken zur Darstellung von Graphen*. http://www.jgraph.com/ und http://jgrapht.sourceforge.net/. (03/2005).

[Kl04] Franziska Klügl. *Multi-Agent Simulation*. http://ki.informatik.uni-wuerzburg.de/~kluegl/teach/EASSS2004%20Multi%20Agent%20Simulation.pdf. (01/2005).

[Ko67] Arthur Koestler. *The ghost in the machine*. Hutchinson. 1967.

[MS03] Michael Schillo. *Self-organization and adjustable autonomy*. Two sides of the same medal. Connection Science. 14(4): 345-359. 2003.

[Mü93] J. Müller und M. Pischel. *The Agent Architecture InteRRaP: Concept and Application*. Technical Report RR-93-26, DFKI Saarbrücken. 1993.

[NI01] Geoff Nitschke. *Cooperating Air Traffic Control Agents*. Erschienen in Applied Artificial Intelligence. 15(2): 209-235. 2001.

[Om00] Andrea Omicini. *SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems*. Springer-Verlag, Heidelberg. Erschienen in *Lecture Notes in Computer Science* 1957: 185-193. 2000.

[Sh93] Yoav Shoham. *Agent-oriented programming*. Artificial Intelligence. 60(1): 51-92. 1993.

[Si69] Herbert Simon. *The sciences of the artificial*. MIT Press, Cambridge, Mass.. 1969.

[Sy98] Katia Sycara. *MultiAgent Systems*. AI Magazine 19(2). 1998.

[Sz98] Cuno Pfister und Clemens Szyperski. *Why objects are not enough*. Cambridge University Press New York, NY, USA. Proceedings of the first component user's conference on Component-based software engineering. 141-147. 1998.

[Tv00] A. Tveit. *A survey of Agent-Oriented Software Engineering*. Report, Norwegian University of Science and Technology. 2000.

[Vo04] Danna Voth. *Holonics in Manufacturing: Bringing Intelligence Closer to the Machine*. Erschienen in IEEE Intelligent Systems. 19(6): 4-7. 2004.

[Wa02] R. Watson. *Modular interdependency in complex dynamical systems*. In *Proceedings of the Alife VIIIWorkshop on the Modelling of Dynamical Hierarchies*: 81-88. University of New South Wales, Sydney, Australia. 2002.

[Wa98] William E. Walsh und Michael P. Wellman. *A Market Protocol for Decentralized Task Allocation*. In Third International Conference on Multi-Agent Systems. 325-332. 1998.

[WC00] Michael Wooldridge und Paolo Ciancarini. *Agent-Oriented Software Engineering: The State of the Art*. Springer-Verlag, Berlin. in: First Int. Workshop on Agent-Oriented Software Engineering. 1957: 1-28. 2000.

[We00] Gerhard Weiss. *Multiagent systems*. A modern approach to distributed artificial intelligence. The MIT Press Cambridge, Massachusetts; London, England. 2000.

[WJ00] Michael Wooldridge und Nicholas R. Jennings. *Agent-Oriented Software Engineering*. in: J. Bradshaw (Ed.), Handbook of Agent Technology, AAAI/MIT Press, 2000, to appear.. 2000.

[WJ95] Michael Wooldridge und Nicholas R. Jennings. *Intelligent Agents*. Theory and Practice. *Knowledge Engineering Review* 10(2). 1995.

[Wo00] Michael Wooldridge, Nicholas R. Jennings und David Kinny. *The Gaia Methodology for Agent-Oriented Analysis and Design*. Erschienen in Autonomous Agents and Multi-Agent Systems. 3(3): 285-312. 2000.

[Wr04] Todd Wright. *Naming Services in Multi-Agent Systems: A Design for Agent-based White Pages*. Erschienen in *Third International Joint Conference on Autonomous Agents and Multiagent Systems*. 3: 1478-1479. 2004.