Multiagentenbasiertes Planen unabhängiger Aufgaben mit Soft Constraint Solving in elektronischen Märkten

Diplomarbeit

von
Sven Jacobi

nach einem Thema von Prof. Dr. Jörg H. Siekmann
im Fachbereich 6.2, Informatik, der Universität des Saarlandes

28. Februar 2002

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich diese Arbeit selbständig verfaßt, nur die im Literaturverzeichnis zitierten Quellen benutzt und sie noch keinem anderen Prüfungsamt vorgelegt habe.

Saarbrücken, im Februar 2002 Sven Jacobi

Danksagung

An dieser Stelle möchte ich mich bei Herrn Prof. Siekmann für die Vergabe des interessanten Themas bedanken und der Möglichkeit, diese Arbeit an seinem Lehrstuhl durchführen zu können.
Im ganz besonderen Maße möchte ich mich bei Michael Schillo für die hervorragende Betreuung und Unterstützung während der Anfertigung dieser Arbeit bedanken.
Desweiteren danke ich meiner Freundin Manuela Ziegler für die moralische Unterstützung.
Meinen Freunden Christian Fink, Pascal Schüler, Rainer Siedle und Tore Knabe danke ich für die konstruktive Kritik.

Zusammenfassung

Die vorliegende Diplomarbeit beschäftigt sich mit der Anwendung von *Multiagentensystemen* und dem Lösen von Planungsproblemen sowie *Hierarchical Constraint Satisfaction Problems*. Es wird gezeigt, wie man diese Forschungsgebiete in einem anwendungsspezifischem Szenario effizient nutzen kann.

In einem elektronischen Markt muß ein Disponent für einen potentiellen Auftrag, welchen er aus diesem Markt erhält, dessen Erfüllbarkeit an den ihm zur Verfügung stehenden Ressourcen überprüfen. In diese fließt unter anderem auch die natürliche Arbeitskraft der Arbeiter ein, welche diesen Auftrag bearbeiten können. Nun haben diese Präferenzen bezüglich ihrer Arbeitszeit, die einen negativen Einfluß auf die Ressourcen haben. In dieser Arbeit wird ein System entwickelt und implementiert, welches *online* Aufträge annehmen und einplanen kann und dabei die Wünsche der Arbeiter bestmöglichst respektiert, solange die Erfüllung des Auftrags dabei nicht verhindert wird.

Die beteiligten Akteure, also Disponent und Arbeiter, werden als Softwareagenten mit getrennten Teilaufgaben zur Gesamtlösung dieser Aufgabe auf einer Plattform, welche den elektronischen Markt darstellt, realisiert. Der Arbeiter kann seine Wünsche mit Präferenzen belegen, womit eine sogenannte *Constrainthierarchie* entsteht, welche von einem in dieser Arbeit begründet ausgewähltem Solver gelöst wird. Im Softwareagenten, welchen den Arbeiter repräsentiert, werden aus den Wünschen in Zusammenhang mit der Leistungsfähigkeit der Maschine mittels *soft constraint solving* die Gebote berechnet, welche dem Disponenten als Grundlage für seine letztliche Verteilung dienen. Im Disponenten werden die Aufträge schließlich auf die einzelnen Arbeiter mit ihren Maschinen verteilt. Hierfür berechnet er anhand einer Menge von Geboten der jeweiligen Arbeiter die günstigste Verteilung bezüglich der Wünsche, welche mittelbar in den Geboten der Arbeiter enthalten sind, und der Auslastung der einzelnen Maschinen.

In der Evaluierung zeigt sich, dass dieses Planen der unabhängigen Aufgaben mit dem vorangestellten *soft constraint solving* in einem Multiagentensystem effizient eingesetzt werden kann. Gerade unter dem Aspekt der praktischen Relevanz liefert das System in ansprechender Zeit Ergebnisse, welche man durch ”manuelles” Planen mit dieser Verteilung und dieser Zeit nicht erreichen würde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In fast allen geschäftlichen Prozessen, in denen die beteiligten Institutionen (z.B. Firmen) miteinander kooperieren müssen, kommt es zu Zielkonflikten zwischen den einzelnen Instanzen, da ihre Ziele meist miteinander konkurrieren. Folgendes Beispiel soll einen Beleg hierfür geben.

In einem elektronischen Markt werden Aufträge zwischen Firmen oder anderen Institutionen verhandelt und verteilt, die in der Agora, dem Begegnungsraum des Marktes, durch Agenten repräsentiert werden. Die Schnittstelle zwischen den einzelnen Institutionen bilden hierbei die Disponenten, welche sich für die letztendliche Entscheidung über die Annahme des Auftrags und dessen tatsächlicher Realisierung in ihrer Firma verantwortlich zeichnen. Hierfür müssen sie für jeden einzelnen potentiellen Auftrag intern prüfen, ob dieser überhaupt erfüllbar ist und welche Kosten dabei entstehen. Er muß somit intern mit allen Arbeitern, die diesen Auftrag bearbeiten könnten, Kontakt aufnehmen und prüfen, ob deren bisherige Auftragslage noch genügend Spielraum für diesen weiteren hat.

Den Arbeitern sei wiederum erlaubt, Wünsche bezüglich ihrer Arbeitszeit zu formulieren und mit in ihre Antworten einfließen zu lassen. Diese Präferenzen haben natürlich Einfluß auf die Arbeitszeit, die sich je nach Gewährung der Art der Wünsche um so mehr verringert. Sie geben dem Disponenten eine begrenzte Anzahl von möglichen Antworten, aus welchen der Disponent dann schließlich seine Lösung bestimmen soll, womit das hierarchische Prinzip erhalten bleibt. Dieses Prinzip liegt in der Tatsache, dass der Disponent letztlich doch entscheiden kann und muß, welchem Arbeiter er welche Wünsche erfüllt, um den Auftrag erfüllen zu können. Der Arbeiter ist also abhängig von der Entscheidung des Disponenten.

Der Konflikt, der hierbei entstehen kann, liegt zwischen der Kostenoptimierung seitens des Disponenten und den Präferenzen bezüglich der Arbeitszeiten auf Seiten der Arbeiter. Der Disponent möchte einerseits den Auftrag möglichst schnell und kostengünstig erfüllen, was er durch eine optimale Auslastung der Maschinen, welche die

Arbeiter bedienen, erreichen würde, aber auf der anderen Seite will er auch die Wünsche der Arbeiter respektieren, welche jedoch der optimalen Auslastung der Maschine widersprechen.

In dieser Arbeit wird ein System entwickelt und beschrieben, welches diesen Konflikt agentenbasiert löst. Es werden Programmeinheiten entwickelt, die gewisse Teilaufgaben, welche zur Gesamtlösung notwendig sind, autonom erledigen. Die Behandlung von Präferenzen bei Bedingungen, dem *soft constraint solving*, wird dabei von einem Solver erledigt, welcher die hierbei entstehenden Constrainthierarchien lösen kann. Eine Constrainthierarchie besteht aus mehreren Mengen von Constraints verschiedener Priorität, wobei eine Menge Bedingungen mit gleicher Gewichtung enthält. Dieses *soft constraint solving* etwa wird von der letztlichen Entscheidung bezüglich der Erfüllbarkeit potentieller Aufträge getrennt, um das gesamte System somit flexibler gestalten zu können.

Ein weiterer Aspekt dieser Arbeit liegt in der Anwendbarkeit des entwickelten Systems. Solche Märkte sind meist räumlich begrenzt sowie auch von der Kommunikation her eingeschränkt. Ein System, welches verschiedene Medien zur Datenerfassung und -übertragung unterstützt etwa über das Internet, *Personal Digital Assistants* (PDA) oder sogar mobile Telefone, die das *Wireless Application Protocol* (WAP) unterstützen, würde die Einsatzgebiete solcher Systeme erheblich vergrößern. Aufgrund der verschiedenen Anwendungsmöglichkeiten eines solchen Systems bilden auch die Perfomanz sowie eine benutzertaugliche Oberfläche wichtige Aspekte in dieser Arbeit.

1.2 Ergebnisse

Das im Verlauf dieser Arbeit entworfene und implementierte System zeigt, dass man das Lösen Planungsproblemen und *Hierarchical Constraint Satisfaction Problems* effizient in einem Multiagentensystem anwenden kann. Gerade vor dem Hintergrund der anwendungsspezifischen Problemstellung wird gezeigt, dass die Erkenntnisse aus der Agententechnologie, speziell durch das Aufteilen des gestellten Problems in kleinere Teilaufgaben, welche von unabhängigen Agenten erledigt werden, in dieser Arbeit von großer Bedeutung sind. Durch die Kapselung des eigentlichen *Scheduling* von dem *soft constraint solving* wird nicht nur die komplexe Aufgabe vereinfacht, sondern es werden somit auch die eigentlichen Problementitäten bewahrt. In der Evaluierung des implementierten Systems werden die Ergebnisse noch ausführlicher diskutiert.

1.3 Gliederung

Im folgenden Kapitel 2 wird ein Überblick über die relevante Literatur sowie die theoretischen Grundlagen gegeben. Es bildet somit die Basis für die anderen Kapitel der Arbeit. Zunächst wird der Bereich der Agententechnologie näher betrachtet und die wichtigsten Begriffe aus diesem Bereich definiert. Es folgt ein Überblick über das

Constraint Solving, welches einen weiteren Schwerpunkt der Diplomarbeit bildet. Genauer wird auf das Hierarchische Constraint Solving, welches eine Variante des ursprünglichen ist, eingegangen und Systeme betrachtet, welche Probleminstanzen aus diesem Bereich lösen können. Danach wird das Scheduling näher betrachtet, welches einen dritten Hauptaspekt der Arbeit bildet.

Im dritten Kapitel der Arbeit wird als Fortführung dieser Einleitung die Problemstellung detailliert beschrieben und die Diplomarbeit in die Forschungsgebiete eingeordnet, deren Erkenntnisse sie sich zu nutzen macht. Hierbei sollen unter anderem die speziellen Herausforderungen, die sich aus der Problemstellung ergeben und sich auf die in Kapitel 2 erläuterten Grundlagen beziehen, hervorgehoben werden.

In Kapitel 4 wird eine Spezifikation gegeben, welche sich auf die detaillierte Problemstellung aufbaut. Dabei sollen die Ziele, die mit dieser Arbeit verfolgt werden, herauskristallisiert werden. Lösung dieses Kapitels ist die Beantwortung der Frage, was erreicht werden soll. Die Frage, wie man diese Ziele erreicht, wird im darauf folgenden Kapitel 5 beantwortet. Dies umfaßt sowohl die Implementierung der im selben Kapitel erarbeiteten Formalismen als auch ihre Evaluierung. Diese soll anhand einer systematischen Untersuchung die Leistungsfähigkeit des implementierten Systems belegen.

Schließlich wird in Kapitel 7 die Arbeit mit ihren Resultaten noch einmal zusammengefaßt. Desweiteren wird ein Ausblick über potentielle Folgearbeiten gegeben, die sich im Verlauf der Bearbeitung und Lösung als sinnvolle und lohnenswerte Folgeziele herausgebildet haben.

Kapitel 2

Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für die vorliegende Arbeit erörtert. Dabei werden zunächst die wichtigsten Konzepte aus dem Bereich der Multiagentensysteme (MAS) beschrieben. Im folgenden Abschnitt wird dann ein Überblick über das *Constraint Solving* gegeben. Es werden die wichtigsten Begriffe aus diesem Bereich definiert, die im Verlauf dieser Arbeit immer wieder benötigt werden. Darüberhinaus werden die bestehenden Ansätze zur Lösung solcher Aufgaben diskutiert und verschiedene existierende Systeme verglichen. Im letzten Abschnitt wird dann das *Scheduling of Independent Tasks* (SIT), welches auch als *Constraint Solving Problem* formuliert werden kann, genauer beschrieben und analysiert.

2.1 Agententechnologie

2.1.1 Multiagentensysteme

Der Forschungsbereich der Verteilten Künstlichen Intelligenz (VKI), welcher in den achtziger Jahren begründet wurde, ist ein Teilbereich der Forschungsrichtung der Künstlichen Intelligenz (KI). In dieser Zeit entwickelten sich schließlich auch die ersten Ansätze der Multiagententechnologie. Einer der Motivationsaspekte ist, dass es komplexe Probleme gibt, deren Lösung man besser auf mehrere kleinere Einheiten mit Teilaufgaben verteilt als den gesamten Lösungsweg zentral von einer Einheit durchführen zu lassen.

Durch das Konzept der Verteilung der Aufgaben bei einem bestimmten Problem verspricht man sich mehrere Vorteile. Erstens können die zur Problemlösung notwendigen Fähigkeiten derart verschieden sein, dass über sie ein einzelner Akteur gar nicht mehr verfügen kann. Man betrachte hierzu zum Beispiel die Entwicklung eines neuen Automobils. Bei dieser komplexen Aufgabe sind schon die verschiedensten Ingenieurfähigkeiten gefragt von der Art der Motorisierung bis hin zur Karosserie in jeglichen Details, welche ein einzelner Akteur gewiß nicht haben kann. Auch könnte das Problem von sich aus schon rein vom räumlichen Aspekt und dem hierauf bezogenen

Wissen her eine Verteilung verlangen, wie dies zum Beispiel bei der Flugleitkontrolle internationaler Flughäfen der Fall ist. Ein weiterer Vorteil ergibt sich aus der Modulierung solcher speziellen Einheiten in Bezug auf deren Flexibilität, Reaktionszeit und Modularisierung [Chaib-draa and Levesque, 1994]. Eine Veränderung kann nämlich nur einen kleinen Bereich des gesamten Prozesses betreffen, der dann in der dafür zuständigen Einheit auch erkannt und bearbeitet werden kann. Alle anderen Einheiten tangiert diese Modifikation gegebenenfalls nicht, womit sie auch keinen Handlungsbedarf haben. Dagegen könnte solch eine Veränderung in einer einzigen großen Einheit schon einen erheblichen Mehraufwand verursachen.

Einhergehend mit diesem Konzept beschäftigt sich die VKI daher auch mit der Verteilung der Intelligenz auf die einzelnen beteiligten Einheiten, der Interaktion zwischen diesen, der gesamten Koordination und der Robustheit sowie Dauerhaftigkeit bei auftretenden Fehlern. Folgende Postulate an das gesamte Verfahren der Multiagententechnologien kristallisierten sich im Verlauf der ersten Ansätze dabei heraus:

- verteilte Problemlösung
- dynamische Anpassung bei Veränderungen in der Realität
- offene Architekturen, die zur Laufzeit modifizierbar sind
- Robustheit und Dauerhaftigkeit bei Ausfall

Da die Forschung bezüglich Agententechnologien noch sehr jung ist, existieren über die einzelnen Begriffe zahlreiche verschiedene Definitionen. Die verschiedenen Einheiten mit kleinen Aufgaben zur Lösung eines komplexen Problems, wie eingangs des Kapitels beschrieben, werden in der VKI als *Agenten* bezeichnet. Von den etlichen Definitionen ist die am weitesten verbreitete und akzeptierte die von Russel und Norvig [Russell and Norvig, 1995].

Definition 1 (Agent nach Russel & Norvig, 1996)
Ein **Agent** *ist eine Einheit, von der man sagen kann, dass sie ihre Umwelt über Sensoren wahrnimmt und sie mit Hilfe von Effektoren beeinflußt.*

Diese Definition beinhaltet jedoch nicht alle zentralen Begriffe, die sich im Laufe der Entwicklung von verschiedenen Autoren wie etwa von Wooldridge & Jennings (1995) oder von Weiss (1996) herausgebildet haben und Agenten mit folgenden Attributen charakterisieren [Wooldridge and Jennings, 1995], [Weiss and Sen, 1996]:

- **Autonomie**: Agenten arbeiten ohne externe Intervention und besitzen Selbstkontrolle.
- **Reaktivität**: Agenten haben die Fähigkeit zur selektiven Wahrnehmung ihrer Umwelt und reagieren auf Veränderungen adäquat.

- **Sozial Fähigkeiten**: Fähigkeit zur Kommunikation und Kooperation mit anderen Agenten zwecks gemeinsamer Ziele.

- **Pro-Aktivität**: Fähigkeit zur Eigeninitiative, also zielgerichtetes Handeln ohne konkrete Anfrage oder Veränderung ihrer Umwelt.

Die vier aufgeführten Attribute bezeichnen Wooldridge und Jennings (1995) als *schwache Agenteneigenschaft* (*weak notion of agency*). Als Pendant hierzu umfassen für sie die *starken Agenteneigenschaften* (*strong notion of agency*) eher mentale Merkmale wie **Wissen**, **Glaube**, **Intention** und **Verpflichtung**. Weiss (1996) erweitert dieses Modell der Agenteneigenschaften um weitere Attribute wie **Rationalität**, **Benevolenz**, **Wahrhaftigkeit**, **Introspektion** und **Mobilität**.

Es bleibt jedoch festzuhalten, dass ein Agent nicht alle oben aufgeführten Eigenschaften erfüllen muß, um als ein solcher anerkannt zu werden; die wichtigsten Eigenschaften umfassen die Reaktivität, Autonomie, Pro-Aktivität und das kooperative Verhalten. Ein Agent muß also nicht zwingend mobil sein, um als Agent angesehen zu werden.

Die Systeme, in denen die Agenten dann miteinander arbeiten und kommunizieren, werden in der Verteilten Künstlichen Intelligenz in zwei Arten unterschieden. Es gibt verteilte Problemlösesysteme (*Distributed Problem Systems* , kurz DPS) und Multiagentensysteme (MAS). Ein DPS wird eigens für ein spezielles Problem entworfen, wobei man zuerst das eigentliche Problem in verschiedene Teilaufgaben zerlegt und für diese dann spezielle Problemlöser kreiert. Die Agentenattribute Koordination, Kooperation sowie die Pro-Aktivität werden dabei explizit vorgegeben und schränken den Agenten somit erheblich ein.

Im Forschungsbereich der Multiagentensysteme sind die einzelnen Problemlösereinheiten autonomer. Eine zentrale Kontrolleinheit, die die Berechnungen der anderen Einheiten gewissermaßen überwacht, gibt es nicht. Die Agenten sind generischer Art und können daher auch flexibler arbeiten als es bei DPS der Fall ist. In MAS arbeiten die Agenten also vereint mit ihrem Wissen, Fähigkeiten und Plänen, um ihre gemeinsamen Ziele zu erreichen [Bond and Gasser, 1988].

2.1.2 INTERRAP-Architektur

Nachdem wir im vorangegangenen Abschnitt den Begriff des *Agenten* definiert haben mit dem Ergebnis, dass ein Agent eine eigenständige Programmeinheit ist, die Daten empfangen und versenden kann, auf diesen Daten Planungen durchführen und die entstehenden Pläne dann auch realisieren kann, wollen wir hier auf die Struktur eines Agenten eingehen.

Prinzipiell unterscheidet man zwischen *reaktiven* und *deliberativen* Agenten. Die reaktiven Agenten treffen ihre Entscheidungen verhaltensbasiert, wogegen die deliberativen über Wissen verfügen, welches sie in Form von Plänen in ihren Aktionen umsetzen. Die wohl bekannteste deliberative Agentenarchitektur ist die BDI-Architektur,

welche von Bratman, Israel und Pollack entwickelt wurde. BDI steht hierbei für die Begriffe *beliefs*, also Fakten, die der Agent glaubt und auf welche er seine Handlungen basiert, *desires*, das sind Wünsche, die noch nicht unbedingt konkretisiert werden oder gar in Konflikt zueinander stehen können und *intentions*, welche die konkreten Absichten beschreibt. Ein sehr bekanntes Beispiel für eine BDI-Architektur ist das *Procedural Reasoning System* (PRS) [Georgeff and Ingrand, 1990], welches für einen mobilen Roboter entwickelt wurde, der von einem hierarchischen Planer gesteuert wird.

Der wohl schwerwiegendste Nachteil dieser BDI-Architekturen ist deren mangelnde Fähigkeit zur Reaktivität. Einer in dieser Architektur entwickelter Agent ist nicht in der Lage, während eines laufenden *updates* zu reagieren.

Gerade aus diesem Nachteil motiviert wurde in den neunziger Jahren am DFKI die INTERRAP-Architektur entwickelt. Diese Architektur wurde für komplexe, dynamische Agentengesellschaften von autonomen und kooperativen Agenten unter anderem speziell für kooperative Planungsanwendungen entworfen [Fischer et al., 1994]. Ziel der Entwicklung war, Reaktivität und Deliberation in einer Agentenarchitektur zu integrieren. Abbildung 2.1 zeigt den Aufbau dieser Agentenstruktur. Ein INTERRAP-

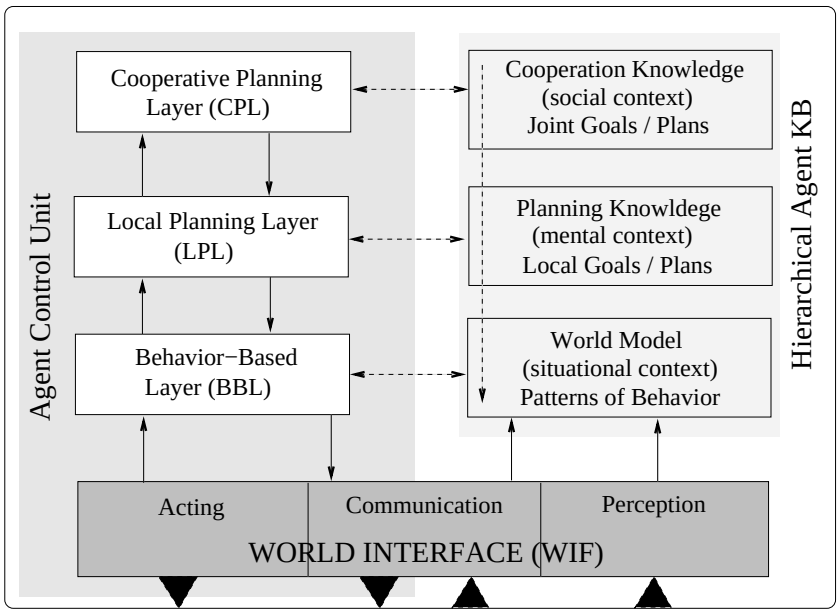

Abbildung 2.1: INTERRAP Agentenstruktur

Agent besteht aus drei Hauptkomponenten, nämlich einer Schnittstelle (*World Interface, WIF*), über die der Agent Kontakt zu anderen Agenten aufnehmen kann, einer hierarchischen Wissensbasis (*Hierarchical Agent KB*) und einer Kontrolleinheit (*Agent Control Unit*). Diese Kontrolleinheit ist wiederum in sich dreifach aufgebaut, sie besteht aus einer kooperativen Planungsebene (*Cooperative Planning Layer, CPL*), die für die Planung verantwortlich ist, die nicht nur den Agenten selbst sondern auch

andere Agenten durch die Interaktion betreffen. Desweiteren besteht sie aus einer lokalen Planungsebene (*Local Planning Layer, LPL*), in der dann nur die den Agenten selbst betreffenden Aufgaben bearbeitet werden; schließlich wird die Kontrolleinheit durch die verhaltensbasierte Ebene (*Behaviour-Based Layer, BBL*) komplettiert. Diese BBL ermöglicht das reaktive Verhalten auf besondere Situationen durch das sogenannte PoB (*reactor patterns of behaviour*), die anderen beiden Ebenen sind deliberativ.

2.1.3 FIPA-Standard

Die *Foundation for Intelligent Physical Agents*, kurz FIPA, ist eine Organisation, die Standards für Softwareagenten entwickelt. Mehrere Organisationen und Firmen sind Mitglied der in Genf ansässigen Organisation, deren Ziel heterogene Agentensysteme sind. Der FIPA-Standard ist Grundlage für etliche Multiagentensysteme, die durch die Umsetzung der FIPA-entwickelten Spezifikationen plattformübergreifend miteinander kommunizieren und arbeiten können. Die Weiterentwicklung der Standards erfolgt durch ständigen Austausch der partizipierenden Anwender und den Entwicklern über eine Mailingliste, aber auch auf regelmässig stattfindenden Konferenzen und Workshops. Sämtliche Spezifikationen sind per Internet zugänglich und hierarchisch wie in Abbildung 2.1 geordnet, nach der sich auch die Gliederung dieses Abschnitts richtet, in dem alle in der Abbildung aufgeführten Aspekte kurz erläutert werden.

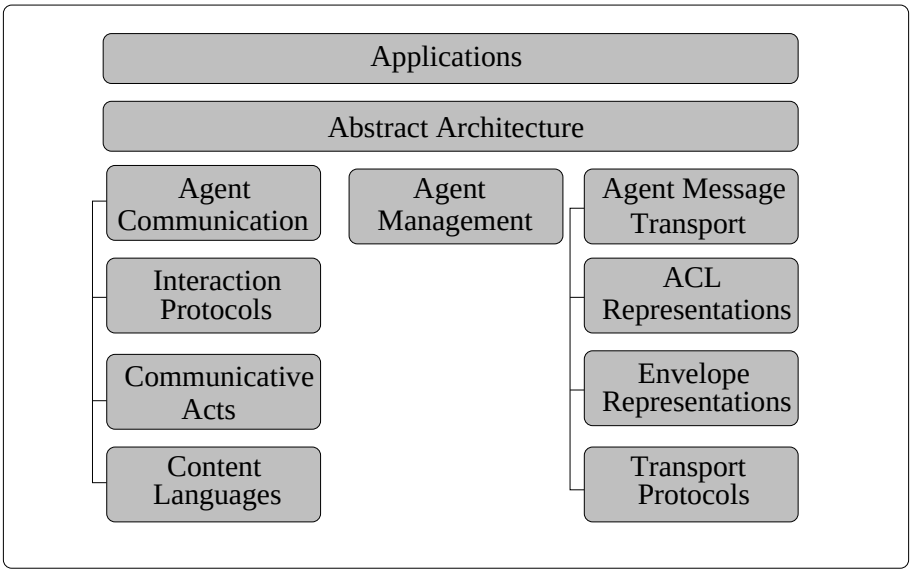

Abbildung 2.2: Spezifikationsgliederung in FIPA

Generell werden hier die externen Eigenschaften und Verhalten, vornehmlich die Interaktion und Kommunikation zwischen Agenten beschrieben. FIPA beschreibt nicht die internen Arbeitsweise eines Agenten oder analysiert ebenso nicht die Informationsinhalte, die sie über Nachrichten erhalten. In diesem Abschnitt wird also näher auf die

Kommunikation, also die Dialoge, Sprechakte oder Ontologien eingegangen. Desweiteren werden die Regeln eben für diese Kommunikation standardisiert, die Architektur und Organisation von Agenten besprochen sowie unterstützende Elemente wie Transportkodierungen erläutert.

Anwendungen ("Applications")

Auf der höchsten Ebene dieser Spezifikationen werden die Anwendungsgebiete von FIPA erläutert. Es werden also die Umgebungen beschrieben, in denen FIPA-Agenten eingesetzt werden können, wobei die Ontologien der jeweiligen Umgebungen näher beschrieben werden.

Abstrakte Architektur

Auf dieser Ebene der Spezifikation werden die Bedingungen für die Agentenplattform und ihre Dienste abgesteckt [FIPA, 2001a]. Die so entstehende abstrakte Architektur sieht in ihren konkreten Realisierungen immer ähnlich aus, und sie besteht aus drei unabdingbaren Elementen, nämlich der Nachrichtenübertragung, einer Suchfunktion für spezielle Dienste sowie einer gemeinsamen Sprache, der *Agent Communication Language*, kurz ACL. Abbildung 2.3 zeigt diese Architektur, wobei eine konkrete Realisierung zum Beispiel aus Java-Elementen bestehen könnte.

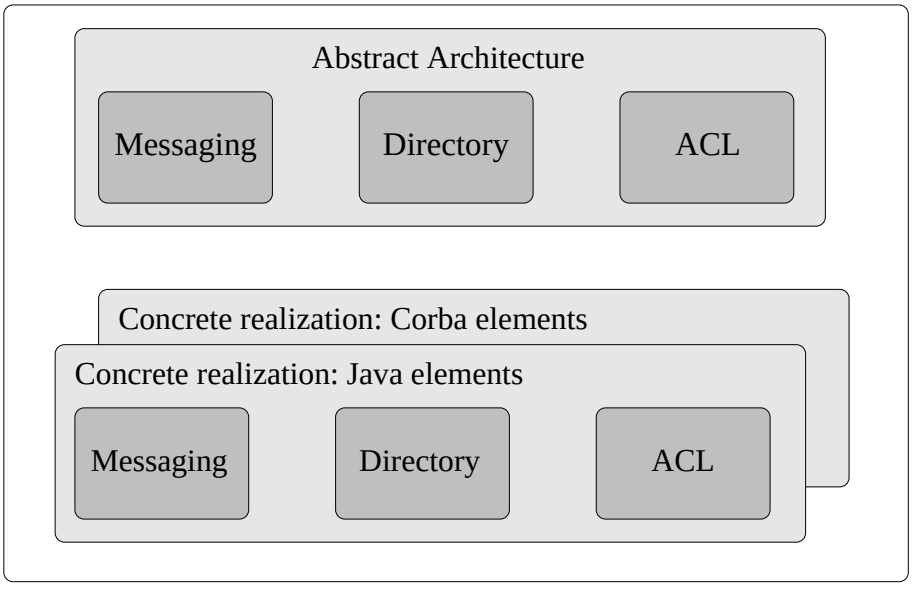

Abbildung 2.3: Abstrakte Architektur

In diesem Bereich der Spezifikation wird auch die Verpackung von Nachrichten, welche zwischen den Agenten versandt werden, beschrieben. Abbildung 2.4 soll den Vorgang veranschaulichen, der aber auch noch in dem Teilabschnitt *Nachrichtenübertragung* näher beschrieben wird. Die Nachricht wird dabei über das *Message-encoding*

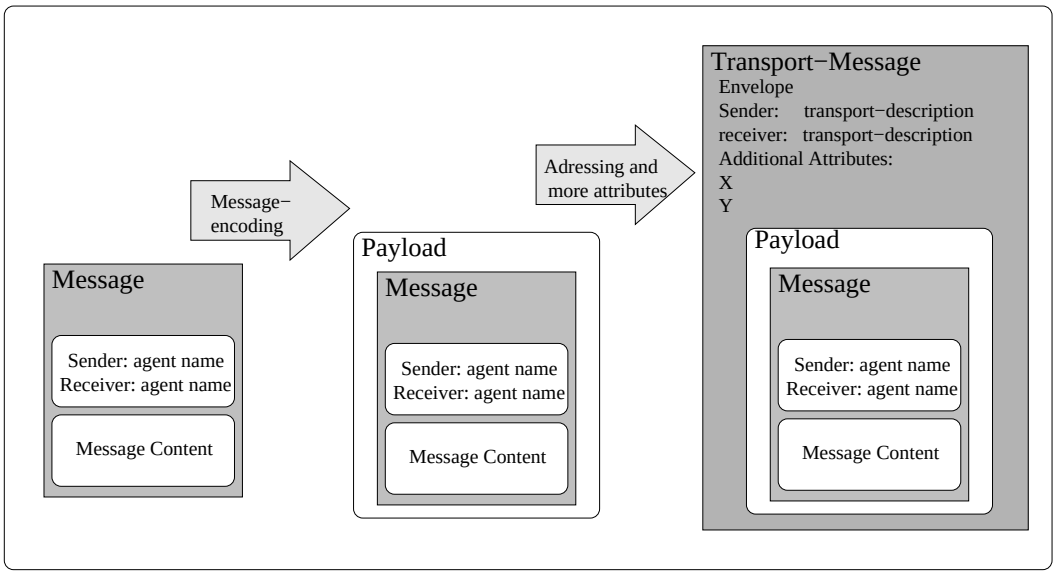

Abbildung 2.4: Nachricht wird verpackt und versandtfähig gemacht [FIPA, 2001a]

gewissermaßen fertiggestellt, ehe sie danach versandfertig gemacht wird, also verpackt und adressiert wird. Der Vorgang kann mit dem Versenden eines Postpaketes verglichen werden, welches erst sicher verpackt und dann erst adressiert und verschickt wird.

Der Grund, warum wir hier von einer abstrakten und nicht direkt von einer konkreten Architektur sprechen, liegt darin, dass wir zuerst die Beziehungen der einzelnen Elemente eines Agenten allgemein beschreiben wollen, ohne dass wir auf implementiertechnische Details, die vom Kern des Themas ablenken würden, eingehen müssen.

Kommunikation (“Agent Communication”)

Auf dieser und allen direkt darunter liegenden Ebenen der Spezifikation wird die Kommunikation genauer beschrieben. Die Kommunikation zwischen den Agenten ist ein sehr wichtiger Aspekt und muß unter allen Umständen korrekt laufen, damit eine Agentenplattform überhaupt Sinn macht, denn über sie wird der gesamte Wissens- und Informationsaustausch abgewickelt.

In FIPA werden die Dialoge durch feste Abfolgen von *Sprechakten*, welche wir noch im Verlauf dieses Abschnitts erklären werden, aufgebaut. Die Dialoge folgen also festen *Protokollen*, den *”Interaction Protocols”*, in denen eine logische Abfolge von Sprechakttypen schon vorgegeben ist. Den eigentlichen Inhalt der Nachricht versenden die Agenten innerhalb eines solchen Sprechaktes dann in dem *Content*, also dem Inhalt, mit dem der angesprochene Agent eigentlich arbeiten kann. FIPA entwickelte für diese Inhalte eigens die *Content Languages*.

- **Protokolle (”Interaction Protocols”)**
  Wie schon erwähnt, definiert FIPA mehrere Protokolle für die Kommunikation.

Es gibt eine Vielzahl von fundamentalen Protokollen, die für den normalen Informationsaustausch benötigt werden, aber auch speziellere wie das *Contract Net Protocol* (CNP), welches unter anderem für Auktionen in Elektronischen Märkten benutzt wird [Smith, 1979]. Dieses CNP (Abbildung 2.5) wird auch in dieser Arbeit von zentraler Bedeutung sein und wird daher nun etwas näher erläutert. Im CNP gibt es einen Initiator, der das Protokoll startet, indem er an

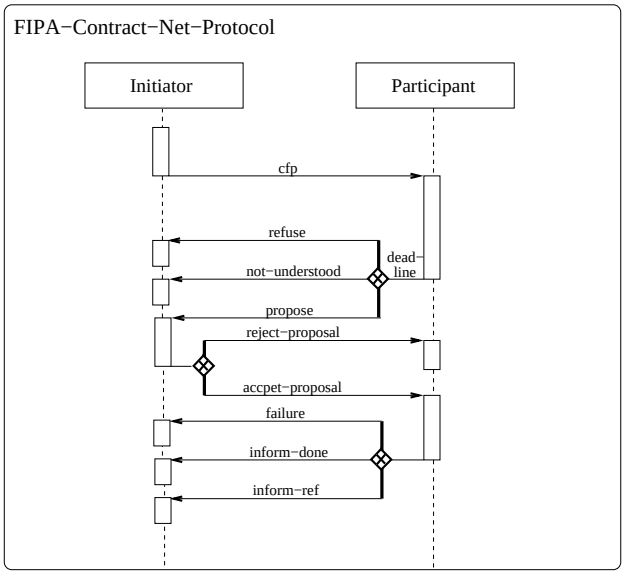

Abbildung 2.5: FIPA Contract Net Interaction Protocol [FIPA, 2001e]

alle beteiligten Agenten ein *call for proposal* (kurz *cfp*) schickt. Diese antworten dann innerhalb einer limitierten Zeit und geben entweder ein Gebot, eine Ablehnung oder eine Fehlermeldung zurück. Bei den letzten beiden Möglichkeiten wird das CNP dann mit diesen Agenten beendet. Der Initiator kann nun entscheiden, ob er das Gebot akzeptiert oder auch nicht, was er dem Agenten im nächsten Schritt des Protokolls mitteilt. Dieser bestätigt diese Meldung nochmals mit wiederum drei Möglichkeiten, womit das Protokoll dann beendet wird. Während einer Auktion kann der Initiator, der also das zu ersteigernde Gut anbietet, aus allen Geboten der an der Auktion beteiligten Bietern/Agenten das für ihn beste Gebot auswählen und dem entsprechenden Agenten den Zuschlag erteilen. Dies hängt jedoch auch von dem vorher gewählten Auktionstyp ab, in dem das Gut ersteigert werden kann.

- **Sprechakte**
  Sämtliche standardisierten Sprechakttypen sind in der Spezifikation ausführlich erläutert [FIPA, 2001d]. Ein Sprechakt ist ist das Ausführen einer Handlung durch eine sprachliche Äußerung [Searle, 1969]. Jeder Sprechakt enthält eine Menge von fest vorgegebenen Informationen wie den Typ des Aktes, den Inhalt

(Content) sowie mehrere Aspekte über die Konversationskontrolle und die Teilnehmer. Zudem werden mögliche Folgen von Sprechakten vorgegeben, damit es nicht zu einer unlogischen Chronologie kommen kann.

- **"Content Languages"**
  Diese Sprachen dienen der Übermittlung des Inhalts der Nachrichten, wobei FIPA mehrere zur Auswahl bietet, die unter anderem auf *Semantic Language*, *Ressource Description Framework* und *Knowledge Interchange Format* basieren.

Agentenverwaltung ("Agent Management")

Auf der Ebene der Agentenverwaltung befinden sich die Spezifikationen rund um den Agenten. Dieser ist hierbei die einzige Komponente, die vom Entwickler durch eigene Software beeinflußt werden kann. In den Spezifikationen werden alle Komponenten, die sich auf einer Plattform befinden, beschrieben [FIPA, 2001b]. Diese sind zum einen das *Agent Management System*, kurz AMS, der *Directory Facilitator* (DF), das *Message Transport System*, welches jedoch erst im nächsten Abschnitt näher beschrieben wird, und schließlich natürlich die Agenten mit ihren vom Entwickler bestimmten Funktionalitäten. Das *Agent Management System* (kurz AMS) verwaltet die auf der Plattform befindlichen Agenten, der *Directory Facilitator* (DF) vermittelt die Agenten untereinander. Er übernimmt also in gewisser Weise die Aufgabe eines Branchenführers, der bei konkreten Anfragen eines Agenten nach anderen Agenten mit bestimmten Eigenschaften die Adressen dieser ihm liefert. Sowohl DF als auch AMS sind ihrerseits wiederum Agenten, bei denen sich jeder auf der Plattform agierende Softwareagent anmelden muß, beim erstgenannten insbesondere mit seinen Eigenschaften, damit dieser ihn an andere vermitteln kann. Abbildung 2.6 soll die eben beschriebenen

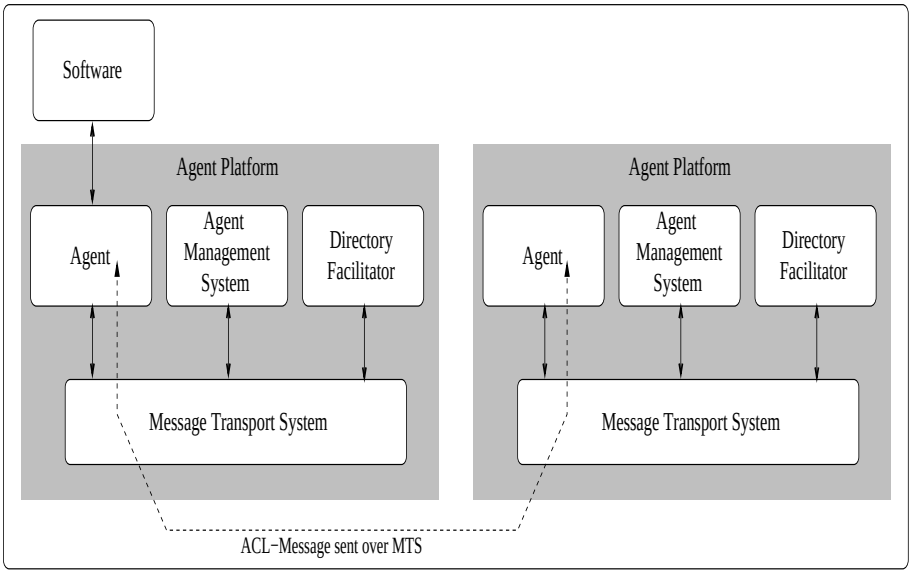

Abbildung 2.6: Agentenverwaltung & Nachrichtentransport

Komponenten, die zusammen das *Agent Management Reference Model* bilden, nochmals graphisch veranschaulichen. Darüberhinaus zeigt die Abbildung die plattformübergreifende Nachrichtenübertragung zwischen zwei Agenten, welche im folgenden Abschnitt näher erklärt wird.

Nachrichtenübertragung

Das *Message Transport System* oder auch *Message Transport Service*, kurz MTS, welches im vorangegangenen Abschnitt schon einmal erwähnt wurde und in Abbildung 2.6 auch abgebildet ist, zeichnet sich für den plattformübergreifenden Transport und Kodierung der Nachrichten verantwortlich. Hierbei müssen sowohl von der Plattform ein- als auch ausgehende Nachrichten über das MTS laufen, damit eine korrekte Übertragung gewährleistet werden kann. FIPA spezifiziert in dieser Ebene und allen darunterliegenden die Nachrichtenübertragung zwischen den Agenten, also die Funktionalität des MTS [FIPA, 2001c].

- **Kodierung der Nachricht (“ACL Representation”)**
  Nachrichten können auf drei verschiedene Arten kodiert werden. FIPA bietet eine effiziente Kodierung auf Bit-Ebene an, eine im XML-Format (*eXtensible Markup Language*) und eine Kodierung als Strings in geklammerten Ausdrücken.

- **Kodierung der Hülle**
  Wie die Nachricht selbst, so muß auch gewissermaßen der Umschlag, also die Verpackung der Nachricht, kodiert werden, damit sie korrekt versendet werden kann. Hier unterstützt FIPA zwei Möglichkeiten, zum einen die Kodierung im XML-Format und zum anderen die Kodierung auf Bit-Ebene.

- **Übertragungsprotokolle**
  In diesem Bereich der Spezifikation werden die Übertragungsprotokolle genannt, die zur Zeit unterstützt werden. Das am weitesten verbreitete ist wohl das *Hypertext Transfer Protocol* (HTTP). Auch wird das *Internet Inter-Orb Protocol* (IIOP) unterstützt, welches bevorzugt in Verbindung mit *Remote Method Invocation* (RMI) verwendet wird. Schließlich wird noch das *Wireless Application Protocol* (WAP), bekannt durch die Verbindung zwischen Mobiltelefonen und reduzierter Internetnutzung, unterstützt. Das WAP wird zum Beispiel in dem CASA-Projekt am DFKI unter anderem im Rahmen von Auktionen benutzt [Gerber and Klusch, 2001].

2.2 Constraint Solving: Ein Überblick

Das *Constraint Programming* (CP) ist eine sich ständig weiterentwickelnde Software-Technologie zur deklarativen Beschreibung und effektiven Lösung von großen, teilweise kombinatorischen Problemen speziell aus dem Bereich des Planens und des Scheduling. In der KI werden *Constraint-Netzwerke* und *Constraint Satisfaction Problems* (CSP) seit den siebziger Jahren studiert.

Ein *Constraint* ist eine Relation zwischen mehreren Unbekannten oder Variablen, von der jede Werte aus einem gegebenen Wertebereich annehmen kann. Ein *Constraint* schränkt also die möglichen Werte, die eine Variable annehmen kann, durch seine Relation ein. "Der Punkt liegt innerhalb des Quadrats" relationiert also die beiden Objekte, die durch ihre Koordinaten spezifiziert werden, wobei die Lage des einen Objektes stark eingeschränkt ist, wenn die Koordinaten des anderen Objektes fest vorgegeben sind.

Das CP wird heute sehr erfolgreich in vielen verschiedenen Bereichen der Informatik angewendet, aber auch in etlichen anderen Forschungsbereichen. In Datenbanksystemen verwendet man es zum Beispiel zur Konsistenzsicherung der Daten oder in Programmiersprachen zur Konstruktion von effizienten Parsern. Auch in der Molekularbiologie spielt das *Constraint Programming* etwa bei der DNA-Sequenzierung eine wichtige Rolle, oder in der Unternehmensforschung werden ganze Optimierungsprobleme durch das CP gelöst. Im folgenden soll hierfür aber erst die Frage beantwortet werden, was ein Constraint überhaupt ist.

Definition 2 (Constraint nach Wilson & Borning, 1993)
Ein **Constraint** *ist eine Relation über einem Universum* **D**. *Das Universum* **D** *entscheidet das Constraint-Prädikaten-Symbol* $\prod_{\mathbf{D}}$ *der Sprache, welche die Relation "=" enthalten muß. Ein Constraint ist dann ein Ausdruck der Form* $\mathrm{p}(\mathrm{t}_1, ..., \mathrm{t}_\mathrm{n})$, *wobei* p *ein n-stelliges Symbol in* $\prod_{\mathbf{D}}$ *ist und jedes* t_i *ein Term ist* [Wilson and Borning, 1993].

Constraints haben viele interessante und nützliche Eigenschaften. Sie spezifizieren nur *partielle Informationen* über ihre Variablen, d. h. sie geben den Wert nicht explizit vor. Desweiteren sind sie *nicht gerichtet*, wie etwa in dem Beispiel mit dem Punkt und dem Quadrat, sie können die Relation also in verschiedenen Richtungen beeinflussen. Sie sind *deklarativ*, sie spezifizieren also ein Verhältnis, nicht aber die exakte Berechnung. Weitere Eigenschaften bilden die *Additivität* und die *vereinzelte Unabhängigkeit*.

Nun wollen wir, nachdem wir schon einige Kernbegriffe aus diesem Forschungsbereich verwendet haben, klären, worum es im *Constraint Programming* überhaupt geht. CP ist die Studie von Berechnungssystemen, die auf Constraints basieren. Die Idee ist, Probleme zu lösen, die man durch Constraints formulieren kann, wobei die Lösungen dann diese Einschränkungen einhalten müssen. Man sagt auch, dass die Lösungen die Constraints erfüllen müssen, daher wird ein solches Problem oft auch als *Constraint Satisfaction Problem* (CSP) bezeichnet.

Definition 3 (CSP nach Hannebauer, 2000)
Ein **Constraint Satisfaction Problem (CSP)** *ist spezifiziert durch ein Tripel* $\prod_{cs} = (X, D, C)$.

- $\mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}\}$ *ist eine Menge von Variablen, wobei jedes* $\mathrm{x_i}$ *Werte aus einem endlichen Universum* $\mathrm{D_i}$ *der Menge* $\mathrm{D} = \{\mathrm{D}_1, \ldots, \mathrm{D_n}\}$ *annehmen kann.*
- *Die Abbildung* $\lambda : X \rightarrow D_1 \cup \cdots \cup D_n, \lambda = (v_1, \ldots, v_n) \in D_1 \times \cdots \times D_n$ *weist jeder Variable* x_i *einen Wert* $v_i \in D_i$ *zu.*
  Die partielle Abbildung
  $\lambda^{'} : (\{x_{\lambda_1'}, \ldots, x_{\lambda_l'}\} = X^{'} \subseteq X) \rightarrow (D_{x_{\lambda_1'}} \cup \cdots \cup D_{x_{\lambda_l'}} = D^{'})$
  weist jeder Variable $x_i \in X^{'}$ *einen Wert* $v_i \in D_i$ *zu.*
- $\mathrm{C} = \{\mathrm{C}_1, \ldots, \mathrm{C_m}\}$ *ist eine Menge von Constraints mit* $C_i \in D_{i_1} \times \cdots \times D_{i_k}$, *wobei jedes* C_i *eine Relation des Typs* $2^{D_{i_1} \times \cdots \times D_{i_k}}$ *ist, welche die Menge der endlichen (partiellen) Abbildungen einschränkt.*

Mit dieser Spezifikation besteht das Problem des **CSP** *darin, eine Abbildung* $\lambda \in \Lambda(C)$ *zu finden, wobei* Λ *der Lösungsraum wie folgt definiert ist:*
$\Lambda(C) = \{(v_1, \ldots, v_n) \in D_1 \times \cdots \times D_n | \, \forall C_i \in C : (v_{i_1}, \ldots, v_{i_k}) \in C_i\}$.
Ein CSP wird **binär (**$\prod_{bcs}$**)** *bezeichnet genau dann, wenn alle Constraints* $C_i \in C$ *binäre Relationen sind.*

Es bleibt anzumerken, dass man einen Constraint als binär bezeichnet im Gegensatz zu unär, wenn man seine Relation in Form eines Graphen darstellen kann, auf dem dann verschiedene Graphenalgorithmen als Grundlage zur Lösungsfindung eingesetzt werden können. In [Hannebauer, 2000] oder [Barták, 1998] wird auf diese Thematik bezüglich unär-binär und *Constraintgraphen* näher eingegangen, was wir hier aber nicht weiter vertiefen wollen. Bei einem CSP sucht man also nach einer Variablenbelegung, die alle Constraints erfüllt.

An dieser Stelle wollen wir auch den Unterschied zwischen *Constraint Satisfaction* und *Constraint Solving* deutlich machen. Bei erstgenanntem sind wie aus der Definition klar hervorgeht die Wertebereiche der Variablen endlich, beim *Constraint Solving* dagegen können manche oder alle Variablen Werte aus unendlichen Universen annehmen, desweiteren können die Constraints auch etwas komplizierterer Art sein, etwa nicht-linear. Folglich basieren die Algorithmen für das *Constraint Solving* auch auf algebraischen und numerischen Aspekten im Gegensatz zum Suchen und Kombinieren bei *Constraint Satisfaction*-Algorithmen.

Ein CSP reicht in vielen Anwendungsszenarien jedoch nicht aus, denn meist ist man nicht an irgendeiner Lösung interessiert, sondern man möchte die optimale Lösung für sein Problem finden. Hierfür kann man ein CSP einfach zu einem *Constraint Satisfaction Optimisation Problem* (CSOP) erweitern.

Definition 4 (CSOP nach Barták,1998)
Ein **Constraint Satisfaction Optimisation Problem (CSOP)** *besteht aus einem CSP und einer Optimierungsfunktion* (objective function)*, welche die gefundenen Lösungen auf numerische Werte abbildet, die dann verglichen und optimiert werden können.*

Hier stellt sich die Frage, warum man diese Techniken im Gegensatz zu den schon länger existierenden und bewährten Ansätzen aus der Mathematik verwendet, die im folgenden beantwortet werden soll.

Es gibt zwei gewichtige Gründe hierfür, zum einen ist die Repräsentation eines Problems als CSP meist näher an dem eigentlichen Problem, das CSP korrespondiert also direkt zu den Problementitäten, und zum anderen finden CSP-Algorithmen, trotz ihres meist einfachen Aufbaus, oft schneller Lösungen als dies der Fall ist, wenn man *Integer-Programming-Methoden* verwendet. Desweiteren muß man lediglich die Lösung beschreiben durch die Constraints, nicht aber den Weg dorthin.

Das Finden von Lösungen für CSP´s oder CSOP´s ist trotzdem nicht trivial. Normalerweise sind die Probleme, die in CSP´s gestellt werden **NP-hart**, das Finden einer Lösung ist also nicht mehr in polynomieller Zeit entscheidbar, da es exponentiell viele Lösungen gibt, und sogar für eine gefundene Lösung ist es nicht mehr in dieser Zeit möglich zu zeigen, dass diese Lösung auch korrekt ist. Der entstehende Suchraum hat die Größe $\prod_{i=1}^{n} * |D_{x_i}|$, gültige Lösungen belegen aber oft nur einen kleinen Teilbereich dieses Suchraumes, so dass es effiziente Verfahren zur Problemreduktion geben sollte, die wir im folgenden Abschnitt beschreiben werden.

2.2.1 Lösungsansätze

Aus dem vorangegangenen Abschnitt geht hervor, dass das Lösen von CSP oft ein NP-hartes Problem ist. Es gibt verschiedene Ansätze, trotzdem möglichst effizient solche Probleminstanzen zu entscheiden, von denen wir hier nun einige kurz erörtern wollen.

Einen möglichen Ansatz bilden die *Systematic Search Algorithms*, die in systematischen Vorgehensweisen verschiedene Lösungen und erzeugen und diese dann überprüfen. *Generate-and-Test* **(GT)** heißt auch das erste Paradigma, welches wir betrachten wollen. Ein GT-Algorithmus generiert jede mögliche Lösung und prüft dann, ob sie alle Constraints erfüllt. Eine andere effizientere und bekanntere Methode bietet das *Backtracking* **(BT)**. BT ist der am weitesten verbreitete Algorithmus bezüglich systematischer Suche. Hierbei wird inkrementell eine partielle Lösung zu einer kompletten ergänzt, wobei man wiederholt den Wert einer bestimmten Variable ändert. Dechter und Frost (1999) sowie Ginsberg und Harvey (1999) haben diese Paradigmen ausführlich beschrieben [Dechter and Frost, 1998], [William D. Harvey, 1995]. Sowohl GT als auch BT haben jedoch einen großen Nachteil, beide erzeugen im Verlauf zu viele falsche Lösungen, welche sie direkt wieder verwerfen müssen, darüberhinaus *lernen* sie nicht aus diesen Fehlern, denn sie machen sie im Verlauf ihrer Berechnungen immer wieder. Auch erkennen beide entstehende Konflikte bei der Zusammensetzung viel zu spät, was sich auf ihre schlechten Laufzeiten auswirkt.

Basierend auf *Constraint-Graphen* existieren verschiedene Konsistenztechniken, die den Suchraum einschränken, um effizienter zu werden. Der gemeinsame Ansatz sämtlicher Algorithmen basiert darauf, eine partielle Lösung, also ein Teilnetzwerk so zu erweitern, dass man in diesem Inkonsistenzen sehr schnell erkennen kann. Die Techniken gehen von der einfachen *node-consistency* und der sehr bekannten *arc-consistency* bis hin zur vollen, aber sehr "teuren" *path-consistency*, welche alle in [Kumar, 1992], [Mackworth, 1977] oder [Mackworth and Freuder, 1985] näher beschrieben sind. Die Konsistenztechniken sind kurz wie folgt umschrieben:

- *Knotenkonsistenz (node-consistency):* Die Domäne jeder Variable ist bezüglich jedes unären Constraints konsistent.
- *Kantenkonsistenz (arc-consistency):* Für alle Variablenpaare gilt, dass ihre Domänen bezüglich aller unären sowie binären Constraints konsistent sind.
- *Pfadkonsistenz (path-consistency):* Für alle Variablentripel gilt, dass ihre Domänen bezüglich aller unären sowie binären Constraints konsistent sind.
- *k-Konsistenz (k-consistency):* Für alle k-Tupel von Variablen gilt, dass ihre Domänen bezüglich aller unären sowie binären Constraints konsistent sind.

Ein dritter, auch sehr populärer Ansatz ist das sogenannte *Constraint Propagation*. Es wird innerhalb des einfachen *Backtracking*-Algorithmus angewendet und benutzt dort dann eine Konsistenztechnik wie eben beschrieben. Da Backtracking an sich schon eine Art Konsistenztechnik beinhaltet, kann man es als eine Kombination von GT und eine Variante der *arc-consistency* betrachten. Das *forward checking* ist eine andere Möglichkeit, künftige Konflikte zu entdecken und zu vermeiden. Es führt eine eingeschränkte *arc consistency* durch, wobei sich das "eingeschränkt" auf die Tatsache bezieht, dass lediglich die Constraints bezüglich der momentanen Variable und den künftigen Variablen geprüft werden. Der Vorteil liegt darin, dass man so inkonsistente Lösungen frühzeitig abfangen kann und den Suchbaum so erheblich reduziert. *Look ahead* führt diesen Ansatz fort, wobei jedoch eine komplette Zweigkonsistenz vollzogen wird. Somit können ganze *Zweige* des Suchbaums frühzeitig abgeschnitten werden. Etliche Systeme benutzen diese Kombination wie auch zum Beispiel Mozart, das frühere Oz, welches am DFKI entwickelt wurde [Smolka, 1998], [Henz et al., 1993].

Auch gibt es Heuristiken und stochastische Algorithmen, die recht schnell zu guten Lösungen kommen, die jedoch keine optimale Lösung garantieren können. Bekannte Methoden wie das *Hill Climbing* werden in diesen Systemen angewendet.

2.2.2 CSP und HCSP

In diesem Teilabschnitt werden wir das eingangs vorgestellte CSP und CSOP noch erweitern. In vielen Anwendungen existieren nämlich keine Lösungen, die alle Constraints erfüllen, in anderen Situationen jedoch sogar mehrere. Systeme, in denen es keine Lösung gibt, die alle Constraints erfüllt, nennt man *over-constrained systems*, solche, in denen es mehrere gibt, nennt man *under-constrained systems*.

Ein sehr populärer Ansatz, dieser Problematik Herr zu werden, ist das Erweitern des CSP zu einem *Hierarchical Constraint Satisfaction Problem* (HCSP). Hierbei werden die Constraints mit Gewichtungen behaftet, die ihrer Wichtigkeit entsprechen. Die dabei entstehenden Constraint-Hierarchien bestehen dann aus sogenannten *harten* und *weichen* (engl.: *soft*) Constraints. Die harten (engl.: *required*) Constraints müssen eingehalten werden, wogegen die weichen Constraints bestmöglich eingehalten werden sollten.

Hierfür müssen die eingangs des Kapitels aufgeführten Definitionen ebenso erweitert werden. Ein Constraint ist ein **gewichteter Constraint**, wenn er zu den Eigenschaften, die ein normaler Constraint erfüllen muß zudem auch noch mit einer *Präferenz* beziehungsweise *Stärke*, welche die Wichtigkeit des Constraints zum Ausdruck bringen soll, versehen werden kann. Eine *Constraint-Hierarchie* kann dann wie folgt definiert werden.

Analog zu der Definition des CSP und des CSOP könnte man auch folgende Definition noch erweitern. Auch hier ist es möglich, die Lösung für die Constraint-Hierarchie durch eine Funktion auf einen numerischen Wert abbilden zu lassen, diese Funktion könnte man dann als Zielfunktion bezeichnen.

Definition 5 (HCSP nach Wilson & Borning,1989)
Ein **Hiearchical Constraint Satisfaction Problem (HCSP)** *ist ein Tripel (X,D,C) definiert durch:*

- $\mathrm{X} = \{\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{n}\}$ *ist eine Menge von Variablen, wobei jedes* x_i *Werte aus einem endlichen Universum* D_i *der Menge* $\mathrm{D} = \{\mathrm{D}_1, \ldots, \mathrm{D}_\mathrm{n}\}$ *annehmen kann.*

- *C ist ein Constraint-System, wobei C partitioniert ist in Teilmengen* $C_0, C_1, \ldots, C_m$*, wobei* C_i *die Constraints mit Stärke i enthält.* C_0 *enthält die harten Constraints,* $C_1 \ldots C_m$ *die weichen in abnehmender Reihenfolge,* C_1 *also die wichtigsten.*

Eine **Constraint-Hierarchie** *ist also eine endliche Menge H bestehend aus Teilmengen* $H_0, H_1, \ldots, H_m$*, die analog C die entsprechenden Constraints enthalten* [Alan Borning and Wilson, 1989].

Analog kann man nun eine Lösung S für ein HCSP definieren, aber die Frage, die sich einem direkt stellt, ist, wie man nun solche Lösungen miteinander vergleichen kann. Hierfür haben Wilson und Borning (1989) aber auch andere Wissenschaftler der KI eigene **Komparatoren** für Constraint-Hierarchien definiert. Ein Komparator

ist eine irreflexive, transitive Relation über den Variablenbelegungen, die den hierarchischen Aspekt einhalten. Es bleibt dem Leser überlassen, sich tiefergehende Kenntnisse über diese Komparatoren einzuholen, da sie in dieser Arbeit nicht von weiterem Interesse sind.

Es gibt auch andere Ansätze, ein bestehendes CSP zu erweitern, um das Problem der Überbestimmtheit bewältigen zu können. *Fuzzy CSP* (FCSP) erlauben zum Beispiel, dass die normal festen Tupellösungen an manchen Stellen verändert werden, *Probalistic CSP* (Prob-CSP) oder *Weighted CSP* (WCSP) sind andere Ansätze, wobei aber jeder Ansatz für sich schon fast einen kleinen Forschungszweig bildet, und wir sie hier nicht weiter vertiefen wollen, da im Verlauf der Arbeit das HCSP im Vordergrund stehen wird.

2.2.3 Existierende Systeme: Stand der Forschung

Nachdem wir einen groben Überblick über CSP mit seiner speziellen Problematik und seinen Varianten gegeben haben, wollen wir uns hier mit den Algorithmen zur Lösung der eben definierten Klasse HCSP beschäftigen. Constraint-Hierarchien werden von zentraler Bedeutung in dieser Arbeit sein, und wir wollen hier einige Systeme vorstellen, die zum Teil auf den schon vorgestellten Lösungsansatzmöglichkeiten basieren.

Grundsätzlich kann man die existierenden Solver drei verschiedenen Ansatztypen zuordnen. Die **refining method** satisfiziert zuerst die harten Constraints, die nicht gebrochen werden dürfen, und danach sukzessive die weicheren Level. Da diese Methode der eigentlichen Definition eines HCSP direkt folgt, ist sie auf alle Constraint-Hierarchien mit beliebigem Komparator anwendbar. Dieser Ansatz dient Systemen wie *DeltaStar, Simple Algorithm* [Wilson, 1993] oder CHAL [Aiba, 1991] als Grundlage.

Bei Systemen, die **Local Propagation** benutzen, werden die Constraint-Hierarchien gelöst, indem man wiederholt ausgewählte satisfizierbare Constraints hinzufügt. Hierbei wird ein einzelner Constraint zur Bestimmung des Wertes einer Variable benutzt. Nachdem der Wert dann bestimmt ist, wird ein anderer Constraint zur Wertermittlung einer anderen Variable benutzt. Voraussetzung für diese Methode ist allerdings, dass die Constraints von einer bestimmten Form sind, beziehungsweise die Objekte des Constraints von bestimmter Form sind. So muß das Objekt mehrere Funktionen aufweisen, deren Argumente Variablen sind, welche von anderen Variablen bestimmt werden. Unter diesen Funktionen wählt *Local Propagation* dann eine passende heraus, so beinhaltet zum Beispiel die Relation $A + B = C$ die Funktionen $C \leftarrow A + B, A \leftarrow C - B$ und $B \leftarrow C - A$. Diese Methode zählt zu den populärsten und wird somit auch in etlichen Systemen verwendet, zum Beispiel *DeltaBlue* [Sannella et al., 1993], *SkyBlue* [Sannella, 1994], *DETAIL* [Hiroshi Hosobe, 1996], *Houria III* [Neveu, 1996] oder *Indigo* [Borning et al., 1996]. Die dritte Möglichkeit, solche Probleme zu lösen, bietet der **Optimierungsansatz**. Hierbei wird die Constraint-Hierarchie in ein Optimierungsproblem transferiert, wobei die Gewichtungen zu Koeffizienten bestimmter Größe werden. *Cassowary* [Badros and Borning, 1998] oder *HiRise* [Hosobe,] sind Systeme, die diesen Ansatz verfolgen, bei dem im Hintergrund

zur Optimierung ein weiteres Verfahren angewendet wird. Bei Cassowary ist dies zum Beispiel der Simplexalgorithmus, der in Abschnitt 2.2.4 noch erklärt wird.

Abbildung 2.7 zeigt die Aufzählung der Systeme und der Lösungsansätze, welche sie verfolgen, noch einmal etwas übersichtlicher. Der Optimierungsansatz wird dabei nicht explizit aufgeführt, sondern zusammen mit noch anderen Ansätzen, wie etwa in dem *Incremental Hierarchical Constraint Solver* (IHCS) [Menezes et al., 1993], der von einer Anfangsbelegung der Variablen diese korrespondierend zur Constraint-Hierarchie erweitert.

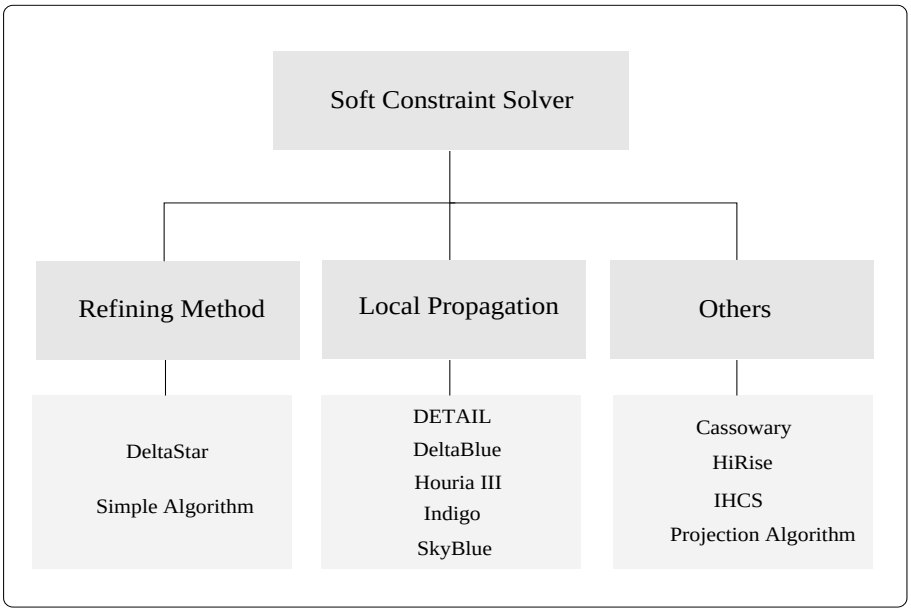

Abbildung 2.7: Existierende Systeme

Von diesen Systemen werden wir nun je eines von jedem Lösungsansatz etwas näher betrachten, um zum einen ausgewählte Algorithmen genauer zu erforschen und zum anderen die verschiedenen Ansätze miteinander besser vergleichen zu können.

Als Vertreter der **Refining-Methode** soll hier *DeltaStar* von Wilson und Borning (1993) betrachtet werden. Dieser Solver erfüllt also zuerst die harten Constraints und versucht danach die Lösung durch Hinzunahme der weichen Constraints zu komplettieren. Um die so entstehenden Lösungen miteinander vergleichen zu können, benutzt *DeltaStar* metrische Komparatoren, mit denen er versucht, Abweichungen bezüglich einer Lösung zu minimieren. Im Hintergrund benutzt *DeltaStar* wie viele andere Solver auch einen sogenannten "flat solver", welcher hier der Simplexalgorithmus ist. Dieser löst das Optimierungsproblem, welches bei dem Minimieren der Abweichungen entsteht. Innerhalb des gesamten Rechenprozesses wird Simplex über eine Prozedur *filter* aufgerufen, die eine Untermenge einer Lösung S liefert, welche die sogenannte *error-function* bezüglich C, der Menge von Constraints, minimiert.

Algorithm 1 Pseudocode für DeltaStar

```
 1: procedure DeltaStar (H: constraint hierarchy)
 2:   i ← 1;
 3:   Solution ← solution of hard constraints from H;
 4:   while(not unique solution and i < number of levels)
 5:    do
 6:      Solution ← filter(Solution, H_i);
 7:      i++;
 8:    od
 9:   return Solution;
10: end DeltaStar;
```

Prinzipiell löst *DeltaStar* (siehe Algorithm 1, Pseudocode für DeltaStar) nur lineare Gleichungen und Ungleichungen, wobei er die Constraint-Hierarchie in Lineare Programme transferiert, welche er mit Hilfe von Simplex dann löst. Die Constraint-Hierarchie, welche dem Solver als Eingabe dient, muß zudem total geordnet sein. Der Solver hat jedoch eine exponentielle Laufzeit in der Anzahl der Variablen, was aus der Benutzung von Simplex im Hintergrund rührt, jedoch hat Simplex bei verhältnismäßig geringer Variablenanzahl noch eine recht gute Laufzeit.

Betrachten wir nun ein System, welches Constraint Hierarchien mittels **Local Propagation** löst. *DeltaBlue* [Sannella et al., 1993] ist ein solcher Solver. Wie schon erwähnt, kann man bei dieser Methode sogenannte *multiway constraints* verwenden, bei denen man zur Bestimmung des Wertes einer Variable von einem Constraint mehrere Methoden anwenden kann. Dieser Aspekt erhöht die Mächtigkeit eines Solvers und er ist zudem auch leichter und effizienter zu implementieren. Der Algorithmus verwendet zum Hinzufügen und Entfernen von Bedingungen/Constraints das Perturbationsmodell, welches im Pseudocode von *DeltaBlue* deutlich wird. Desweiteren ist das System inkrementell, was den entscheidenden Vorteil gegenüber *DeltaStar* zum Beispiel bringt, dass man seine Berechnungen nach Modifikationen der Constraintmenge im laufenden Betrieb nicht wieder von vorne beginnen muß. Desweiteren speichert er seine Zwischenergebnisse in einem *Solution Graph*, welcher schon nach Definition 3 in Abschnitt 2.2 erwähnt wurde. Von zentraler Bedeutung ist das Hinzufügen und Entfernen von Constraints, was wir durch den nachstehenden Pseudocode (Algorithm 2, Pseudocode für DeltaBlue) auch genau erklären wollen.

Algorithm 2 Pseudocode für DeltaBlue

```
 1: procedure AddConstraint (c: constraint)
 2:   c.selectedMethod ← none;
 3:   for all variable v ∈ c.variables
 4:   do
 5:     Add c to v.constraints;
 6:   od
 7:   IncrementalAdd(c);
 8: end AddConstraint
 9:
10: procedure RemoveConstraint (c: constraint)
11:  if (Enforced(c)) then
12:   IncrementalRemove(c);
13:  else
14:    for all variable v ∈ c.variables
15:    do
16:      Remove c from v.constraints;
17:    od
18:  fi
19: end RemoveConstraint
```

Die Laufzeit von *DeltaBlue* beträgt $\mathcal{O}(MN)$, wobei M die maximale Anzahl von Methoden in einem Constraint ist und N der Anzahl von Constraints entspricht. Der Solver hat aber auch einige Nachteile, er kann nämlich keine Zyklen im Graphen verwalten beziehungsweise verarbeiten. Ein Zyklus entspricht hier einem *deadlock* von Bedingungen, solche Verklemmungen treten zum Beispiel auf, wenn Constraint A von Constraint B abhängt, B von Constraint C abhängt, C aber wiederum von A abhängt. Abbildung 2.8 soll so einen potentiellen Zyklus nochmals graphisch darstellen.

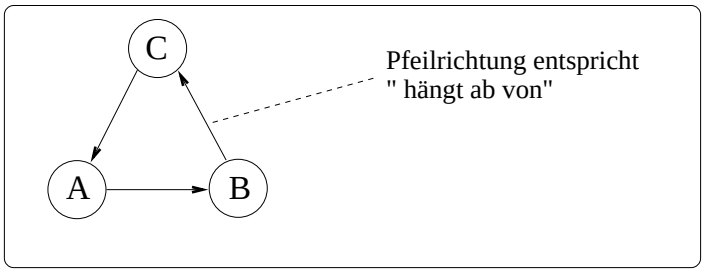

Abbildung 2.8: Zyklus (deadlock) in einem Constraintgraphen

Desweiteren kann *DeltaBlue* nur Methoden mit einer Ausgabevariable bearbeiten. Dies kann die Art der Constraints unter Umständen erheblich einschränken, wenn man etwas komplexere Zusammenhänge mit ihnen zum Ausdruck bringen will.

Zum Abschluß wollen wir noch ein System aus dem **Optimierungsansatz** genauer betrachten. Der hierarchische Constraintsolver *Cassowary* von Badros und Borning (1998) transferiert die Constraint-Hierarchie, welche er als Eingabe erhält, in ein Optimierungsproblem. Dieses löst er dann wie *DeltaStar* mit Hilfe eines "flat solvers", nämlich dem Simplexalgorithmus. Ein Vorteil, den *Cassowary* gegenüber den bisher erwähnten Solvern hat, ist seine Fähigkeit, simultane Gleichungssysteme wirklich effizient zu behandeln, was bei den bisherigen Systemen doch eine erhebliche Einschränkung darstellte. Desweiteren können die Variablen aus unbeschränkten Domänen ihre Werte annehmen, weswegen auch eine besondere Variante des Simplex, nämlich der "augmented simplex" von *Cassowary* verwendet wird. Demnach können also auch Variablen mit negativen Werten mit in Betrachtung gezogen werden, was die Mächtigkeit des Systems erhöht.

Einer der Hauptaspekte bei allen Systemen, die Constraint-Hierarchien lösen können, ist das Hinzufügen und Entfernen von Bedingungen (*Constraints*). Da *Cassowary* für jede Instanz den Simplex benutzt, was zudem eine **optimale Lösung** für diese Instanz garantiert, ist die Veränderung der Hierarchie durch die Sensitivitätsanalysen im Simplexalgorithmus relativ leicht zu behandeln. Das bedeutet, der Solver muß nicht nach einer Veränderung seine Berechnungen von vorne beginnen, sondern er kann die vorangegangene Lösung benutzen, um durch wenige Schritte die nächste Lösung hieraus zu generieren, seine inkrementelle Eigenschaft macht *Cassowary* gegenüber den weiteren Systemen noch interessanter. Die Sensitivitätsanalysen werden im folgenden Abschnitt etwas näher betrachtet.

Cassowary kann zudem auch quadratische Gleichungen betrachten, was jedoch den direkten Einsatz von Simplex im Hintergrund unmöglich macht. Der Solver macht hierbei eine quasi lineare Optimierung durch Abschätzungen der quadratischen Gleichungen. Durch diese Fähigkeit ist *Cassowary* vielen anderen Solvern in seiner Mächtigkeit überlegen, da man so auch komplexere Probleme als HCSP formulieren kann, welche trotzdem noch gelöst werden können.

2.2.4 Simplexalgorithmus

Wie wir im vorangegangenen Abschnitt gesehen haben, basieren *Cassowary* und *DeltaStar* bei der Optimierungsaufgabe auf den Simplexalgorithmus. Dieser von Dantzig bereits 1947 entwickelte, weitläufig bekannte Algorithmus erhält als Eingabe eine Menge von linearen Ungleichungen oder Gleichungen und eine Zielfunktion, ein sogenanntes *Lineares Programm* (LP), welches er optimieren muß [Dantzig, 1963]. Allgemein formuliert sieht ein *Lineares Programm* dann wie folgt aus:

$$\begin{array}{rl} \text{optimize} & a^T \mathrm{x} \\ \text{subject to} & \mathrm{Ax} \quad \Delta \quad \mathrm{d} \end{array}$$

Hierbei ist *optimize* $\in \{min, max\}$ die Optimierungsvorgabe, $\Delta \in \{\leq, =, \geq\}$, $a^T \in \mathbb{R}^n$ der Koeffizientenvektor bezüglich der Variablen *x*, *subject to* gibt die einzuhaltenden Bedingungen an, $d \in \mathbb{R}^m$ der Begrenzungsvektor und $A \in \mathbb{R}^{m \times n}$ eine Koeffizientenmatrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix},$$

in welche dann in Cassowary die Gewichtungen eingetragen werden können.

Eine effiziente Möglichkeit der Implementierung dieses Verfahrens bietet das Simplex-Tableau. Durch sogenannte primale und duale Simplexschritte erreicht der Algorithmus ausgehend von einer zulässigen Basislösung immer neue Basislösungen, die seinen Zielfunktionswert sukzessive verbessern. Durch endlich viele dieser Schritte gelangt man schließlich zu einer optimalen Lösung, und zwar wenn man primal und dual zulässig ist. Man hat dann bestimmt, ob das Problem unbeschränkt ist oder eine optimale Lösung gefunden.

Der Simplexalgorithmus bietet durch seine *Sensitivitätsanalysen* die Möglichkeit, relativ einfach neue Bedingungen hinzuzufügen oder welche zu entfernen. Auch ist es möglich, ganze Entscheidungsvariablen zu streichen oder neue zu instantiieren, ohne dass der Algorithmus seine Berechnungen wieder von vorne beginnen muß. Durch seine inkrementelle Eigenschaft sowie der Möglichkeit, Lösungen auf zwei mögliche Varianten (primal oder dual) berechnen zu können, kann man auch im Verlauf der Berechnungen das eigentliche *Lineare Programm* verändern. Der Vergleich von Lösungen nach Hinzufügen oder Entfernen von Ungleichungen stellt im Simplexalgorithmus somit kein Problem dar. Die genaue Vorgehensweise des Simplexalgorithmus wird an dieser Stelle als bekannt vorausgesetzt [Nemhauser and Wolsey, 1988], ebenso wollen wir hier nicht diese Sensitivitätsanalysen weiter vertiefen.

Nun könnte man die weichen Constraints in einer Constraint-Hierarchie durch diese Methode dem LP hinzufügen, in dem ursprünglich nur die harten Constraints enthalten waren, und die neuen Lösungen miteinander vergleichen anhand von verschiedenen Komparatoren, welche in Abschnitt 2.2.2 beschrieben wurden. Cassowary folgt in etwa diesem Schema, wobei ein direktes Eingreifen bei den Vergleichen nicht möglich ist.

2.3 Scheduling

Planungsprobleme (*scheduling problems*) treten in der realen Welt in fast allen geschäftlichen Prozessen auf. Aber nicht nur aus diesem Grund spielen diese Probleme auch in der Informatik eine gewichtige Rolle. Generell kann man Planungsprobleme als Allokation von Ressourcen über die Zeit ansehen, um eine bestimmte Menge an Aufgaben zu erfüllen. Unter Ressourcen versteht man hierbei die benötigten Mittel oder Fähigkeiten zur Lösung der Aufgaben, welche recht unterschiedlicher Art sein können. Die menschliche Arbeitskraft, Geld, Energie, aber auch die eingesetzten Prozessoren (Maschinen) sowie Werkzeuge sind zum Beispiel konkrete Ressourcen. Auch die Aufgaben können sehr verschieden sein, dies können etwa bestimmte Mengen eines Gutes bis zu einem bestimmten Zeitpunkt sein aber auch benötigter Speicherplatz und Rechengeschwindigkeit in Computersystemen.

2.3.1 Überblick und Grundlagen

Es gibt zahlreiche Algorithmen und Heuristiken, welche gestellte Probleme dieser Art auf verschiedene Arten lösen können. Doch bevor wir verschiedene Algorithmen für dieses NP-harte Problem [J. Blazewicz, 1993] etwas genauer untersuchen, wollen wir das Problem exakt definieren und seine Abwandlungen beschreiben.

Definition 6 (Scheduling-Problem nach Blazewicz et al.,1993)
Ein **Scheduling-Problem** *(Planungsproblem) ist spezifiziert durch ein Tripel* (T, P, R).

- $T = \{T_1, \ldots, T_n\}$ *ist eine Menge von n Aufgaben*
- $P = \{P_1, \ldots, P_m\}$ *ist eine Menge von m Prozessoren (Maschinen)*
- $R = \{R_1, \ldots, R_s\}$ *ist eine Menge von s zusätzlichen Ressourcen*

Beim **Scheduling** *ordnet man Maschinen aus P mit möglichen Ressourcen aus R Aufgaben aus T zu, um alle Aufgaben aus T zu erfüllen unter vorgegebenen Rahmenbedingungen. Ziel der Zuordnung ist eine minimale Auslastung aller Maschinen.*

Die in der Definition erwähnten Rahmenbedingungen beinhalten, dass jede Aufgabe höchstens von einer Maschine bearbeitet werden kann und jede Maschine zu einem Zeitpunkt maximal einen Auftrag bearbeiten kann.

Nun kann man auch noch die Maschinen genauer charakterisieren. Diese können entweder **parallel** sein, was bedeutet, dass alle Maschinen dieselbe Funktionalität haben, oder sie sind **bestimmt**, also spezialisiert auf bestimmte Aufgaben. Die parallelen Maschinen unterscheidet man noch in Abhängigkeit ihrer Geschwindigkeit. Ist diese bei allen gleich, so spricht man von **identischen** Maschinen, ist die Geschwindigkeit der Maschinen unterschiedlich, jedoch haben alle eine Mindestgeschwindigkeit b_i, und die Geschwindigkeit allgemein ist unabhängig von der Aufgabe, so spricht man von

uniformen Maschinen. Wenn schließlich die Geschwindigkeit der Maschinen unterschiedlich ist und sie von der Art der Aufgabe abhängt, so sind die Maschinen **nicht relationiert**.

Im Falle von *bestimmten* Maschinen unterscheidet man drei Modelle der Zuordnung der Aufgaben: **flow shop**, **open shop** und **job shop**. Um diese Modelle präzise beschreiben zu können, muß man die Menge der Aufgaben T spezialisieren. Man geht dabei davon aus, dass jede Aufgabe n Teilmengen bildet (Ketten im Fall von *flow shop* und *job shop*), wobei jede einzelne *Job* genannt wird. Job J_j wird unterteilt in n_j Aufgaben $T_{1j}, \ldots, T_{n_j j}$ sowie zwei weiteren Aufgaben, welche von verschiedenen Maschinen bearbeitet werden müssen.

Beim *flow shop* müssen die Jobs in einer festen Reihenfolge an den Maschinen bearbeitet werden, wobei an jeder Maschine eine spezielle Teilaufgabe eines Jobs bearbeitet wird. Beim *open shop* ist dies ebenso der Fall, mit der Ausnahme, dass die Reihenfolge der Teilaufgaben, die an einer Maschine bezüglich eines Jobs erledigt werden müssen, beliebig ist. Im Falle des *job shop* ist die Anzahl der Teilaufgaben pro Job, ihre Zuteilung zu den einzelnen Maschinen sowie die Reihenfolge ihrer Bearbeitung an den einzelnen Maschinen beziehungsweise Prozessoren beliebig, jedoch a priori bekannt.

Desweiteren unterscheidet man beim Scheduling **deterministische** und **nicht-deterministische** Probleme. Bei den nicht-deterministischen Problemen können die Entscheidungsvariablen etwa mit Wahrscheinlichkeiten behaftet sein. Zudem unterscheidet man **statische** und **dynamische** Systeme, bei den statischen sind im Gegensatz zu den dynamischen Problemen die Mengen der Aufgaben mit all seinen Teilmengen sowie die Menge der Ressourcen zu Beginn bekannt. Bei den dynamischen kommen deswegen häufig *online-Strategien* zum Einsatz. Auch kann man bei Planungsaufgaben die Ressourcen beschränken, was gerade in realen Geschäftsprozessen häufig der Fall ist. Man kann es auch unterteilen in das sogenannte *Single Processor Scheduling* (SMS) und das *Parallel Processor Scheduling* (PMS). Beim SMS steht, wie der Name schon sagt, lediglich eine Maschine zur Verfügung, wobei man dann vom *Serialisieren* anstelle vom Planen spricht. Beim PMS stehen dem Planer entsprechend mehrere Maschinen zur Verfügung. Die Algorithmen für das erstgenannte sind gerade in Situationen von Engpässen auch bei der Verwendung von mehreren Maschinen von Bedeutung.

Sämtlich genannte Variationen und Unterscheidungen des ursprünglichen Problems, welche gerade kurz genannt und erklärt wurden, sollen nicht weiter vertieft werden, sondern einen Überblick über das komplexe Einsatzgebiet von Planungsaufgaben geben. Zudem dient die Unterteilung im Verlauf der Arbeit als Grundlage zur Zuordnung der gestellten Aufgabe zu den einzelnen Spezialfällen.

2.3.2 Algorithmen und Heuristiken

In diesem Teilabschnitt werden wir verschiedene Algorithmen, die eine Planungsaufgabe lösen können, vorstellen. Ein Scheduling-Algorithmus ist ein Algorithmus, der für ein gegebenes Problem einen Plan (engl.: *schedule*) berechnet. Normalerweise würden wir uns nur für die **Optimierungsalgorithmen** interessieren, da nur sie eine optimale Lösung garantieren, jedoch sind diese aufgrund der Komplexität des Problems nur bedingt einsetzbar. Sie würden gegebenenfalls zu lange für ihre Berechnungen brauchen, was ihren Einsatz in realen Prozessen unmöglich macht. Deswegen werden hier auch **Approximierungsalgorithmen** sowie **Heuristiken** im Vordergrund stehen, die zwar keine optimalen Lösungen garantieren können, aber gewisse Schranken einhalten und von der Laufzeit um einiges besser sind als die Optimierungsalgorithmen.

Ein Algorithmus, der eine optimale Lösung garantiert, ist der bereits in Abschnitt 2.2.4 vorgestellte Simplexalgorithmus. Man kann ein Planungsproblem auch als Lineares Programm formulieren, welches der Simplex als Eingabe benötigt. Die Zielfunkion des Linearen Programms könnte zum Beispiel die Endzeitpunkte von Arbeitsschritten in jeder einzelnen Maschine minimieren, und die Nebenbedingungen beinhalten dann die Restriktionen bezüglich der Ressourcen wie etwa der zur Verfügung stehenden Arbeitszeit, Geld oder Material.

Als Beispiel für einen Approximierungsalgorithmus betrachten wir das einfache Planen unabhängiger Aufgaben (SIT). Hierbei sollen *n* Aufgaben mit Längen $w_i \in N$ auf *m* Maschinen verteilt werden, so dass jede einzelne Maschine möglichst wenig arbeitet.

Einfacher Approximationsalgorithmus

- Sortiere die Aufgaben der Länge nach aufsteigend: $w_1 \geq w_2 \geq \ldots \geq w_n$
- Plane eine Aufgabe nach der anderen, wobei w_i der am wenigsten ausgelasteten Maschine zugeteilt wird

Dieser Algorithmus [Seidel, 1996] erzeugt einen Plan der Länge L, wobei L sich im Vergleich zur Länge des optimalen Plans L_{opt} wie folgt verhält:

$$L \leq (\tfrac{4}{3} - \tfrac{1}{3m}) L_{opt}$$

Diese obere Schranke ist eine recht akzeptable Grenze, wenn man hierfür die benötigte Laufzeit für diesen Algorithmus mit der eines Optimierungsalgorithmus vergleicht. Diese setzt sich zusammen aus der Zeit für das Sortieren und dem Verteilen, also m Vergleichen bei jeder Aufgabe.

Beispiel:

Gegeben seien Aufgaben der Längen 5, 3, 2, 2 und 1 sowie drei Maschinen. Im ersten Schritt werden die Aufgaben sortiert und danach wie in untenstehender Abbildung verteilt.

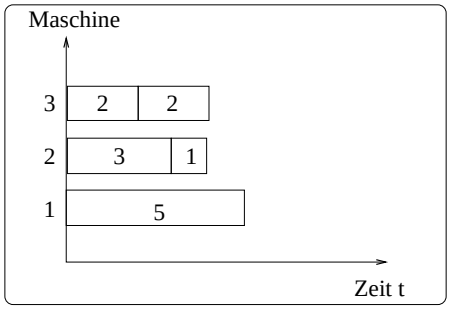

Abbildung 2.9: Aufgaben nach Länge den Maschinen zugeteilt

Dieser Algorithmus könnte noch verbessert werden, indem man zuerst die *k* längsten Aufgaben, die eine größere Auswirkung auf die Gesamtlänge haben als die kürzeren Aufgaben, optimal plant und danach mit obigen Algorithmus fortfährt [Seidel, 1996]. Die Laufzeit setzt sich zusammen aus der Zeit für das Sortieren und dem Verteilen, also *m* Vergleichen bei jeder Aufgabe.

Als Vertreter einer Heuristik wollen wir hier das *Simulated Annealing* betrachten. Es ist ein *Monte Carlo*-Versuch zur Optimierung multivariater Funktionen. Prinzipiell dienen *Monte Carlo*-Simulationen zur numerischen Approximation verschiedenartiger Funktionen, besonders geeignet sind sie bei höherdimensionalen Problemstellungen [Fishman, 1986]. Das *Simulated Annealing* ist ein iteratives statisches Verfahren, dessen Begriff aus der Physik bei der Herausbildung von Kristallstrukturen bei kontrollierten Abkühlungsverfahren stammt.

Bei dem Verfahren wird anfänglich eine zufällige Startkonfiguration ermittelt, die als Basis zur Berechnung einer neuen Konfiguration dient. Ist der Wert, hier der Plan für die Maschinen, besser als der alte Wert, so wird dieser als neuer Startwert genommen. Ist der berechnete Wert schlechter, so wird er trotzdem mit einer gewissen Wahrscheinlichkeit akzeptiert. Dadurch könnte sich die Lösung jedoch von einem lokalen Optimum entfernen. Die Wahrscheinlichkeit, dass neue Konfigurationen den Plan verschlechtern, wird im Laufe des Algorithmus langsam gesenkt, bis sie den Wert 0 annimmt.

Diese Technik erlaubt ein konsequentes Annähern zu einer besseren Lösung, aber auch das ”Springen” aus diesem lokalen Bereich mit seinem lokalen Optimum zu einem anderen Bereich des Suchraums, in dem das gleiche Szenario durchlaufen wird.

Mit diesen beiden Techniken zum effizienten Lösen von Schedulingproblemen wollen wir die Diskussion über diese auch beenden. Es gibt viele weitere Möglichkeiten, dieses in der Informatik populäre Problem zu lösen mit weiteren Techniken und Schranken, im Rahmen dieser Arbeit genügt jedoch der gegebene Überblick.

Kapitel 3

Problemstellung

In diesem Kapitel wird die in dieser Arbeit behandelte Problemstellung diskutiert. Dabei wird zunächst im ersten Abschnitt das Problem beschrieben, danach werden resultierende spezielle Herausforderungen im zweiten Abschnitt erläutert. Mit Bezug auf das vorangegangene Kapitel wird dann im dritten Abschnitt die Relevanz des Problems, sowohl in der Literatur als auch in der Praxis, gezeigt.

3.1 Behandelte Problemstellung

In sämtlichen Produktionsketten, in denen es ein Hierarchiegebilde in der Struktur der beteiligten Institutionen gibt, treffen Personen Entscheidungen, welche Auswirkungen gegenüber anderen beteiligten Personen haben. In einer Spedition zum Beispiel entscheidet ein Disponent darüber, welcher Fahrer an welchem Tag wohin fährt, wobei der jeweilige Fahrer diese Entscheidung akzeptieren muß. In vielen Fällen wäre es dabei jedoch möglich, dass der Disponent auf Wünsche der Personen, die von seinen Entscheidungen abhängen und betroffen sind, eingehen könnte, ohne dass sich dabei sein Resultat sonderlich verschlechtern würde.

Diese Arbeit setzt sich mit oben beschriebener Problematik auseinander, wobei wir dieses Problem hier nun etwas genauer beschreiben wollen. Das Szenario mit der Spedition ist nur ein Beispiel, in dem es zu dieser Situation kommen kann, betrachten werden wir aber den allgemeinen Fall, den man dann in solch speziellere Szenarien überführen kann.

Ein Disponent erhält einen potentiellen Auftrag, dessen Erfüllbarkeit er anhand der ihm zur Verfügung stehenden Ressourcen überprüfen muß. Diese Ressourcen zur Bearbeitung des Auftrags sind begrenzt und umfassen hier zum einen die hierfür notwendigen Maschinen mit ihrer begrenzten Kapazität sowie die Arbeiter, welche diese Maschinen bedienen können und eine bestimmte Gesamtarbeitszeit zur Verfügung stellen. Im Beispiel der Spedition wären dies also der Lastwagen mit seiner begrenzten Ladefläche sowie der Fahrer mit seiner gesetzlich vorgeschriebenen Maximallenkzeit.

Nun darf der Arbeiter Wünsche bezüglich seiner Arbeitszeit äußern, welche bei der Lösung mit in Betracht gezogen werden müssen. Desweiteren ist es dem Arbeiter gestattet, diese Wünsche zudem noch mit Prioritäten zu belegen, deren Wichtigkeit ebenso Einfluß auf die Lösung haben sollen. Im Fall der Spedition könnte der Fahrer zum Beispiel den Wunsch formulieren, dass er eine Tour fahren will, bei der er abends wieder zurück ins Lager kommt anstatt unterwegs übernachten zu müssen. Ein anderer Wunsch könnte sein, dass er die folgende Woche den ihm zustehenden Urlaub in Anspruch nehmen will. Letztgenannter Wunsch könnte ihm dabei viel wichtiger sein als der bezüglich der Tour, worauf er diesen auch mit einer höheren Priorität gewichten würde.

Es entstehen somit Bedingungen, welche nicht überschritten werden können, wie die maximale Kapazität der Maschine oder Präferenzen der Arbeiter, welche nicht gebrochen werden dürfen, sowie Bedingungen, welche von sehr wichtig bis weniger wichtig unterschieden werden, aber durchaus gebrochen werden können. Die Erfüllung des Auftrags hat jedoch Vorrang, so dass die Wünsche, welche nicht unbedingt eingehalten werden müssen aber der Erfüllbarkeit widersprechen, verwehrt werden können.

Die Art und Weise der Gewährung der Wünsche zur Einhaltung eines Auftrags soll deterministisch sein, da jeder Arbeiter gleich behandelt werden soll. Man muß also aus der Menge der möglichen Lösungsmöglichkeiten, welche in Abhängigkeit der Anzahl der Arbeiter sowie der Unterscheidungsmöglichkeiten bezüglich der Wünsche äußerst mächtig werden kann, diejenige Kombination finden, welche zum einen den Auftrag erfüllt und zum anderen die Präferenzen aller Arbeiter gleich behandelt und entsprechend eine Verteilung vornimmt. Zudem sollen möglichst viele Wünsche gewährt werden.

Nachdem wir nun mehrfach den Begriff *Auftrag* verwendet haben, wollen wir ihn für den allgemeinen Fall etwas näher beschreiben. Wenn man die Geschäftswelt reflektiert, so sind folgende drei Punkte bei fast jeder Transaktion wesentliche Bestandteile. Dies sind der Typ des Gutes, damit man überhaupt definieren kann, um was für eine Art Auftrag es sich handelt, die Menge des georderten Gutes ist ebenso unabdingbar und der Termin, bis zu welchem der Auftrag erledigt sein soll. Desweiteren spielt natürlich der Preis eine zentrale, wenn nicht die zentrale Rolle bei jeder Transaktion auf Märkten. Dies ist jedoch eine Verhandlungssache zwischen den beteiligten Institutionen und somit nicht von zentraler Bedeutung in dieser Arbeit.

Man kann also sagen, dass wir für einen Auftrag, der durch einen Typ, eine Menge sowie einen Abgabetermin spezifiziert ist, eine Zuteilung zu einer oder mehreren Maschinen suchen, die diesen Auftrag vom Typ her potentiell bearbeiten können unter der Voraussetzung, dass die Arbeitspläne der Arbeiter noch Kapazität übrig haben. Den Arbeitern sollen hierbei Wünsche bezüglich ihrer Arbeitszeit weitesgehend gewährt werden, sofern sie der Erfüllbarkeit des gesamten Auftrags nicht wirdersprechen.

Die Wissensgrundlage von Disponent und Arbeiter sollen hierbei wie bei den natürlichen Personen auch verschieden sein. Während der Disponent sehr wohl weiß, bis wann er wieviel von einem geforderten Gut bereitstellen soll, hat er keine Kenntnisse

darüber, wie die Kapazitäten der Arbeiter sind. Darüberhinaus kennt er auch die Wünsche der Arbeiter nicht. Er muß diese also fragen, wieviel sie bis zu diesem Termin im Stande sind, zu produzieren. Umgekehrt weiß der Arbeiter nicht, wie hoch der gesamte Auftragsumfang ist und welcher Arbeiter noch für diesen Auftrag in Frage kommt. Es besteht also eine gewisse Abhängigkeit zwischen beiden Akteuren, wobei diese nicht statisch sein muß. Es ist also erlaubt, dass ein Arbeiter von verschiedenen Disponenten kontaktiert werden kann, wogegen der Disponent durchaus verschiedene Arbeiter beauftragen kann.

Desweiteren sollte obiger Prozeß nicht nur einmal, sondern fortlaufend, also *online*, durchlaufen werden können. Es handelt sich also für den Disponenten um folgendes Problem; er soll *n* Aufträge auf *m* Maschinen beziehungsweise Arbeiter verteilen und dies möglichst so, dass die Auslastung aller Maschinen optimal ist. Dieses Problem ist in der Informatik ein sehr bekanntes, nämlich das Planen unabhängiger Aufgaben (englisch: *Scheduling of Independent Tasks*, kurz SIT), welches in Kapitel 2 schon näher beschrieben wurde.

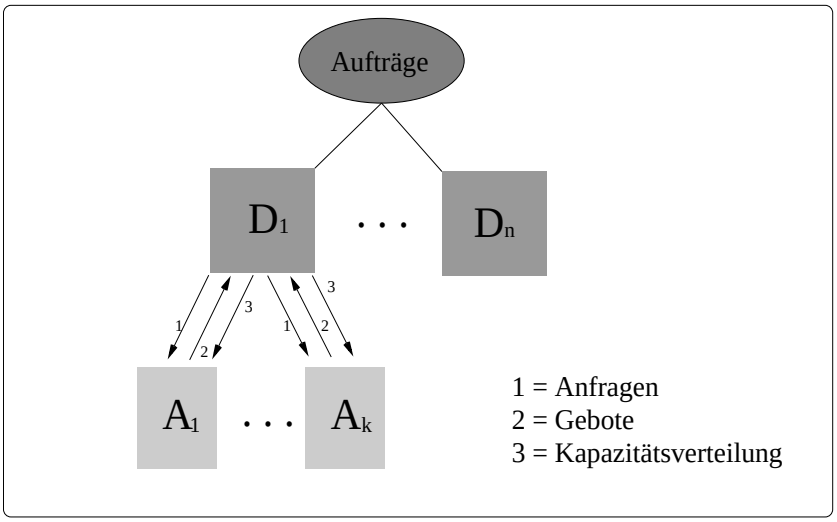

Abbildung 3.1: Problematik der gesamten Aufgabe

Abbildung 3.1 soll die gesamte Problematik nochmals veranschaulichen. Dabei sind die Disponenten durch D_1 bis D_n dargestellt. Sie erhalten fortlaufend Aufträge, welche sie auf die verschiedenen Arbeiter A_1 bis A_k verteilen sollen. Sie können dabei das Gesamtvolumen eines Auftrages auf mehrere Arbeiter verteilen, um eine bessere Verteilung zu bekommen und die Wünsche so besser respektieren zu können. Dies muß unter Umständen sowieso geschehen, wenn die Kapazität des Auftrags die maximale potentielle Produktionsmenge des Arbeiters mit dem höchsten Gebot übersteigt.

Die Arbeiter bearbeiten jede Anfrage, indem sie in ihre Gebote ihre Wünsche mit einfließen lassen. Schließlich muß der Disponent eine Kapazitätsverteilung finden, die einerseits den Auftrag erfüllt und andererseits die Wünsche der Arbeiter respektiert. Darüberhinaus soll die Verteilung der Aufgaben auf die einzelnen Arbeiter so sein, dass die Gesamtauslastung einer jeden Maschine möglichst optimal ist.

3.2 Spezielle Herausforderungen

Nachdem wir die Problemstellung im letzten Teilabschnitt erörtert haben, wollen wir uns hier mit der speziellen Problematik der Kernthemen sowie den einhergehenden Aufgaben beschäftigen, die im Verlauf dieses Szenarios entstehen können.

Durch die Ausgangslage des Problems mit seinen natürlichen Personen, welche während des gesamten Prozesses immer wieder auf Informationen der jeweilig anderen Personen angewiesen sind, wird ein hohes Maß von Datenaustausch beziehungsweise Kommunikation erforderlich sein, zumal die beteiligten Personen schon rein vom räumlichen Aspekt her getrennt sein können. Auch die Flexibilität der beteiligten Personen, welche fortlaufend immer wieder mit anderen Personen ähnliche Arbeitsabläufe vollziehen können sollen, ist ein wichtiger Faktor. Demnach ist die Konzeption des gesamten *Softwarepaketes*, welches den Rahmen für die Realisierung bildet, von großer Bedeutung. Diese Rahmenbedingungen für das System sollten also von Grund auf schon Aspekte wie die Kommunikation oder die Flexibilität der beteiligten Akteure unterstützen, damit ein fehlerloser Ablauf gesichert wird.

Aus diesen Anforderungen heraus drängt sich direkt der Gedanke auf, diese natürlichen Personen als *Softwareagenten* auf einer geeigneten Agentenplattform zu repräsentieren. Diese Art der Umsetzung bietet mit seinen unterstützten Funktionalitäten bezüglich der Kommunikation und der Flexibilität einen prädestinierten Rahmen, was wir hier aber nicht weiter vertiefen wollen, sondern im Rahmen der Implementierung in Kapitel 5 diskutieren werden.

Der Disponent erhält also aus dem Markt, in dem er sich befindet, einen potentiellen Auftrag, der durch einen Typ, eine Menge sowie einen Abgabetermin spezifiziert ist. Seine erste Aufgabe besteht nun darin, mit denjenigen Arbeitern Kontakt aufzunehmen, welche diesen geforderten Typ produzieren können. Anhand des Abgabtermins sollen diese ihm ein Gebot schicken, welches beinhaltet, wieviel sie bis zu diesem Termin produzieren können.

Auf der Grundlage der von den Arbeitern zurückgesendeten Gebote muß der Disponent dann entscheiden, welcher Arbeiter wieviel zum Erreichen des Gesamtvolumens beitragen soll. Während dieser Entscheidung verfolgt er gewisserweise zwei Ziele wie wir dies eingangs des Kapitels erörtert haben. Zum einen versucht er, den Auftrag zu erfüllen, also eine Verteilung der Kapazitäten auf die einzelnen Arbeiter zu finden, welche mindestens den Umfang der geforderten Menge entspricht. Desweiteren soll er versuchen, die Wünsche der Arbeiter gerecht zu behandeln und möglichst alle zu respektieren, wobei jedoch die Erfüllung des Auftrags Vorrang hat.

Bei der Lösung dieses Problems sind demnach zwei wichtige Aspekte zu beachten. Der Suchraum bezüglich der möglichen Lösungen kann in Abhängigkeit der Einteilung der Präferenzen seitens der Arbeiter sowie der Arbeiteranzahl bezüglich des zu bearbeitenden Auftrags sehr mächtig werden. Um diesen Suchraum von Beginn an ein wenig einzuschränken, sollten die Arbeiter ihre Wünsche gewissen Klassen, welche ausgewählten Wichtigkeiten entsprechen, zuordnen. Nun soll jede Kombination über alle Arbeiter möglich sein inklusive dem Fall, dass ein Arbeiter zu diesem Auftrag nichts beiträgt. $(|Prioritätsklassen|+1)^{|Arbeiter|}$ ist somit die Anzahl der potentiellen Lösungen, welche nicht mehr polynomiell entscheidbar ist, da die Anzahl der Arbeiter die sich ständig ändernde Größe ist. Man sollte daher versuchen, die Basis nicht zu groß werden zu lassen, damit die Antwortzeiten in einem akzeptablen Rahmen bleiben. Trotzdem sollten dem Arbeiter immer noch genügend Klassifizierungsmöglichkeiten geboten werden, damit die Gewichtung überhaupt noch sinnvoll ist.

Abbildung 3.2 stellt den gerade beschriebenen groben Ablauf für einen Auftrag im Disponenten graphisch dar. Dabei muß er erst einmal anhand des Typs feststellen, welche Arbeiter diesen Auftrag überhaupt bearbeiten können. Mit diesen Arbeitern nimmt er folglich Kontakt auf, und fordert ein Gebot bezüglich dieses Auftrags und dem mitgesandten Abgabetermin. Wenn der Disponent die Gebote aller beteiligten Arbeiter erhalten hat, muß er bezüglich den beiden genannten Zielen ein optimale Lösung finden. Diese soll zum einen den Auftrag erfüllen und zum anderen die Wünsche der Arbeiter, welche mittelbar in den Geboten enthalten sind, weitestgehend respektieren.

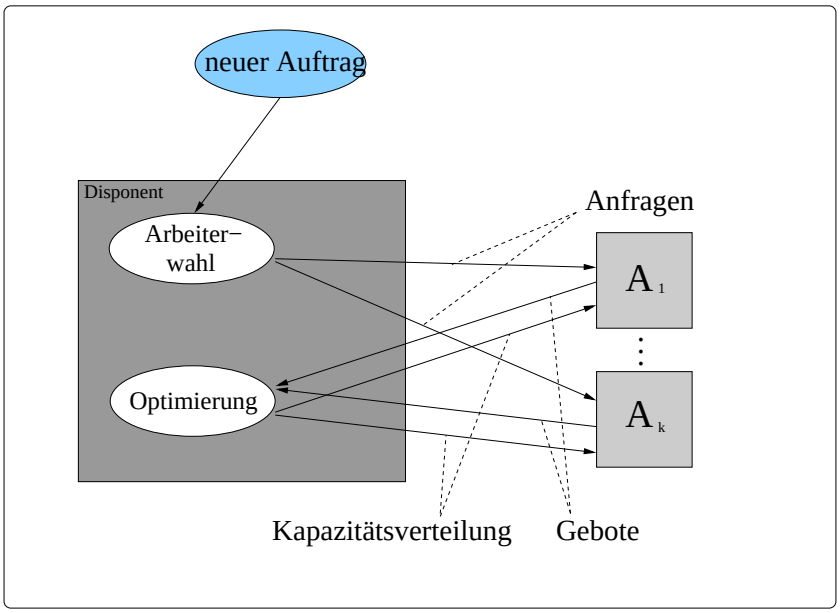

Abbildung 3.2: Ablauf im Disponenten

Betrachten wir nun den Arbeiter mit seinen aus der Aufgabe resultierenden speziellen Herausforderungen. Das Kernproblem für den Arbeiter ist das Einbeziehen seiner

Wünsche. Er wird, wie im vorangegangenen Abschnitt erörtert, vom Disponenten gefragt, wieviel er bis zu dem den Auftrag betreffenden Abgabetermin im Stande ist zu produzieren. Da der Disponent keine Kenntnisse über die Präferenzen des Arbeiters hat, soll der Arbeiter sie in seine Antwort mit einbeziehen.

Der Arbeiter muß demnach seine Wünsche formulieren und sie dem System mitteilen, welches dann daraus die Gebote, welche der Disponent angefordert hat, berechnet. Es entsteht dabei eine sogenannte *Constrainthierarchie*, wie sie in Kapitel 2 definiert wurde. Diese muß nun von dem System gelöst werden, wobei dies durch einen geeigneten Solver erledigt werden soll, der die so entstandene Instanz der Klasse *HCSP* entscheiden kann. Im vorangegangenen Kapitel haben wir schon einige dieser Systeme vorgestellt, eine endgültige Entscheidung wird jedoch erst im Rahmen der Implementierung in Kapitel 5 getroffen.

Auch die bereits erwähnte Klassifizierung der Wünsche in bestimmte Prioritätsklassen zur Vermeidung unnötig großer Suchräume während der Optimierung im Disponenten ist ein Problem, welches im direkten Zusammenhang mit dem gewählten Solver steht. Diese Klassen werden im Zusammenhang mit denen im Solver unterstützten Komparatoren verschiedener Lösungen entschieden. Es ist jedoch wichtig, daß die Anzahl der Klassifizierungsmöglichkeiten seitens des Arbeiters nicht zu groß wird, da sie die Basis der exponentiell vielen Lösungsmöglichkeiten für den Disponenten bildet $((|Prioritätsklassen| + 1)^{|Arbeiter|})$.

Die Gesamtlaufzeit des Systems zur Bearbeitung eines Auftrags ist ein sehr wichtiger Aspekt, der bei der Konzeption stets beachtet werden soll. Um den Prozeßablauf nicht unnötig zu verlängern, sollte der Arbeiter demnach seine Wünsche unabhängig von einem Auftrag dem System mitteilen können. Hierfür ist eine graphische Benutzerschnittstelle erforderlich, über welche der Arbeiter mit dem System kommunizieren kann. Der *Softwareagent* sollte dann stellvertretend für den Arbeiter bei einer konkreten Anfrage seine Wünsche in Constraints transferieren und diese dem ausgewählten Solver übergeben. Der Agent sollte dabei, wie schon mehrfach erwähnt, flexibel handeln und reagieren können. Diese Eigenschaften weist ein nach der in Kapitel 2 vorgestellten INTERRAP-Architektur strukturierter Agent auf.

Abbildung 3.3 soll den Ablauf im Arbeiter nochmals graphisch darstellen. Der *Taskmanager des Arbeiters* bildet dabei die Schnittstelle des *Softwareagenten* nach außen. Er erkennt die eintreffenden Nachrichten und kann dementsprechend die verschiedenen Methoden in den Komponenten des Arbeiters starten.

3.3 Relevanz des Problems

In diesem Abschnitt wollen wir die Arbeit in die Forschungsgebiete einordnen, zu denen sie einen Beitrag leistet oder deren Erkenntnisse sie sich zu Nutzen macht. Desweiteren soll auch die praktische Relevanz der Arbeit gezeigt werden.

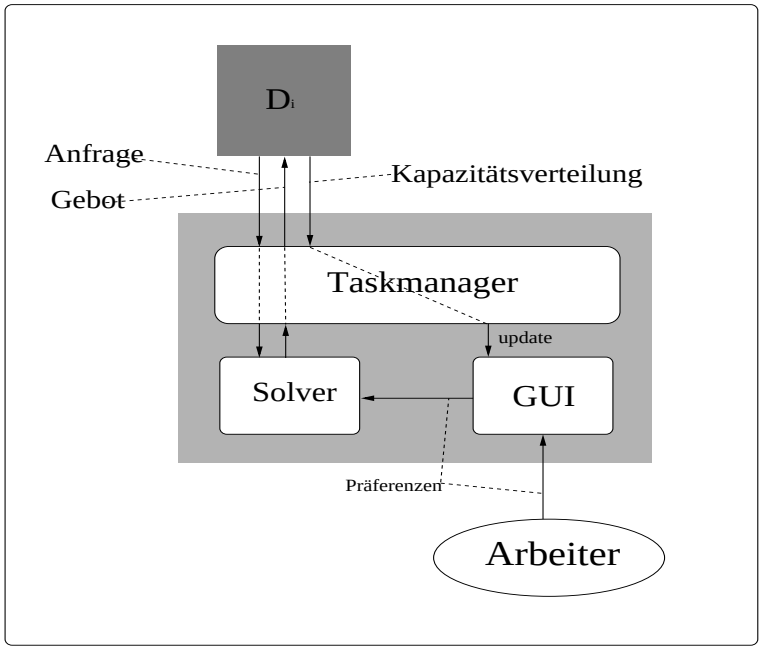

Abbildung 3.3: Ablauf im Arbeiter

3.3.1 Forschungsbeitrag

In Kapitel 2 haben wir die drei Kerngebiete, in denen diese Diplomarbeit einzuordnen ist, schon näher beschrieben. Diese Kerngebiete umfassen die Multiagentensysteme, das Hierarchische Constraint Solving sowie das Scheduling. In diesem Teilabschnitt werden wir erörtern, wie wir diese Forschungsgebiete miteinander in Zusammenhang gebracht haben und welchen Nutzen wir daraus ziehen können.

Im Verlauf dieses Kapitels haben wir aus der Problemstellung heraus die speziellen Herausforderungen der gestellten Aufgabe beschrieben. Aus dieser geht hervor, dass wir es prinzipiell mit einem vektoriellen Entscheidungsproblem (VEM) zu tun haben. Ein solches VEM ist gegeben, wenn man bezüglich einer Menge von Entscheidungsvariablen verschiedene Ziele verfolgt, die zudem noch miteinander konkurrieren. Also haben wir eigentlich zwei Zielfunktionen, welche wir erfüllen wollen; in unserem Kontext sind dies die optimale Verteilung der Aufgaben und die optimale Gewährung der Wünsche der Arbeiter, welche dem erstgenannten Ziel widersprechen.

In dieser Arbeit bewältigen wir diese Problematik, indem wir diese beiden Ziele durch gewisse Vorleistungen voneinander kapseln und dezentral Lösungen berechnen, mit denen die jeweils andere Komponente ihre Berechnungen fortsetzen kann. Dies ist zugleich eines der Hauptargumente für ein Multiagentensystem, als welches das gesamte System schließlich implementiert wird.

Bezüglich der Präferenzen der Arbeiter machen wir uns hier die Erkenntnisse aus dem Forschungsbereich über das Constraint Solving zu Nutzen. Speziell betrachten

wir das *Soft Constraint Solving*, indem wir einen geeigneten Solver für die Behandlung der Wünsche im Arbeiter benutzen. Die mit Hilfe dieses Solvers berechneten Lösungen im Agenten, welche danach an den Disponenten gesendet werden, entsprechen den Wünschen des Arbeiters. letztliche Entscheidung der Gewährung liegt natürlich nicht in seiner Macht, aber mit dieser Vorleistung haben wir den Rechenaufwand im Disponenten gewaltig verringert.

Im Disponenten wird nun die letztendliche Entscheidung, welche beide Ziele angeht, getroffen. Dabei wird das Planungsproblem mit einer allgemeingültigen guten Schranke gelöst und mit den Lösungen aus den Arbeiter-Agenten so kombiniert, dass man nur noch eine Zielfunktion benötigt, die aber als Nebeneffekt auch das andere Ziel mit erfassen kann.

Diese Art der Kombinierung der beiden Problemfelder und dem Erreichen der guten Ergebnisse bezüglich beider Ziele machen letztlich die Motivation für diese Arbeit aus. Auch die praktische Relevanz der behandelten Problematik, welche im folgenden Abschnitt noch näher erörtert wird, ist ein wichtiger Aspekt dieser Arbeit.

3.3.2 Praktische Anwendung

Die zunehmende Bedeutung des Internets in fast allen geschäftlichen Prozessen bildet ein Hauptargument für die vorliegende Arbeit. Viele Interaktionen zwischen Firmen, die früher über andere Kommunikationsmechanismen abgewickelt wurden, werden heute auch über das Internet abgeschlossen. Es ist daher wichtig, dass hierfür auch adäquate Plattformen geschaffen werden, die der realen Geschäftswelt entsprechen und diese Prozesse auch besser unterstützen, als dies vorher möglich war.

In dem Projekt *TeleTruck* am DFKI [Bürckert et al., 1998] werden etwa Fahrten von Speditionen verplant, welche vorher von den Spediteuren von Hand eingeplant wurden. Das System dient als Unterstützung für diese Planung oder kann diese auch vollständig übernehmen und dabei sogar eine bessere Auslastung garantieren als dies mit der Einplanung von Hand in vergleichbarer Zeit möglich wäre. Es ist denkbar, die Ergebnisse dieser Arbeit in das Projekt eingehen zu lassen, um etwa den Fahrern Wünsche bezüglich ihrer Lenkzeiten am Steuer gewähren zu können.

In dem Sozionik-Projekt [Schillo and Klein, 2001], in dessen Rahmen diese Arbeit gestellt wurde, werden Auktionen vertrauensbasiert gestartet und abgewickelt. Die dabei verhandelten Aufträge können unter Benutzung dieses Systems dann auch innerhalb einer Institution, welche durch Agenten auf diesen Plattformen vertreten sind, effizienter bearbeitet werden.

Die Erkenntnisse und Ergebnisse dieser Arbeit können also überall dort eingesetzt werden, wo auf elektronischen Märkten mehrere Institutionen zum Erreichen eines Auftragsziels benötigt werden und dabei zudem auch die Präferenzen der zur Realisierung benötigten Arbeiter eine Rolle spielen sollen.

Das Medium zur Kommunikation kann dabei recht unterschiedlich sein, was das mögliche Einsatzgebiet erheblich vergrößert. Über das Internet ist dies zum Beispiel möglich, womit der Markt, auf welchem diese Transaktionen durchgeführt werden

können, räumlich praktisch nicht mehr eingeschränkt ist. Ein Einsatz über *Personal Digital Assistants* (PDA) ist ebenso denkbar genauso wie über mobile Telefone, welche das *Wireless Application Protocol* (WAP) unterstützen. Dies wird bereits in dem am DFKI entwickelten CASA-Projekt genutzt [Gerber and Klusch, 2001]. Die Benutzer könnten somit jederzeit und auch fast an jedem Ort über das System verfügen.

Kapitel 4

Spezifikation

In diesem Kapitel wird ausgehend von der detaillierten Problemstellung und -analyse bestimmt, welche Kriterien das System erfüllen muß. Es dient als Basis für die Implementierung und Evaluierung, also der Realisierung der in in diesem Kapitel folgenden Aspekte. Das gesamte Kapitel kommt einer Instanzaufnahme gleich, es wird gezeigt, *was* realisiert werden soll, *wie* man es tatsächlich umsetzt, wird im nächsten Kapitel *Implementierung und Evaluierung* beschrieben. Die Gliederung der Spezifikation richtet sich nach der Methodik von Balzert [Balzert, 1998].

Im ersten Abschnitt werden die Ziele, die durch das System erreicht werden sollen, nochmals kurz beschrieben, wobei jedoch eine Klassifizierung unter ihnen vorgenommen wird. Es wird zwischen Mußkriterien, also Zielen, die unbedingt erreicht werden müssen, Wunschkriterien, also nicht unabdingbaren aber doch erstrebenswerten Zielen, sowie Abgrenzungskriterien, welche jene Ziele umfassen, die man bewußt nicht erreichen will, unterschieden. In diesem Abschnitt wird also der Entscheidungsraum für die Realisierung des Problems abgesteckt.

Im folgenden Abschnitt wird dann kurz auf den Einsatz des Systems eingegangen. Der Anwendungsbereich sowie die Betriebsbedingungen sind für den Entwurf ein nicht zu vernachlässigender Faktor. Danach wird die Umgebung des Systems genauer beschrieben. Hierzu zählt die *Software*, also Betriebssystem oder Datenbank, welche auf der Maschine oder den Maschinen, auf denen das System eingesetzt werden soll, verwendet wird. Auch die *Hardware* ist sehr wichtig, um etwa die Anforderungen an das System von der benutzten Software nicht zu überschreiten. Schließlich spielt auch die *Orgware* einen wichtigen Aspekt beim Entwurf, hierbei werden die organisatorischen Rahmenbedingungen, unter denen das System letztendlich eingesetzt werden soll, analysiert.

Im vierten Teilabschnitt des Kapitels wird auf die Funktionalität des Systems eingegangen. Die aus den eingangs des Kapitels erarbeiteten Ziele werden in Funktionen formuliert, welche die Details, die für die Realisierung notwendig sind, enthalten. Es werden die Einzelanforderungen an die jeweiligen Komponenten in verbaler Form konkretisiert, jedoch nicht auf deren Umsetzung eingegangen.

In dem Abschnitt über die Produktdaten und -leistungen werden wir einen kur-

zen Überblick auf die längerfristig zu speichernden Daten geben. Desweiteren werden auf die zeitbezogenen und umfangsbezogenen Anforderungen erörtert, also maximaler Datenumfang, Antwortzeiten oder dergleichen.

Schließlich wird noch die Benutzeroberfläche spezifiziert. In diesem Abschnitt werden die einzelnen Gesichtspunkte wie Bildschirmlayout oder Dialogstruktur festgelegt. Auch sollen die Schnittstellenkonventionen zwischen den einzelnen Komponenten hier beschrieben werden.

4.1 Ausgangslage und Zielsetzungen

Die Aufgabe des Systems besteht darin, softwarebasierend eine optimale Lösung für folgendes Problem zu finden. Ein Disponent soll für einen Auftrag, der an einen Typ, eine Menge und einen Abgabetermin gebunden ist, eine optimale Verteilung des Gesamtvolumens auf alle Arbeiter, die diesen Auftrag potentiell bearbeiten können, berechnen. Der Begriff des *Auftrags* ist somit von der Syntax her gesehen vorgegeben. Semantisch bedeutet er für den Disponenten eine Anfrage von außen, dem elektronischen Markt, ob er gewillt und im Stande ist, diesen spezifizierten Auftrag anzunehmen und zu erfüllen. Den Arbeitern, die letztlich den Auftrag bearbeiten, sollen hierbei ihre Wünsche bezüglich ihrer Arbeitszeit weitestgehend gewährt werden. Die Entscheidung darüber, wem welche Wünsche gewährt werden, liegt jedoch wiederum beim Disponenten, der darüber allerdings deterministisch zu entscheiden hat.

Da das gesamte System in ein Multiagentensystem integriert werden soll und zudem das Wissen von Disponent und Arbeiter als natürliche Personen aus dem anwendungsspezifischem Szenario heraus voneinander verschieden sind, ordnen wir sie schon in diesem Abschnitt eigenen Agenten zu. Normalerweise würde diese Trennung erst im nächsten Abschnitt teilweise und im nächsten Kapitel vollständig vorgenommen werden, aber da dieser Aspekt der Aufgabenteilung von der Problemstellung her schon intuitiv klar ist, wollen wir dies schon hier im Ansatz tun, in der Folge aber dann erst konkretisieren.

4.1.1 Disponent

Mußkriterien

Die wichtigste Entscheidung, die der Disponent treffen muß, ist die Frage der Erfüllbarkeit des ihm angetragenen Auftrags. Am Ende eines jeden Prozesses des Systems muß diese Frage also eindeutig beantwortet sein.

Darüberhinaus soll er die Wünsche der Arbeiter in einer eindeutigen, also algorithmischen Weise behandeln, um für vergleichbare Instanzen zu vergleichbaren Lösungen zu gelangen. Dies ist außerdem den Arbeitern gegenüber die *gerechteste* Art der Behandlung ihrer Wünsche.

Der Disponent verfolgt also zwei Hauptziele, zum einen das Entscheidungsproblem bezüglich der Erfüllbarkeit des Auftrags und zum anderen das Optimierungsproblem bezüglich der Wünsche der Arbeiter. Diese Ziele können miteinander konkurrieren, wobei jedoch das erstgenannte, also die Erfüllung des Auftrags, Vorrang hat.

Der gesamte Prozeß soll jedoch nicht nur einmal vom System durchgeführt werden können und auch nicht zwangsläufig immer mit den selben Arbeitern, sondern fortlaufend mit Aufträgen verschiedensten Typs, was dann auch mit verschiedenen Arbeitern möglich sein muß. Das System soll also in Abhängigkeit vom Typ des Auftrags erkennen können, mit welchen Arbeitern es in Kontakt treten soll. Darüberhinaus muß es *online* arbeiten, also jederzeit neue Aufträge bearbeiten können.

Auch die Kommunikation mit den Arbeitern soll das System komplett übernehmen, nach der Suche nach den möglichen Arbeitern soll das Programm also auch direkt seine Anfragen an diese stellen und deren Antworten ebenso ohne die konkrete Mitwirkung des realen Disponenten bearbeiten.

Die oben fünf genannten Aspekte sind also unabdingbare Anforderungen an das Produkt, ohne welche der Einsatz des Systems zu keinen erheblich neuen und besseren Ergebnissen führen würde.

Wunschkriterien

Wunschkriterien wie eingangs des Kapitels erwähnt gibt es für den Disponenten eigentlich nicht, da sein Aufgabenbereich klar abgegrenzt ist und somit nur wenig Raum für Änderungen im Rahmen dieser Arbeit läßt. Der einzige Aspekt, der an das System gewissermaßen als Wunsch gestellt werden kann, ist eine benutzerfreundliche Bedienung sowie eine Kontrollinstanz für die eingeplanten und verteilten Aufträge, wobei diese Aspekte zumindest bezüglich der Benutzerfreundlichkeit selbstverständlich sind.

Abgrenzungskriterien

Wenn der Disponent einmal einen Auftrag angenommen und eingeplant hat, soll er nicht mehr im Nachhinein abgegeben werden. Dies könnte sehr schnell zu *deadlocks* führen zum einen durch die Kommunikation zwischen den Agenten und zum anderen durch die damit entstehende nötige Neuplanung, welche ihrerseits wiederum zu potentiellen Neuplanungen führen könnte.

Ein weiterer Aspekt, der nicht vom Disponenten erfüllt werden soll, ist die Behandlung der Wünsche innerhalb eines Arbeiters. Der Disponent soll lediglich die Antworten der Arbeiter zusammen betrachten und anhand der geforderten Kapazität prüfen, ob er den Auftrag überhaupt erfüllen kann. Wie die Arbeiter zu ihren Antworten gelangen, ist nicht die Aufgabe des Disponenten. Er kann demnach auch nichts mit dem Abgabetermin des Auftrags anfangen, welchen er folglich auch an die Arbeiter weitergeben muß.

Dieser Aspekt der Aufgabenteilung ist sogar ein gefordertes Kriterium von Multiagentensystemen, in denen die Agenten autonom mit verschiedenen Instanzen eigenständig arbeiten sollen. Die Lösungen, die dabei von den Agenten berechnet werden, können wieder als Eingabeinstanz eines anderen dienen.

4.1.2 Arbeiter

Für den Arbeiter werden wir zu Beginn erst den Begriff des *Gebotes* erklären, da er im Verlauf dieser Spezifikation immer wieder gebraucht wird. Ein Gebot eines Arbeiters beinhaltet die seinen Präferenzen entsprechenden verschiedenen Produktionsmengen, dieses Gebot schickt er an den Disponenten, der dieses von ihm gefordert hat, innerhalb eines Sprechaktes zurück. Das Gebot des Arbeiters entspricht also der Antwort auf die Frage des Disponenten, wieviel er bis zu einem bestimmten Termin im Stande ist, zu produzieren.

Mußkriterien

Die Frage, welche Ziele der Arbeiter erreichen muß, soll in diesem Teilabschnitt beantwortet werden. Wie schon soeben angedeutet, besteht die Hauptaufgabe des Arbeiters darin, dem Disponenten mitzuteilen, wieviel er bis zu dem übergebenen Abgabetermin im Stande ist zu produzieren.

Dem menschlichem Arbeiter soll hierbei die Möglichkeit geboten werden, seine Präferenzen bezüglich seiner Arbeitszeit dem System zu übergeben. Folglich muß ihm das System eine graphische Oberfläche bieten, welche als Schnittstelle zwischen ihm und dem System dient und über die er mit dem System kommunizieren kann. Zudem soll er seine Wünsche jederzeit, also unabhängig von einem potentiellen Auftrag formulieren können.

Desweiteren soll das System in Abhängigkeit der Klassifizierung der Wünsche die Antwort für den Disponenten berechnen. Folglich muß in diesem Bereich des Systems algorithmisch diese Lösung berechnet werden, damit die deterministische Behandlung dieser *constraints* gewährleistet ist.

Das Produkt soll zudem die Kommunikation mit dem Disponenten übernehmen, dem menschlichem Arbeiter soll somit vom Programm sein für ihn berechneter Plan direkt übermittelt werden, wobei er lediglich seine Wünsche über eine GUI formulieren muß. Dieser Plan soll nach jedem neu angenommenen Auftrag aktualisiert werden, sofern es eine Änderung für den Arbeiter gab. Ebenso sollen seine Gebote ohne seine konkrete Hilfe direkt vom System an den Disponenten übermittelt werden.

Die seitens des Systems komplett abgewickelte Kommunikation mit dem Disponenten ist ein sehr wichtiger Aspekt, da er die gesamte Perfomanz des Produktes, die in der Arbeitswelt durch eben diese doch zum Teil recht erheblich beeinträchtigt wird, verbessert.

Wunschkriterien

Auch hier ist es wie beim Disponenten durch die klare Abgrenzung des Problems in seine Teilprobleme, welche wiederum eindeutig zu definieren sind, schwer zu formulieren welche Ziele noch erstrebenswert sind, wenn sie ohnehin nicht schon zwingend erforderlich sind.

Abgrenzungskriterien

Da wir das System eingangs des Kapitels schon in seine Teilprobleme unterteilt haben, die sich aus den Problementitäten klar ergeben haben, ist es hier nötig darauf hinzuweisen, dass sich die einzelnen Teilaufgaben nicht überschneiden dürfen. Dieser Aspekt wurde jedoch schon in Kapitel 2 unter dem Aspekt *Scheduling* erklärt.

Klar abgrenzen muß man jedoch, dass die Aufgabe des Arbeiters nicht darin besteht, darüber zu entscheiden, welche Wünsche gewährt werden und welche nicht. Die Aufgabe des Arbeiters endet also nach Formulierung der Präferenzen und den daraus resultierenden Geboten des Arbeiters.

4.2 Produkteinsatz

Nachdem wir im vorangegangenen Abschnitt die Ausgangslage sowie die Ziele beschrieben haben, soll in diesem Abschnitt über den Einsatz des Systems gesprochen werden. In Softwareunternehmen ist dieser Aspekt sehr wichtig, da die Zielgruppen generell über recht unterschiedliche Kenntnisse mit dem Umgang von Software verfügen, und ein Unternehmen ja nur nach konkreten Aufträgen arbeitet. Diese Arbeit entstand innerhalb eines Projektes an der Universität des Saarlandes und hat demnach nicht unbedingt konkrete Zielgruppen. Potentiell kann man es jedoch in sämtlichen Unternehmen verwenden, die computergestützt ihre Aufträge bearbeiten und intern die dafür notwendigen Arbeitsschritte verteilen. In dem Sozionik-Projekt, zu dem diese Diplomarbeit einen Beitrag leistet, wird es auf einer Plattform, auf der Agenten vertrauensbasiert miteinander kooperieren und Aufträge innerhalb von Auktionen verhandeln, verwendet. Speziell soll ein potentieller Auftrag auch über diesen elektronischen Markt an die einzelnen Arbeiter, welche auch als Agenten auf dieser Plattform enthalten sind, verteilt werden unter den spezifizierten Rahmenbedingungen.

Auch ist ein Einsatz in dem Projekt *TeleTruck* am DFKI, welches die einzelnen Routen in einer Spedition plant, denkbar. Die Fahrer könnten somit ihre Wünsche bezüglich der zu fahrenden Strecken formulieren [Bürckert et al., 1998].

4.3 Produktumgebung

Hier wollen wir uns mit der Umgebung für Entwicklung und Einsatz der Software beschäftigen, was auch die zu verwendende Hardware sowie Betriebssystem umfaßt. Die Wahl der Programmiersprache wird maßgeblich durch die Agentenplattform, auf der das System integriert werden soll, vorgegeben. Diese Plattform stellt zugleich die Produktschnittstelle dar, da das System eine Komponente eines größeren Projektes ist, welches diese Agentenplattform als Grundlage benutzt.

Da FIPA-OS diese Plattform ist, welche auf Java-Technologien basiert und die in Kapitel 2 erklärten FIPA-Standards für Agententechnologien implementiert, scheint Java die Sprache zu sein, mit der die Umsetzung am sinnvollsten ist, um etwaige

Schnittstellenprobleme mit anderen Sprachen zu vermeiden. Wir wollen hier aber nicht näher auf FIPA-OS mit seinen speziellen Protokollen, welche im nächsten Kapitel nochmals näher beschrieben werden, eingehen.

Als Voraussetzung seitens FIPA-OS sei aber noch erwähnt, dass es einen Pentium 166MHz-Prozessor benötigt sowie mindestens 64 MB Arbeitsspeicher und 20 MB freien Speicherplatz. Getestet wurde es hauptsächlich auf Windows 2000, ist aber auch auf Linux und Solaris einsetzbar (FIPA-Homepage). Diese recht geringen Anforderungen an die Hardware machen einen breiten Einsatz des Systems möglich, ohne dass sich die beteiligten Institutionen eine komplett neue Hardware zulegen müssen.

Durch seine Plattformunabhängigkeit bietet Java zudem die Möglichkeit, das System auf verschiedenen Betriebsystemen miteinander laufen zu lassen, womit wir uns hier nicht auf eines festlegen müssen.

4.4 Produktfunktionen

In diesem Teilabschnitt wollen wir die Produktfunktionen, welche aus den Zielen hervorgehen, die wir im ersten Abschnitt dieses Kapitels erörtert haben, beschreiben. Es soll die funktionale Beschreibung aus Benutzersicht erfolgen, welche die Details, die zur Realisierung der beschriebenen Aspekte benötigt werden, erörtert. Dieser Abschnitt soll einen Überblick über das System geben und die Faktoren beschreiben, welche die Konzeption des Produktes maßgeblich beeinflussen.

4.4.1 Disponent

Hier werden die Funktionen, die der Disponent enthalten muß, erörtert. Dieser bekommt aus dem elektronischen Markt, wie in der Problemstellung in Kapitel 3 und eingangs dieses Kapitels beschrieben, einen Auftrag, der an einen Typ, eine Kapazität sowie einen Abgabetermin gebunden ist. Anhand dieser Informationen muß das System nun herausfinden, welche Arbeiter überhaupt für diesen Auftrag in Frage kommen. Die Funktion erhält also als Parameter die oben drei genannten Größen anhand derer sie die Arbeiter bestimmen soll, mit denen der Disponent für diesen Auftrag weiter arbeitet. Die Ausgabe dieser Funktion ist demnach eine Menge, welche die entsprechenden Arbeiter enthalten muß.

Der Rückgabewert dieser ersten Methode dient gleichzeitig als Eingabe für die zweite Funktion, welche nun beschrieben wird. Der Disponent muß nun mit eben diesen Arbeitern in Kontakt treten und sie fragen, wieviel sie bis zu dem Abgabetermin des Auftrags produzieren können. Die hierfür notwendige Kommunikation muß ohne Einwirkung von außen, also ohne den menschlichen Arbeiter gestartet werden. Desweiteren muß der Disponent den Abgabetermin mit an die Arbeiter übergeben, damit diese ihre notwendigen Arbeitsschritte dann ausführen können.

Nachdem er die Gebote der Arbeiter erhalten hat, muß er dann die optimale Lösung bezüglich der Wünsche der Arbeiter, aber unter der Gewährleistung der Erfüllung

des Auftrags berechnen. Ein jedes Gebot eines Arbeiters beinhaltet dabei vier verschiedene Mengen, die der Arbeiter unter Gewährung von Präferenzen gewisser Wichtigkeit produzieren kann. Aufgabe des Systems ist es nun, aus dieser Menge die Kombination zu finden, bei welcher der Auftrag erfüllt ist und die Wünsche aller Arbeiter bestmöglich gewährt sind. Diese sicherlich sehr komplexe Funktion beinhaltet also das Planen der Aufgaben auf die einzelnen Maschinen, dessen Problematik in Kapitel 2 unter dem Abschnitt des Scheduling genauer beschrieben wurde. Hinzu kommt die Aufgabe der bestmöglichen Einhaltung der Wünsche der Arbeiter.

Anhand der Lösung dieser Funktion muß nun das System wiederum ohne Einwirkung von außen den entsprechenden Arbeitern die ihnen zugeteilte Menge miteilen, auch wenn die Arbeiter nichts zu diesem Auftrag beitragen sollen oder der Auftrag gar nicht erfüllt werden kann, wenn nämlich die Gesamtkapazität des Auftrags die Summe aller maximalen Produktionsmengen aller Arbeiter übersteigt. Zudem muß auch die Klasse von Wünschen, die dem jeweiligen Arbeiter gewährt wird, übergeben werden.

Hiermit ist die Arbeit für das System bezüglich eines Auftrags erledigt. Wie dieses Szenario dem Disponenten graphisch dargestellt wird und wie er diesen Prozeß starten kann, wird in diesem Kapitel unter dem Abschnitt 4.6 der Benutzeroberfläche erklärt. Obiger Prozeß soll von dem System fortlaufend, also *online* bearbeitet werden können. Auffallend sind hierbei die immer gleichen Kommunikationsschritte zwischen Disponent und den Arbeitern, daher ist es sinnvoll, diese Kommunikation innerhalb eines festen Protokolls laufen zu lassen.

4.4.2 Arbeiter

In diesem Teilabschnitt werden die Funktionen, welche das System für den Arbeiter übernehmen soll, beschrieben. Die Hauptaufgabe des Produktes im Bereich des Arbeiters liegt in der Behandlung der Wünsche der Arbeiter und den hieraus resultierenden Lösungen. Im Agenten, der den Arbeiter repräsentiert, findet also der zweite Hauptaspekt der Arbeit, nämlich das *Soft Constraint Solving* statt.

Hierfür muß das System demnach dem Arbeiter erst einmal die Möglichkeit bieten, ihm die Präferenzen mitzuteilen. Dies wird über eine graphische Benutzerschnittstelle geschehen, welche in Abschnitt 4.6 näher beschrieben wird. Hier ist nur wichtig, dass das System die Präferenzen seitens des Arbeiters mitgeteilt bekommt, da diese notwendige Parameter für die Funktionen im Arbeiteragenten sind. Desweiteren bleibt anzumerken, dass der Arbeiter seine Wünsche jederzeit, also unabhängig von einem konkreten Auftrag formulieren können soll.

Wenn der Arbeiter eine Anfrage seitens des Disponenten erhält, so beinhaltet diese, wie im vorangegangenen Abschnitt bereits erarbeitet, den Abgabetermin für den Auftrag. Anhand dieser Eingabe muß es eine Funktion geben, welche hieraus die Gebote des Arbeiters berechnet. Innerhalb dieser Berechnung findet das *Soft Constraint Solving* statt. Das System muß die Wünsche, die dem Zeitraum des Auftrags entsprechen, in Betracht ziehen und zusammen mit dem Abgabetermin die Gebote berechnen.

Der Rückgabewert dieser Funktion beinhaltet dann also vier verschiedene Kapazitäten, welche genau den Mengen entsprechen, die der Arbeiter unter Berücksichtigung der verschiedenen Präferenzklassen produzieren kann. Diese Menge muß das System dann innerhalb des eben erwähnten Protokolls dann an den Disponenten senden, damit dieser dann seine Optimierung starten kann.

Eine nächste Funktion erhält als Eingabe die vom Disponenten zugeteilte und herzustellende Menge sowie die Art der Gewährung der Präferenzen. Diese Daten erhält der Arbeiter wiederum aus dem schon angesprochenen Kommunikationsprotokoll. Anhand dieser Parameter muß das System nun den Plan des Arbeiters aktualisieren, damit die Arbeitszeiten, die bezüglich der Wünsche für diesen Auftrag gewährt werden, nicht mehr für andere Aufträge in Betracht gezogen werden dürfen. Das System muß jedoch in der Lage sein, bei einem neuen Auftrag, der zu erfüllen wäre, wenn intern der Plan umgestellt werden würde, dies erstens zu erkennen und zweitens auch diese Umplanung durchzuführen und dem Arbeiter dies über die Benutzeroberfläche mitzuteilen.

Wie man sieht, soll es dem Arbeiter nicht möglich sein, in ein laufendes Protokoll eingreifen zu können. Dieser Aspekt ist gewünscht, da ein Eingreifen oder gar die Ausführung der Kommunikation durch den real existierenden Arbeiter den gesamten Prozeß bezüglich eines Arbeiters extrem verlangsamen würde. Seitens des Disponenten ist dies analog. Hier liegt auch einer der Perfomanzaspekte des gesamten Systems. Reflektiert man die Geschäftswelt, so kann diese Prüfung innerhalb eines Unternehmens den gesamten Ablauf, also die Bestätigung des Disponenten an seinen Auftraggeber erheblich verzögern.

Betrachten wir kurz den Aspekt von Multiagentensystemen, als welches das Softwarepaket implementiert werden soll. Einen der Hauptargumente für ein solches System erfüllen wir mit dieser Aufteilung der einzelnen Aufgabenbereiche, nämlich den Punkt der Autonomie. Der Disponent kann von dem eintreffenden Auftrag lediglich mit dem Typ und der Menge direkt arbeiten, den Abgabetermin gibt er innerhalb des Protokolls an die Arbeiter weiter in Form einer Anfrage, wieviel die Arbeiter bis zu diesem Termin im Stande sind, zu produzieren. Auch die Entscheidung über die Vergabe trifft der Disponent ohne weitere Vorgaben von außen. Der Arbeiter kann nichts mit dem Typ und der Menge des potentiellen Auftrags anfangen. Er kann lediglich angeben, wieviel er unter welchen Wunschaspekten produzieren könnte bis zu diesem geforderten Abgabetermin. Darüberhinaus muß der Arbeiter seine Präferenzen auch gar nicht offenlegen.

4.5 Produktdaten und -leistungen

Hier sollen die vom System langfristig zu speichernden Daten erläutert werden. Gerade in Systemen, in denen auch Datenbankzugriffe und deren Änderung einen Hauptbestandteil der Arbeit am System ausmachen, ist dieser Aspekt wichtig für den Entwurf derselben. In unserem System gibt es nur wenige Daten, die gespeichert werden müs-

sen und diese auch nicht unbedingt sehr langfristig. Im Disponenten als auch im Arbeiter sind dies die eingeplanten Aufträge. Diese können und sollen vom System auch gelöscht werden, nämlich wenn der Abgabetermin verstrichen und der Auftrag somit vollständig beendet wurde. Diese Daten und deren Umfang hängen letztlich auch von der Menge der am elektronischen Markt partizipierenden Agenten ab. Desweiteren hängt dieser Datenumfang auch von dem betrachteten Einplanungszeitraum ab, wenn dieser weit in die Zukunft reicht und entsprechend viele Aufträge verplant werden, kann man sich unter Umständen Gedanken über die Datenstruktur zur Speicherung dieser Menge machen, aber dieser Punkt sollte im Rahmen dieser Diplomarbeit nicht weiter verfolgt werden, weswegen er hier auch nur aus Gründen der Vollständigkeit erwähnt wird.

Unter Produktleistungen versteht man alle Anforderungen, die zeitbezogen oder umfangsbezogen sind. Dies umfassen die Antwortzeiten in den Dialogen, die maximale Rechenzeit für bestimmte Algorithmen sowie Datenumfang oder -durchsatz bei plattformübergreifenden Aktionen. Auch dieser Punkt ist bei einem Softwareunternehmen durch den Kunden klar vorgegeben, hier existiert dieser konkrete Auftraggeber in dem Sinne nicht. Trotzdem stellen wir hier gewisse Anforderungen an das System. Aus Kapitel 2 geht hervor, dass die Komplexität der zu bearbeitenden Probleme NP-hart ist, jedoch sollte die Bearbeitungszeit des Systems auch im Hinblick der Größe der Testszenarien die Grenzen einhalten, die dem bisherigen Stand der Forschung entsprechen. Der Simplexalgorithmus zum Beispiel hat zwar auch eine exponentielle Laufzeit, bei einer Anzahl von Variablen und Constraints, die nicht über 1000 steigt, hat er aber immer noch eine Antwortzeit im Sekundenbereich natürlich in Abhängigkeit der betrachteten Implementierung.

Bezüglich der umfangsbezogenen Anforderungen bleibt anzumerken, dass die Daten, welche im Verlauf eines Szenarios, wie wir es in diesem und im vorangegangenen Kapitel erläutert haben, kritische Bereiche bei weitem nicht erreicht. Die Daten, die im Verlauf eines Protokolls zwischen Arbeiter und Disponent versendet werden, sollten von jeder Agentenplattform ohne jedes Problem bearbeitet werden können.

4.6 Benutzeroberfläche

In diesem Teilabschnitt wollen wir kurz auf die schon erwähnte Benutzeroberfläche mit ihrer Funktionalität eingehen.

Für den Disponenten soll diese ihm anzeigen, welche Aufträge er wann angenommen hat und welchen Arbeitern er dabei wieviel mit welchen Wünschen zugeteilt hat. Dies bedeutet, daß nach jedem abgeschlossenen Protokoll diese GUI aktualisiert werden muß. Zudem soll ihm die Möglichkeit geboten werden, einen neuen Auftrag über sie zu starten, also einen Auftrag, den er aus dem elektronischen Markt erhalten hat mit seinen spezifischen Daten einzugeben und zu bestätigen, womit er ein neues Protokoll startet. Diesen Vorgang soll er unabhängig von einem laufenden Protokoll, also jederzeit, vollziehen können. Bereits erledigte Aufträge, deren Abgabetermin also bereits abgelaufen ist, sollen aus dieser GUI gelöscht werden.

Die Benutzeroberfläche für den Arbeiter soll folgende Funktionalität aufweisen. Der Arbeiter soll die Möglichkeit haben, über einen Kalender seine täglichen Arbeitsstunden mit Präferenzen belegen zu können. Diesen Kalender, in dem dann auch immer die aktuellen Wünsche und zu erledigenden Arbeiten enthalten sind, soll er jederzeit öffnen können, um seine neuen Wünsche zu formulieren oder alte zu ändern. Er soll somit zum Beispiel auch die Möglichkeit haben, den ihm zustehenden Urlaub schon frühzeitig zu planen und mit entsprechender Wichtigkeit zu belegen. In diesem Kalender auch der aktuelle Plan des Arbeiters, also seine Auftragslage enthalten. Dieser Kalender bildet also gewissermaßen die Schnittstelle zwischen dem menschlichem Arbeiter und dem System, welches ihn im elektronischen Markt repräsentiert.

Kapitel 5

Implementierung und Evaluierung

In diesem Kapitel soll die Frage beantwortet werden, ob und wie wir die in der Spezifikation erarbeiteten Aspekte realisiert haben. Zudem soll anhand verschiedener Testszenarien systematisch gezeigt werden, wie leistungsfähig das gesamte System ist.

Hierfür werden wir im ersten Teilabschnitt ein System auswählen, welches in der Lage ist, Constraint-Hierarchien effizient zu lösen, und welches darüberhinaus auch noch anderen entwicklungsspezifischen Entscheidungskriterien genügt. Im zweiten Abschnitt des Kapitels werden wir dann einen Überblick über die verwendete Agentenplattform geben. Auch hier werden wir uns auf die die Implementierung betreffenden Aspekte konzentrieren. Im dritten Abschnitt erfolgt die eigentliche Lösung des Problems. Hier wird beschrieben, wie die sich im Verlauf der Arbeit herauskristallisierten Kernprobleme gelöst und implementiert wurden. Dabei werden die wichtigsten Klassen mit ihren Besonderheiten vorgestellt. Hiermit endet der erste Hauptaspekt des Kapitels, nämlich der Beantwortung der Frage, wie das Problem gelöst wird.

Im vierten Teilabschnitt des Kapitels werden die Evaluierungskriterien erläutert. Es werden Testszenarien vorgestellt, die zum einen exemplarisch für die gesamte Problematik sind, welche aber andererseits ebenso das System bewußt an seine Grenzen führen soll, damit seine Leistungsfähigkeit tatsächlich belegt werden kann. Im folgenden und letzten Abschnitt des Kapitels werden dann die Ergebnisse aus diesen Testläufen geschildert und analysiert.

5.1 Begründete Wahl eines Solvers

In diesem Abschnitt wollen wir ein System auswählen, welches im Arbeiteragenten das *Soft Constraint Solving* erledigt. Da es etliche Systeme gibt, welche in der Lage sind, Constrainthierarchien effizient zu bearbeiten, brauchen wir im Rahmen dieser Arbeit keinen eigenen Solver zu kreieren, sondern es genügt, eines der bestehenden Systeme auszuwählen, welches unseren Anforderungen am ehesten nachkommt, um es zu integrieren.

Hierfür werden wir ausgehend von der Spezifikation diese Kriterien bestimmen und sie danach an den in Abschnitt 2.2.3 vorgestellten Systemen prüfen, um so eine begründete Entscheidung zu erlangen.

Generell erhält der Solver vom System, also von dem Bereich, welches den Arbeiteragenten repräsentiert, eine Constrainthierarchie als Eingabe. Diese beinhaltet die Präferenzen bezüglich der Arbeitsstunden, die der Arbeiter potentiell zur Verfügung stellt. Als Vorleistung muß das System die Constraints konstruieren, welche das System erkennen kann. Diese Constraints beziehen sich aufgrund des Abgabetermins, welcher vorgegeben ist, und dem somit entstehenden Zeitintervall also auf eine beschränkte Menge von Arbeitsstunden, die gewichtet sind. Um zu prüfen, wieviel der Arbeiter unter Gewährung bestimmter Wünsche produzieren kann, erhält man also relativ einfache lineare Ausdrücke. Andere Formen von Constraints sind aus der Problemstellung heraus nicht nötig.

Die Tatsache, dass man nur lineare Gleichungen und Ungleichungen betrachten muß, grenzt die Wahl des Systems schon etwas ein. Gesucht ist also ein System, welches speziell bezüglich linearer Ausdrücke sehr effizient ist. Unter der Prämisse, dass das System den mit Präferenzen behafteten Kalender vor dem *Soft Constraint Solving* in eben diese Constraints transferiert, betrachten wir noch einmal jene Solver aus Abschnitt 2.2.3, welche im Hintergrund den Simplexalgorithmus benutzen. Dieser Algorithmus ist nämlich bezüglich Linearer Programme sehr effizient, sofern die Anzahl der Variablen und Bedingungen (Constraints) die Grenze von 1000 nicht übermäßig übersteigt. Diese Solver waren *DeltaStar* von Wilson und Borning (1993) sowie *Cassowary* von Badros und Borning (1998). Letztgenannter hat den Vorteil, dass er simultane Gleichungssysteme wirklich effizient bearbeiten kann. Desweiteren können bei *Cassowary* die Variablen Werte aus unbeschränkten Domänen annehmen, also auch etwa aus den natürlichen Zahlen $\mathbb{N}$ oder den reellen $\mathbb{R}$. Auch die anderen Systeme können obige Problematik bearbeiten, jedoch liegt die Effizienz eines Systems, welches speziell für solche Gleichungssysteme entworfen wurde, doch um einiges höher als bei den anderen. Auch der Vorteil bezüglich der unbeschränkten Domänen bei *Cassowary* läßt diesen hier als den am meisten geeigneten Solver erscheinen.

Ein weiteres Kriterium liegt in der Gewichtung der Constraints. Prinzipiell könnte man bei allen Systemen die verschiedenartigsten Gewichtungsmöglichkeiten umsetzen, jedoch betrachten wir hierfür zunächst noch einmal unsere Ausgangslage. Dem Arbeiter soll erlaubt sein, seine Arbeitszeit mit Präferenzen zu belegen. Wenn man die Berufswelt reflektiert, so ist ein Großteil der Arbeitszeit jedoch vorgegeben. Dies würde hier dem Fall entsprechen, in dem der Arbeiter gar keine Angaben bezüglich einer bestimmten Stunde macht. Nun würde eine Klassifizierung, bei der er zwischen Wünschen, die ihm sehr wichtig, die ihm wichtig, aber nicht unbedingt notwendig und nicht sonderlich wichtigen Wünschen durchaus ausreichen, um seine tatsächliche Präferenzlage zum Ausdruck zu bringen.

Bei dieser Vorgabe würde sich wiederum *Cassowary* als *Soft Constraint Solver* anbieten, da er von vorne herein eine Klassifizierung zwischen *required, strong, medium* und *weak* (erforderlich, stark, mittel und schwach) bezüglich der Constraints unterstützt. Eine feinere Klassifizierung würde auch nicht unbedingt viel Sinn im Hinblick auf die Optimierung im Disponenten sowie auch der Bearbeitung im Arbeiter machen, da der Rechenaufwand und die Komplexität um ein Vielfaches steigen würde. Dieser

Aspekt wurde bereits in Kapitel 3 der Problemstellung in Abschnitt 3.2 über die speziellen Herausforderungen herausgestellt, der Suchraum des Disponenten hat somit die Größe $(|Prioritätsklassen| + 1)^{|Arbeiter|}$, wonach man versuchen sollte, eben diese Basis möglichst gering zu halten, ohne jedoch die Klassifizierungsmöglichkeiten für den Arbeiter allzu sehr einzuschränken.

Auch die Programmiersprache, in der das System auch frei erhältlich sein soll, bildet ein Kriterium bezüglich der Entscheidung für ein bestimmtes System. Da FIPA-OS auf Java-Technologie basiert und die anderen Arbeiten des Sozionik-Projektes ebenso in Java implementiert sind, sollte auch das System in Java verfügbar sein. Dieser Aspekt ist zwar nicht unabdingbar, jedoch erleichtert es die Implementierung ungemein, wenn man nicht noch auf Schnittstellenprogrammierung aufgrund von Sprachunterschieden eingehen muß. Von denen in Kapitel 2 erwähnten Systemen sind lediglich zwei in Java implementiert und frei erhältlich. *HiRise* von Hosobe (1997) sowie wiederum *Cassowary* sind diese Systeme. *Cassowary* bietet jedoch den Vorteil gegenüber *HiRise*, dass es dem Entwickler leichter gemacht wird, auf die Werte der einzelnen Variablen zuzugreifen, was im Hinblick auf die Berechnung der Zielfunktionswerte bezüglich einer bestimmten Menge von Constraints eine Erleichterung darstellt. Im Verlauf dieses Kapitels wird noch erklärt, wie man die jeweiligen Werte der Zielfunktion ermitteln kann.

Die vorangegangenen Aspekte lassen bei der Wahl für den Solver, der im Rahmen der Implementierung und Lösung des gestellten Problems verwendet wird, nur auf *Cassowary* schließen. Der Solver scheint abgesehen von dem erwähnten Nachteil bezüglich der Zielfunktion, dessen Lösung wir noch im Verlauf dieses Kapitels erklären werden, geradezu prädestiniert für die Verwendung und Realisierung in unserem System. Auf detaillierte Aspekte des Solvers wird während der Klassenbeschreibung noch genügend eingegangen.

5.2 FIPA-OS

Im Verlauf dieser Arbeit wurde schon mehrfach der Begriff des *Agenten* sowie *Multiagentensystem* benutzt. Hier wollen wir nun begründen, warum die Realisierung des gestellten Problems durch *Softwareagenten* in einem solchen *Multiagentensystem* am sinnvollsten ist.

Wie in den beiden vorangegangenen Kapiteln der Problemstellung und der Spezifikation erörtert, haben die beteiligten natürlichen Personen des Disponenten und des Arbeiters eine unterschiedliche Wissensbasis. Beide sind jedoch zur Lösung des gesamten Problems unabdingbar eben gerade mit diesem unterschiedlichen Wissen und Fähigkeiten. Der Disponent hat keine Kenntnisse über die Kapazitäten der Maschinen und die Wünsche der Arbeiter. Umgekehrt weiß der Arbeiter nicht, wie hoch die Gesamtkapazität des Auftrags ist und mit wievielen Arbeitern der Disponent noch in Kontakt steht. Gerade im Hinblick auf die Wünsche der Arbeiter ist es sogar erwünscht, dass der Disponent und Arbeiter disjunkte Wissensbasen haben.

Desweiteren können sie rein vom räumlichen Aspekt her gesehen voneinander getrennt sein, durch ihr nötiges Zusammenwirken wird also ein hohes Maß an Kommunikation von Nöten sein. Nun sind die Akteure auch nicht unbedingt fest aneinander gekoppelt, sondern sie können ihre Arbeitsabläufe immer wieder mit neuen und anderen Personen durchführen. Beim Disponenten kommt dies nicht zuletzt alleine schon durch die unterschiedlichen Aufträge zustande, welche er bearbeiten muß und welche dabei nur von unterschiedlichen Arbeitern erfüllt werden können. Die Arbeiter wiederum können von mehreren verschiedenen Disponenten angesprochen werden, die potentielle Aufträge an sie erteilen können. Das System sollte somit auch recht flexibel sein. Sämtlich genannte Anforderungen, welche den korrekten Arbeitsablauf möglich machen, werden von bestimmten Agentenstrukturen sowie Multiagentensystemen mit ihren Plattformen und unterstützten Funktionalitäten erfüllt. Der Schluß, die gestellte Aufgabe mit seinen einhergehenden Problemen in Form von *Softwareagenten* auf einer geeigneten Agentenplattform zu realisieren, ist somit eine logische Konsequenz.

Nachdem in Kapitel 2 schon bezüglich Agententechnologien auf die FIPA-Standards hingewiesen wurde, ist es klar, dass es auch verschiedene Implementierungen für diese Standards gibt. Das Softwareunternehmen Emorphia, welches sich aus der Firma Nortel Networks aus Kanada heraus entwickelte, bietet mit seinem System FIPA-OS, welches sehr weit verbreitet ist, ein solches Agentenwerkzeug. FIPA-OS, das ”OS” steht hierbei für *open source*, was bedeutet, dass das System frei erhältlich, einsetzbar und modifizierbar ist, bildet auch in dem Projekt ”Sozionik“ die verwendete Agentenplattform. Es wird aber auch in mehreren anderen Projekten am DFKI in der Forschungsabteilung der Multiagentensysteme wie etwa in CASA oder SAID benutzt. Wie bereits in Kapitel 2 erläutert, bietet ein solches Agentenwerkzeug gerade im Bezug auf die Kommunikation zwischen den Agenten eine große Hilfe, da man viele bekannte und bewährte Protokolle, welche vom System direkt unterstützt werden, relativ einfach umsetzen kann.

Das auf Java-Technologien basierende FIPA-OS bietet dem Entwickler einen einfachen Zugriff auf verschiedene Übertragungsprotokolle wie RMI, HTTP oder CORBA, welche schon in dieser Arbeit (Abschnitt 2.1.3) erklärt wurden. Ebenso werden Parser für Datenformate wie XML von FIPA-OS zur Verfügung gestellt. Der Aufbau eines FIPA-OS-Agenten wird ebenfalls vom System durch eine ausführliche Dokumentation erleichtert [Guide, 2001]. Die Version 2.1 des Systems wurde als Grundlage in dieser Arbeit verwendet. Im vorangegangenen Kapitel der Spezifikation sowie in Abschnitt 2.1.3 über den FIPA-Standard wurde schon mehrfach auf die nötigen Dialoge zwischen den Agenten eingegangen. Diese Dialoge innerhalb vordefinierter Protokolle abgearbeitet, wobei diese in FIPA-OS in sogenannte *Tasks* umgesetzt werden. Diese *Tasks* sind normalerweise protokollgebunden für den am Dialog teilnehmenden Agenten verantwortlich. Für ihre Verwaltung besitzt jeder Agent einen *Taskmanager*, der für den korrekten Ablauf verantwortlich ist und die eintreffenden Nachrichten den korrespondierenden Tasks zuordnet.

Prinzipiell werden alle Agenten, welche nicht direkt beim Hochfahren des Systems gestartet werden, auf die gleiche Art und Weise auf der Plattform angemeldet. Über eine graphische Benutzerschnittstelle (Abbildung 5.1), welche von FIPA-OS zur Verfügung gestellt ist, kann man den Agenten starten, worauf diese sich dann bei dem bereits in Kapitel 2 beschriebenen *Directory Facilitator* und dem *Agent Management System*, welche ebenfalls als Agenten auf der Plattform realisiert sind, mit ihren speziellen Eigenschaften und ihren Adressen anmelden.

Abbildung 5.1: FIPA-OS *Agent-Loader*

5.3 Problemlösung

Kommen wir nun zu der Lösung und Implementierung der in der Spezifikation herausgestellten Anforderungen an das System. Im ersten Teil werden wir die speziellen Herausforderungen, welche sich für den Disponenten im Verlauf des erörterten Szenarios bilden können, nochmals kurz auffassen, um dann deren Lösung zu beschreiben. Im zweiten Teilabschnitt werden wir dies analog für die im Arbeiter anstehenden Aufgaben tun. Die Aufteilung in eben diese Agenten ist schon von der Ausgangslage und den natürlich existierenden Personen her klar, so dass wir diese hier nicht mehr weiter beschreiben.

5.3.1 Disponenten-Agent

Die erste Methode, welche in der Spezifikation beschrieben wurde, läßt sich dank der verwendeten Plattform FIPA-OS recht einfach lösen. Die Aufgabe des Systems besteht darin, anhand eines neuen Auftrags und den einhergehenden Eingabedaten, dem Typ, der Kapazität sowie dem Abgabetermin zu bestimmen, welche Arbeiter diesen Auftrag überhaupt bearbeiten können. Hierfür ist eigentlich nur der Typ von Interesse, da man in Abhängigkeit seines Wertes mit Hilfe von FIPA-OS diese Arbeiter-Agenten bestimmen kann. Auf der Agentenplattform befinden sich zwei vorgegebene Agenten, der sogenannte *Directory Facilitator* (DF) sowie das *Agent Management System* (AMS),

bei denen sich jeder Agent mit seinen spezifischen Eigenschaften bei ”Betreten” der Plattform anmelden muß. Nun startet man eine sogenannte *DF-Search* nach dem geforderten Typ des Auftrags und erhält als Ergebnis die Menge der Arbeiter, die in der Lage sind, diesen Typ zu produzieren.

Diese Methode befindet sich in der eigentlichen ”Hauptklasse” des Disponenten. Diese ist jene Klasse, von der alle übrigen den Disponenten betreffenden Klassen abgeleitet sind. So auch die *DisponentAgentContractNet*, von der eine Instanz nach einer erfolgreichen oben beschriebenen Suche gebildet wird. Der Taskmanager dieser Klasse startet dann ein *Contract Net Protocol* [Smith, 1979] mit jedem Agenten aus dieser Menge, indem er ihnen ein sogenanntes *call for proposal* schickt, wobei er in dieser Nachricht auch den für den Arbeiter notwendigen Abgabetermin mitsendet.

Im nächsten Schritt des *Contract Net Protocols* findet dann in der selben Klasse die eigentlich Hauptaufgabe des Disponenten statt. Er muß aus allen Geboten der Arbeiter die Kombination finden, welche zum einen den Auftrag erfüllt, sofern dies überhaupt möglich ist, und zum anderen muß er die Wünsche der Arbeiter, welche in den Geboten und den damit zusammenhängenden Kapazitäten seitens des Arbeiters zum Ausdruck gebracht sind, optimal einhalten.

Im folgenden soll die Lösung dieser beschriebenen Problematik erörtert werden. Der Disponent erhält also von jedem Arbeiter ein Gebot der Form:

$$GebotAgent_i = (x^i_{required}, x^i_{strong}, x^i_{medium}, x^i_{weak})$$

Hierbei ist $i \in \{1, \ldots, |Arbeiterkandidaten|\}$, wobei die Menge der Arbeiterkandidaten denen am CNP beteiligten Arbeitern entspricht, und die Kapazität $x_{required}$ der Menge von dem georderten Gut entspricht, die der Arbeiter i produzieren kann, wenn nur die Constraints, also Wünsche eingehalten wurden, die gar nicht gebrochen werden durften. Die Kapazität x_{strong} entspricht dann der Menge des Gutes, die er unter Einhaltung der *required* und der *strong* Constraints produzieren könnte, die anderen beiden setzen sich entsprechend fort. Diese Einteilung der Wünsche ergibt sich aus der in Abschnitt 5.2 getroffenen Wahl des hierarchischen Solvers, wobei wir diese Einteilung nochmals im Abschnitt bezüglich des Arbeiter-Agenten aufgreifen werden.

Hat der Disponent alle Gebote der Arbeiter erhalten, so generiert er sich eine *Antwortmatrix*, über welche er dann seine Optimierung durchführt. Der Aufbau dieser Matrix wird im folgenden beschrieben und ist in Abbildung 5.3 in dem dazugehörenden Beispiel nochmals veranschaulicht. Um das Einhalten von Präferenzen metrisch beurteilen zu können, haben wir *Strafpunkte* eingeführt, welche der Disponent ”bezahlen” muß, falls er gewisse Wünsche der Arbeiter brechen muß, um seinen Auftrag zu erfüllen. Ziel des Disponenten muß es dann sein, diese Strafpunkte zu minimieren aber unter der Bedingung der Einhaltung der Kapazität des Auftrags. Diese *Antwortmatrix* hat dann $n+1$ Spalten für n angesprochene Arbeiter und eine “Strafpunktspalte”, deren Einträge wir noch erklären werden. Desweiteren besteht diese Matrix aus fünf Zeilen, und zwar vier für die jeweiligen Kapazitäten und einer ”Nullzeile”. Diese Nulleinträge sind nötig für den Fall, dass ein Arbeiter trotz Gebot keinen Zuschlag des Disponenten erhält und folglich auch nichts zu produzieren hat. Trotzdem muß er innerhalb der

optimalen Lösung, welche dann aus den verschiedenen Zählerständen für jede Spalte besteht, enthalten sein. Der Zählerstand für jede Spalte gibt dann genau die Kapazität an, welche der dieser Spalte entsprechende Arbeiter zu produzieren hat.

Um eine Lösung, bei der viele Arbeiter relativ geringe Kapazitäten produzieren, jedoch unter Gewährung aller ihrer Präferenzen, einer anderen möglichen Kombination vorzuziehen, bei der ein Arbeiter den gesamten Auftrag alleine erledigen muß, dies jedoch aber nur schaffen kann, wenn ihm ein Wunsch nicht gewährt wird, müssen die Strafpunkte entsprechend definiert werden. Es muß also sichergestellt sein, dass eine Lösung mit geringerer Strafpunktzahl unabhängig von ihrer Zusammensetzung einer anderen Lösung mit höherer Strafe vorgezogen wird, auch wenn die zweite von der produzierten Menge her besser ist, wobei die erstgenannte natürlich den Auftragsumfang immer noch erfüllen muß. Die Strafpunkte *p* seien also wie folgt definiert:

$$\mathbf{p} = \begin{cases} 0 & für \quad Zeile1 \\ 1 & für \quad Zeile2 \\ \text{n+1} & für \quad Zeile3 \\ \text{n(n+1)+1} & für \quad Zeile4 \\ \text{n(n(n+1))+1} & für \quad Zeile5 \end{cases}$$

Hierbei entspricht n der Anzahl der Arbeiter-Agenten in diesem Auftrag. Die Antwortmatrix sieht dann wie folgt aus:

Strafpunkte	A_1	A_2	$\cdots$	A_n
0	0	0	$\cdots$	0
1	x^1_{weak}	x^2_{weak}	$\cdots$	x^n_{weak}
$n+1$	x^1_{medium}	x^2_{medium}	$\cdots$	x^n_{medium}
$n*(n+1)+1$	x^1_{strong}	x^1_{strong}	$\cdots$	x^n_{strong}
$n*(n*(n+1))+1$	$x^1_{required}$	$x^2_{required}$	$\cdots$	$x^n_{required}$

Die Datenstruktur dieser Matrix (zweidimensionales Array) dient als Grundlage der danach folgenden Tiefensuche. Es ist nicht schwer einzusehen, dass der Suchraum dabei in Abhängigkeit der Anzahl der Agenten steht und exponentiell wächst ($5^{|Arbeiter|}$). Jedoch kann man einen großen Bereich dieses Raums durch geschickte Verfahren sehr schnell reduzieren. So wird eine gewisse Art des Branch&Cut-Algorithmus implementiert, der wie folgt funktioniert. Man hält immer eine aktuell beste Lösung, welche man mit allen neu generierten vergleicht. Ein Blatt wird dabei nur dann potentielle Lösung, wenn seine Kapazität für den Auftrag ausreicht. Jedoch wird schon während des Durchlaufens eines Pfades an jedem Knoten, also bei jedem Arbeiter, die momentan für diese Teillösung aktuelle Strafpunktanzahl berechnet. Übersteigt diese der Strafpunktanzahl der aktuell besten Lösung, so wird der Durchlauf abgebrochen und der gesamte Unterbaum dieser Zwischenlösung kann abgeschnitten werden, da sich die Strafpunktzahl nur noch verschlechtern kann, also keine bessere Lösung in diesem Unterbaum mehr liegen kann. Hat eine neu generierte Lösung eine geringere Strafpunktanzahl, so wird sie neues lokales Optimum. Bei gleicher Strafe wird die Lösung

nur dann getauscht, wenn die produzierte Kapazität näher an der geforderten liegt. Die Reihenfolge der in dem Baum betrachteten Arbeiter hängt von der Reihenfolge der beim Disponenten eintreffenden Gebote ab. Der Arbeiter, dessen Gebot am schnellsten eintrifft, wird demnach Wurzel, die anderen folgen entsprechend.

Allgemein läßt man zwei Zähler über die Matrix laufen, merkt sich dabei jeweils ihre Zählerstände und berechnet dabei noch die Gesamtstrafpunktanzahl. Diese berechnet sich wie folgt:

$$\sum_{i=1}^{n}\sum_{j=1}^{4}\frac{Antwortmatrix[0][j] * Antwortmatrix[i][j]}{Antwortmatrix[i][j]} * b_j$$

Hierbei stehen i und j für die beiden Zähler über die Matrix, b_j ist eine Boolesche Variable, die genau dort den Wert 1 annimmt, an der der Zähler j über die Zeilen einer Spalte in dieser gehaltenen Lösung für den jeweiligen Arbeiter steht. Schließlich wird über alle Arbeiter summiert, womit man exakt die Strafpunkte erhält, welche der Disponent für das Brechen von Wünschen eines jeden Arbeiters bezahlen muß. Der zweite Zähler läßt dabei die erste Zeile bewußt aus, da in ihr lediglich die Arbeiter enthalten sind, wenn der Zähler dort auch steht, die nichts zu produzieren haben und demnach auch keine Strafpunkte verursachen. Desweiteren muß gewährleistet sein, dass man jeden Arbeiter tatsächlich nur einmal in Betracht zieht, was man erreicht, wenn man die Summe der b_j bezüglich jeder Spalte ≤ 1 hält.

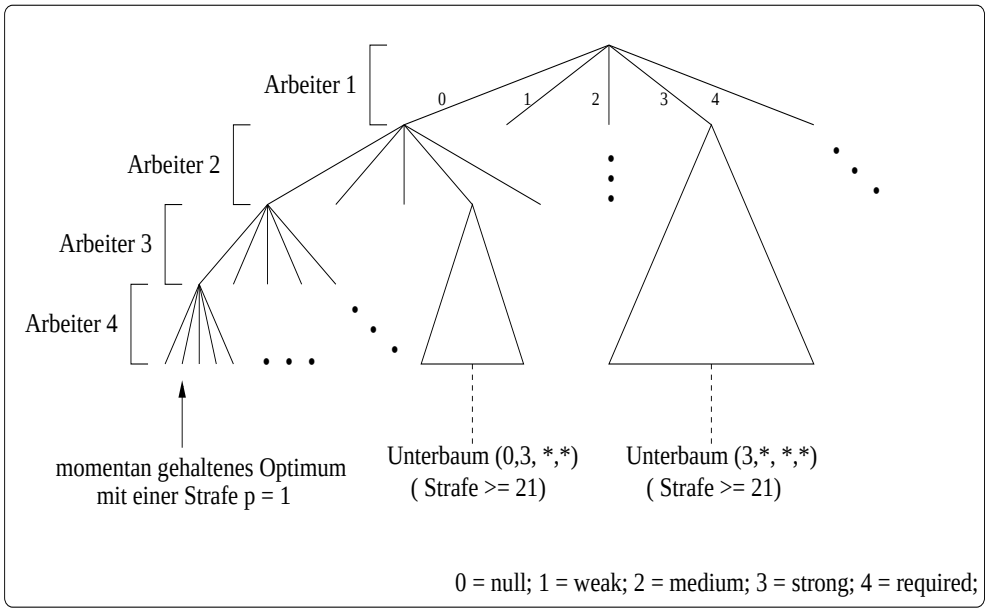

Abbildung 5.2: Optimierung im Disponenten

Abbildung 5.2 soll diese Vorgehensweise nochmals graphisch darstellen. In diesem Fall sucht der Disponent unter vier Agenten das Optimum bezüglich der Strafpunkte, also der Präferenzen der Arbeiter, unter der Voraussetzung, dass sein Auftrag noch erfüllt wird. Der Disponent hält eine momentane Lösung, bei der die Arbeiter 1 bis 3

nichts produzieren müssen, Arbeiter 4 unter Gewährung aller Wünsche die geforderte Kapazität alleine herstellen kann. Die Strafpunkte bezüglich dieser Lösung betragen nach obiger Definition demnach ”1”. Der Unterbaum (0,3,*,*), bei dem Arbeiter 1 nichts produziert, Arbeiter 2 nur seine *strong*-Wünsche gewährt bekommt sowie Arbeiter 3 und 4 noch nicht feststehen, kann abgeschnitten werden, da schon hier die Strafpunkte mit momentan ”21” über der aktuell besten Lösung liegen. Da sich diese für alle in diesem Unterbaum befindlichen Blätter nicht mehr verringern kann, bricht man die Berechnungen hier ab, und beginnt mit der nächsten Schleife. Gleiches gilt für den in Abbildung 5.2 eingezeichneten Unterbaum (3,*,*,*).

Nun ist es möglich, dass man so recht lange braucht,bis man zur ersten Lösung gelangt. Dies kann der Fall sein, wenn die geforderte Gesamtkapazität so hoch ist, dass man alle Arbeiter zur Erfüllung benötigt und diesen zudem noch fast alle Wünsche verwehren muß. Ist die Gesamtkapazität des Auftrags sehr hoch, so vergleicht man diese mit den Einträgen der Matrix in der letzten Zeile und setzt die Zähler dann gegebenenfalls zu Beginn auch hoch und zählt rückwärts. Analog macht man dies bei Aufträgen, die nahe der Summe der zweiten Zeile der Matrix sind, die erste Zeile enthält die Nulleinträge. Mit dieser Heuristik gelangt man recht schnell zu einer ersten kompletten Lösung, die als ”lokales Minimum” beim Abschneiden von Teilbäumen dient. Um in dem Fall der hohen Auftragskapazitäten ebenfalls Teilbäume abschneiden zu können, kann man nun jedoch keine guten Verbesserungen durch den Vergleich der Strafpunkte erzielen, da sie sowieso recht hoch sein werden. Man muß demnach versuchen, die in der Abbildung relativ weit *links* liegenden Teilbäume streichen zu können. Geometrisch betrachtet liegen in diesem Bereich des Baumes die Lösungen, bei denen die Arbeiter noch recht viele Wünsche gewährt bekommen, was jedoch bei einer extrem hohen Gesamtkapazität nicht mehr zu erwarten ist. Man vergleicht daher neben der aktuellen Strafe auch die bisher erreichte Kapazität und prüft, ob man die geforderte Gesamtmenge mit den restlich zur Verfügung stehenden Arbeitern und deren Maximalmenge überhaupt noch erreichen kann. Ist dies nicht der Fall, so kann man diesen Teilbaum abschneiden.

Nun ist es trotzdem möglich, dass es bei entsprechend hoher Arbeiteranzahl zu lange dauern würde, bis die optimale Lösung berechnet wird. Aus diesem Grund haben wir eine Zeitschranke für die Optimierung festgelegt, durch die eine laufende Optimierung abgebrochen wird, falls sie über diese Zeitschranke hinweg noch Rechenaufwand betreiben müßte. Die Optimierungsphase wird dann abgebrochen und das momentan gehaltene lokale Optimum als Lösung genommen.

Mit dieser Art der Optimierung und Vergabe der Aufträge erreicht man zudem eine sehr gute Verteilung der Aufgaben auf die einzelnen Maschinen, welche die Arbeiter bedienen. Man kann davon ausgehen, dass ein Arbeiter, dessen Auftragslage sehr hoch ist und er somit generell nur noch wenig Restkapazität für neue Aufträge zur Verfügung hat, auch sehr geringe Gebote bezüglich neuer Aufträge macht. Diese verringern sich um so mehr, wenn er diese restlich verbleibende Arbeitszeit zudem auch noch mit Präferenzen belegt. Umgekehrt kann ein Arbeiter mit geringer Auslastung höhere Gebote machen und diese zudem auch noch mit sämtlichen Präferenzen belegen. Demnach werden bei neuen Aufträgen zuerst die Arbeiter die Zuschläge erhalten, deren Auslastung im Vergleich zu denen der anderen Arbeiter geringer ist. Dieser Effekt

entspricht genau dem in Abschnitt 2.3.2 vorgestellten einfachen Approximierungsalgorithmus, der eine recht gute Schranke bezüglich der optimalen Lösung im Bezug auf das Planungsproblem unterbietet.

Innerhalb der oben beschriebenen Methode werden also zwei Ziele verfolgt. Das Planungsproblem wird mit der eben erwähnten oberen Schranke erfüllt, die Gewährung der Wünsche der Arbeiter optimal. Es bleibt anzumerken, dass man die Strafpunktzahl einer Lösung nur innerhalb eines Auftrags mit der einer anderen Lösung vergleichen kann. Global, also bezüglich verschiedener Aufträge, kann man diese nicht zum Vergleich ziehen, da die Strafpunkte von der Anzahl der Arbeiter, die für den Auftrag in Frage kommen, der Kapazität und dem Zeitpunkt des Einplanens abhängen. Ist die Anzahl der Arbeiter von Auftrag zu Auftrag verschieden, so sind die potentiellen Strafpunkte nach Definition sowieso unterschiedlich, so dass ein Vergleich unmöglich ist. Bei gleicher Arbeiteranzahl wird ein Auftrag mit hoher Kapazität in Bezug auf die Gesamtleistung der Maschinen wird potentiell mehr Strafpunkte "kosten" als einer mit kleiner Kapazität, zudem wenn er noch zu einem Zeitpunkt eingeplant werden soll, wenn die Auslastung aller Arbeiter schon sehr hoch ist. Auch hier ist somit ein Vergleich nicht möglich. Nachstehender Pseudocode soll die eben beschriebene Optimierung im Disponenten nochmals zusammenfassen. Die beiden Abschneidekriterien

Algorithm 3 Pseudocode für die Optimierung

```
 1: procedure optimize(matrix)
 2:  optimum ← null;
 3:  while (run = true)
 4:     for all (workers w ∈ matrix)
 5:        for all (rows r ∈ matrix)
 6:           punishment ← punishment + matrix[0][r];
 7:           capacity ← capacity + matrix[w][r];
 8:           if (prune criterion 1 = true) then break;
 9:           if (prune criterion 2 = true) then break;
10:           currentmin ← [punishment; capacity];
11:           if (capacity ≥ ordercapacity) then
12:              if (currentmin is better than optimum) then
13:                 optimum ← currentmin;
14:     run ← false;
15: end optimize(matrix)
```

in dem Code entsprechen den beiden Bedingungen zum Abschneiden von Unterbäumen in der oben beschriebenen Optimierung. Die erste "Abschneidebedingung" ist die bezüglich der aktuellen Strafpunkte, ein Unterbaum kann demnach komplett abgeschnitten werden, wenn die aktuelle Strafpunktanzahl an diesem Knoten / Arbeiter die eines bereits gefundenen lokalen Minimums übersteigt. Die zweite "Abschneidebedingung" ist die bezüglich der noch erreichbaren Kapazität. Man prüft hierbei in einem Knoten, ob die bisher erreichte Kapazität in dieser Teillösung zusammen mit der noch maximal erreichbaren Kapazität über die anderen noch zur Verfügung stehenden Ar-

beiter ausreicht, um die geforderte Gesamtmenge zu erreichen. Da diese Bedingungen bereits ausführlich beschrieben sind, verzichten wir hier auf ihren Pseudocode.

Die Bedingung *currentmin is better than optimum* in Zeile 12 des obenstehenden Codes bedeutet, dass man die gehaltene Lösung nur dann zu einem neuen Optimum macht, wenn deren Strafpunkte geringer sind als die des gehaltenen Optimums, oder wenn ihre Strafpunkte gleich sind, die Kapazität des *currentmin* jedoch näher an der geforderten Gesamtmenge liegt.

Beispiel

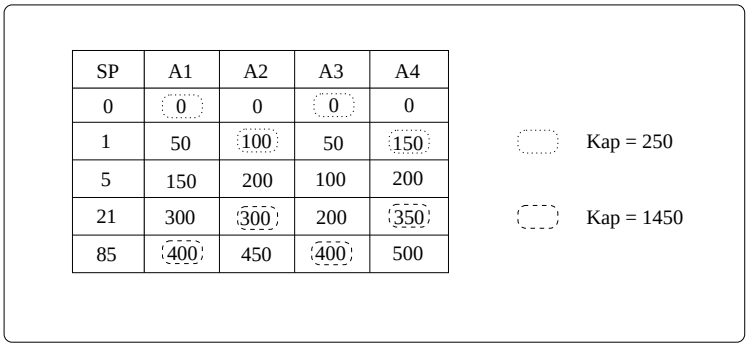

Abbildung 5.3: Antwortmatrix mit zwei Beispiellösungen

In den in Abbildung 5.3 dargestellten Beispielen hat der Disponent ein laufendes *Contract Net Protocol* mit vier Arbeitern. Wir betrachten zwei Fälle mit verschiedenen Kapazitäten, an denen wir die allgemeine Vorgehensweise nochmals verdeutlichen wollen. Zum einen soll der Disponent insgesamt 250 Stück von dem geforderten Gut herstellen lassen, wobei wir das Datum hier außer Acht lassen wollen. Wie man leicht sieht, ist die eingezeichnete Lösung auch die optimale für diese Kapazität. Die zu entrichtende Strafe für den Disponenten beträgt hierbei 2. Im anderen Fall sollen 1450 Stück hergestellt werden. Auch hier kann man die Lösung schnell verifizieren und erhält dabei 212 zu zahlende Strafpunkte.

Man sieht auch schnell ein, dass man durch die bezüglich der geforderten Gesamtmenge in Relation zur insgesamt produzierbaren Kapazität gewählten Startzählerstände schneller die optimale Lösung erhält. Im ersten Fall sollte man die Zähler anfangs auf “0” setzten, da man so sehr früh eine gute Lösung erhält. Der Suchraum kann so auch sehr schnell “abgeschnitten” werden, da alle Lösungen und Zwischenlösungen, die über das lokale Minimum von 2 hinaus gehen, abgebrochen werden und zudem alle Folgelösungen, die diesen Zwischenzählerstand mit beinhalten, ebenfalls sofort mit abgeschnitten werden. Umgekehrt sollte man beim zweiten Fall von “hinten” rückwärts zählen, da man so sehr schnell erkennen kann, dass viele Teillösungen die geforderte Kapazität gar nicht mehr erreichen können und somit wiederum abgeschnitten werden können.

Nach Beendigung dieser Optimierung befindet sich der Disponent immer noch im *Contract Net Protocol*, in welchem er im nächsten Schritt die Gebote der Arbeiter akzeptieren oder ablehnen muß. Obige Optimierung liefert ihm hierfür die Grundlage. Es muß lediglich das Array, welches von obigem Task und deren Methode zurückgegeben wird, durchlaufen werden und den entsprechenden Arbeiter-Agenten in Abhängigkeit des Wertes an der Stelle des Arrays ein Zuschlag (*accept-proposal*) oder eine Verwehrung (*reject-proposal*) geschickt werden.

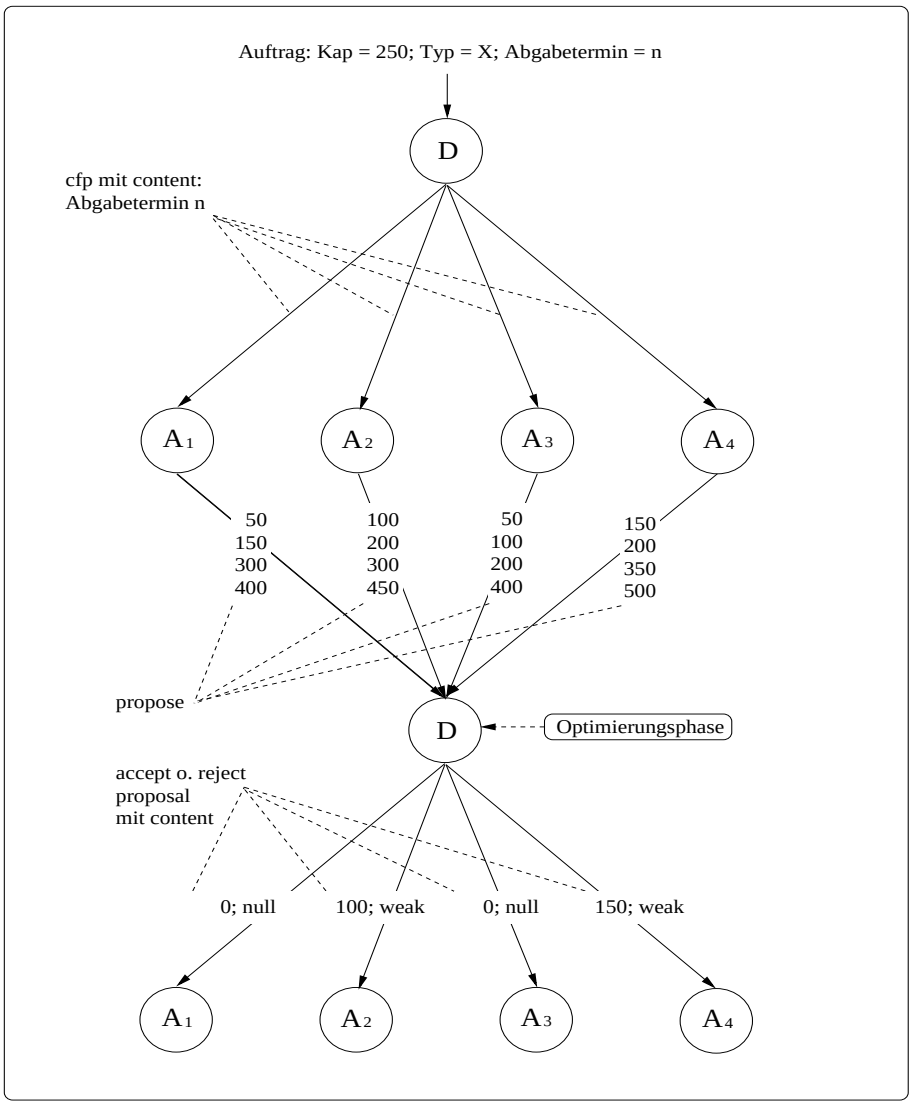

Abbildung 5.4: Ablauf eines Protokolls

Abbildung 5.4 soll einen solchen kompletten Durchlauf nochmals graphisch darstellen. Der Disponent ”D” erhält einen Auftrag, der 250 Stück von Typ *X* bis zum Abgabetermin *n* umfaßt. Typ und Abgabetermin können in diesem Beispiel vernachlässigt

werden, da die Visualisierung des Protokolls im Vordergrund steht und diese Daten den Ablauf nicht verändern. Nun sendet der Disponent, nachdem er über eine *DF-Search*, eine FIPA-OS-spezifische Methode, alle in Frage kommenden Arbeiter herausgefunden hat, ein sogenanntes *cfp (call for proposal)* an jeden dieser Arbeiter ("A_1" bis "A_4") und übergibt ihnen dabei den geforderten Abgabetermin. Die Arbeiteragenten berechnen nun mit Hilfe von *Cassowary* ihre Gebote und senden diese an den Disponenten zurück. In diesem Agenten beginnt dann nach Eintreffen aller Gebote die eben ausführlich beschriebene Optimierungsphase. Abbildung 5.4 und 5.3 können hierfür zusammen betrachtet werden, die Antwortmatrix aus Abbildung 5.3 entspricht genau der für die Optimierungsphase aus Abbildung 5.4 benötigte. Zum Abschluß sendet er seine Zuteilung an die jeweiligen Arbeiter zurück, welche die zu produzierende Menge sowie die Klasse der gewährten Wünsche beinhaltet. Nun senden die Arbeiter zum korrekten Abschluß des Protokolls noch eine Nachricht an den Disponenten zurück, dass ihr Protokoll erfolgreich beendet wurde, aus Gründen der Übersicht haben wir dies nicht in die Abbildung mit eingezeichnet.

Betrachtet man diesen Vorgang genau, so befinden wir uns hier nicht in einem reinen *Contract Net Protocol* nach Smith sondern eher in einer Modifikation desselben [Smith, 1979]. Normalerweise erhält im CNP lediglich ein Partizipant den Zuschlag vom Initiator, hier ist es dagegen möglich, dass mehrere oder gegebenenfalls sogar alle Agenten einen Teilzuschlag erhalten können, der zudem auch noch aus einer Menge von Geboten des Partizipanten (hier: Arbeiter) stammt. Auch muß der Zuschlag nicht exakt den Mengen der Gebote entsprechen, sondern es sind vielmehr Kapazitätszuweisungen aus den Intervallen, welche durch die Prioritäten entstehen.

Desweiteren gibt jeder Arbeiter nicht ein Gebot ab, sondern sein Gebot besteht aus einer Menge von vier potentiellen Produktionsmengen, welche er unter Gewährung gewisser Klassen von Wünschen im Stande ist, zu produzieren.

Auf die Implementierung der Benutzeroberfläche wird am Ende dieses Abschnitts zusammen mit der für den Arbeiter näher eingegangen.

5.3.2 Arbeiter-Agent

In diesem Abschnitt wird die Realisierung der in der Spezifikation erörterten Anforderungen an das Produkt in Bezug auf den Arbeiter beschrieben. Im Arbeiter-Agenten wird der zweite Hauptaspekt der gesamten Diplomarbeit, nämlich das Lösen von Constraint-Hierarchien umgesetzt. Eingangs dieses Kapitels haben wir mit der begründeten Wahl für ein existierendes System, welches solche Probleminstanzen lösen kann, schon eine wichtige Voraussetzung für diese Aufgabe geleistet.

Wie schon im vorangegangenen Abschnitt erläutert, wird das gesamte Szenario innerhalb eines festen Protokolls, dem *Contract Net Protocol*, basierend auf FIPA-Standards und in FIPA-OS implementiert, bearbeitet. Der Arbeiter ist Partizipant in diesem Protokoll und ist somit von dem Initiator, dem Disponenten, abhängig. Damit der Disponent aber überhaupt mit dem Arbeiter in Kontakt treten kann, muß sich dieser an der Agentenplattform mit seinen spezifischen Eigenschaften anmelden.

Bekommt der Arbeiter dann ein sogenanntes *call for proposal*, soll das System mit Hilfe von *Cassowary* seine Gebote berechnen und diese zurück an den Disponenten schicken. Diesen Ablauf wollen wir nun detailliert erklären. Das System benötigt zur Berechnung dieser Gebote zum einen den Abgabetermin des Auftrags und zum anderen die diesen Zeitraum betreffenden Constraints mit den entsprechenden Gewichtungen. Den Abgabetermin bekommt der Agent vom Disponenten mit übergeben, von gesteigertem Interesse sind hier die Constraints.

Grundlage für diese Constraints sind die Angaben des real existierenden Arbeiters. Er bekommt seitens des Systems über eine graphische Benutzerschnittstelle einen Kalender zur Verfügung gestellt, in welchen er seine Präferenzen bezüglich jeder Arbeitsstunde eintragen kann. Seine Arbeitszeit verringert sich potentiell mit jedem Wunsch, den er in diesen Kalender einträgt und welche folgendermaßen aussehen. Gewichtet er eine gewisse Stunde mit dem Wert “4”, so bedeutet dies, dass er zu diesem Zeitpunkt auf keinen Fall arbeiten kann. Die ”4” entspricht also dem von *Cassowary* unterstütztem *required*, die weiteren Zuordnungen sind: ”3” korreliert mit *strong*, ”2” mit *medium* und die ”1” mit *weak*. An den Stellen, an denen der Arbeiter keine Angaben im Kalender macht, steht eine ”0”, die besagt, dass diese Stunde von dem System frei eingeplant werden kann. Es ist hierbei wichtig anzumerken, dass es dem realen Arbeiter zu jedem Zeitpunkt möglich ist, dem System seine Wünsche mitzuteilen. Dieser Aspekt ist zum einen aus benutzerfreundlichen Gründen wichtig, aber zum anderen noch erheblicher im Bezug auf die Geschwindigkeit des gesamten Prozesses. Die Antwortzeiten würden bei jedem Auftrag und den hierauf bezogenen Wünschen des Arbeiters unnötig verlängert werden, wenn er sie erst nach Eintreffen einer Gebotsforderung formulieren könnte.

Das System betrachtet nun bei jedem Auftrag den Zeitraum vom Augenblick des eintreffenden *call for proposal* bis zu dem Abgabetermin. Hierfür wird der Kalender bis zu diesem Abgabetermin einmal durchlaufen, wobei die dann möglichen Arbeitsstunden bezüglich jeder Präferenzklasse bestimmt werden. Hieraus konstruieren wir die linearen Ungleichungen, welche wir *Cassowary* übergeben.

Nun haben wir schon erwähnt, dass Cassowary gar nicht direkt ein Gebot berechnet, sondern lediglich versucht, die Constraint-Hierarchie möglichst optimal zu erfüllen. Grundsätzlich werden hierbei vom Solver alle Constraints erfüllt, sofern sie nicht im Widerspruch zueinander stehen. Stehen zwei Constraints im Widerspruch zueinander, so wird derjenige gebrochen, der eine geringere Gewichtung hat. Haben beide die gleiche Stärke, so vergleicht man die direkt in Abhängigkeit stehenden Constraints der beiden auf die gleiche Art und Weise miteinander, was immer so forgeführt wird.

Die Berechnung des Zielfunktionswertes, welcher sich aus der Anzahl der zur Verfügung gestellten Stunden und der Kapazität der Maschine ergibt, muß also von außen erfolgen. Nun bietet *Cassowary* die Möglichkeit, auf die einzelnen Variablen von außen zuzugreifen und deren Werte auszulesen, womit wir nach jeder Iteration und dem Hinzufügen der nächsten Klasse von Wünschen ein Zwischenergebnis berechnen können. Dieses entspricht dann exakt der Menge, welche unter den bisher hinzugefügten Constraints produzierbar ist.

Ist die Menge der Gebote komplett, so wird diese in Form eines Strings an den Disponenten geschickt. Daraufhin wartet der Arbeiter innerhalb des Protokolls ab, wie der Disponent den Auftrag verteilt, was er wiederum in einer Nachricht übermittelt bekommt. In Abhängigkeit dieser Antwort, die aus der ihm zugeteilten Produktionsmenge und den gewährten Wünschen besteht, muß das System noch den Arbeitsplan des Arbeiters lokal aktualisieren. Dies macht man, indem man ausgehend vom frühsten Zeitpunkt der möglichen Einplanung alle die Stunden belegt, die überhaupt nicht mit Wünschen belegt wurden und demnach frei verplanbar sind. Danach nimmt man sukzessive die Stunden mit in den belegten Präferenzen in den Plan auf, die nicht mit einer größeren Gewichtung behaftet sind als dies vom Disponenten aus gewährt ist.

Eine weitere Anforderung an das System ist die potentielle Umplanung innerhalb eines Agenten. Dies ist nicht zu verwechseln mit dem Umplanen auf Agentenebene, was sehr schnell zu Verklemmungen (deadlocks) im System führen könnte, wie wir bereits in Kapitel 3 erörtert haben. Das System erkennt für den Arbeiter, ob er durch Verschieben und Umlegen von bereits eingeplanten Aufträgen für eine neue Anfrage ein besseres Gebot erzielen könnte. Folgendes Beispiel soll das Problem veranschaulichen.

Beispiel

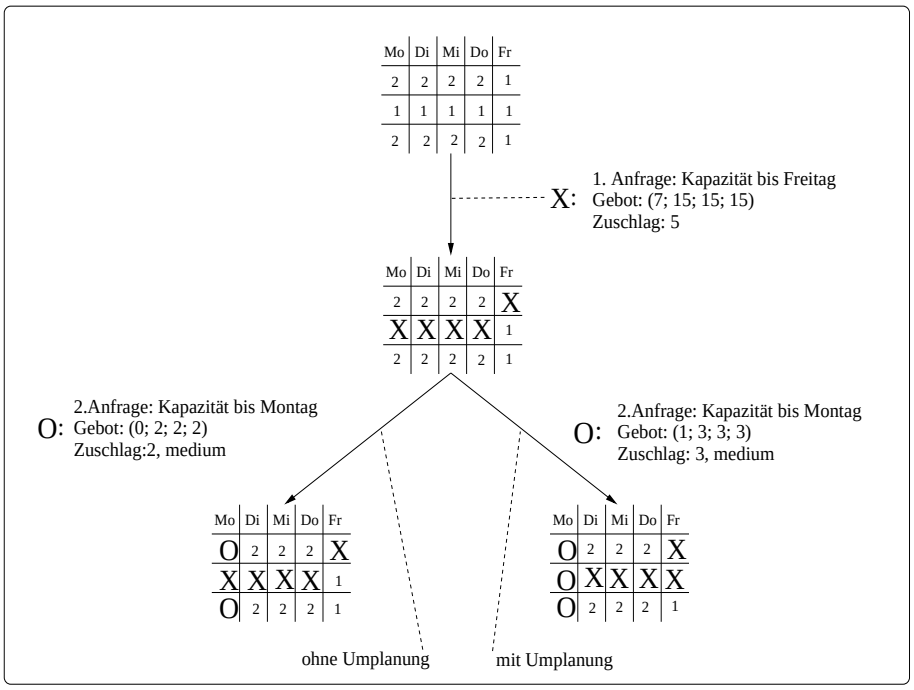

Abbildung 5.5: Beispiel für internes Umplanen

Wie die Abbildung 5.5 verdeutlichen soll, kann ein eingeplanter Auftrag, dessen Abgabetermin noch etwas weiter in der Zukunft liegt, das Gebot für einen neuen Auf-

trag mit kurzfristiger Deadline erheblich beeinträchtigen. Im Beispiel ist eingangs der Stundenplan des Arbeiters, der hier aus Veranschaulichungsgründen vereinfacht nur mit drei Stunden pro Tag dargestellt ist, mit Präferenzen belegt. Die ”1” entspricht *weak*, die ”2” entspricht *medium*. Im ersten Auftrag wird der Arbeiter gefragt, wieviel er bis Freitag herstellen kann, worauf er sein Gebot (7; 15; 15; 15) an den Disponenten zurück schickt. Er erhält einen Zuschlag von ”5” unter Gewährung der *weak*-Wünsche. Nun plant der Algorithmus die Stunden in den Plan des Arbeiters so ein, dass er immer die frühest mögliche Stunde, die er unter der gewährten Priorität findet, für diesen Auftrag blockiert. Das Ergebnis ist im zweiten Kalender der Abbildung 5.5 zu sehen. Danach trifft eine weitere Anfrage für den Montag ein. Die maximale Menge im Gebot des Arbeiters liegt im Fall ohne Umplanung bei lediglich zwei Stück, im Fall mit Umplanung jedoch bei drei Einheiten.

Beim Umplanen prüft das System, ob für jeden anderen Auftrag mit der gewährten Kapazität und dem dazugehörenden Abgabetermin noch genügend Spielraum vorhanden ist, um einzelne Stunden, die das Gebot des neuen Auftrags erhöhen würden, nach hinten zu schieben, ohne dabei jedoch den alten Auftrag mit seinen gewährten Prioritäten grundlegend zu verändern, abgesehen von den verschobenen Stunden bezüglich dieses Auftrags. Ohne Umplanung im Arbeiteragenten könnte dies im schlechten Fall, wenn er bei vielen Arbeitern gleichzeitig eintreten würde, zur kompletten Ablehnung von Aufträgen seitens des Disponenten führen, wobei der Auftrag jedoch ohne weiteres zu erledigen wäre, wenn man diese Umplanung im Arbeiter durchführt.
Dieses Umplanen sieht im obigen Beispiel recht einfach aus, kann aber bei einer Vielzahl von Aufträgen etliche Vergleiche mit sich ziehen, da dieses Umplanen ”nach hinten” einen kaskadierenden Effekt haben kann. Man muß hierfür für jeden einzelnen alten, also bereits eingeplanten Auftrag prüfen, sofern er überhaupt von der Umplanung betroffen sein könnte, ob die restliche zur Verfügung stehende Arbeitszeit für die umzuplanenden Stunden von dem Abgabetermin des neuen Auftrags aus gesehen ausreicht. Dieser kaskadierende Effekt hat jedoch spätestens bei dem am weitesten in der Zukunft liegenden Auftrag ein Ende mit einer eindeutigen Antwort. Der Rechenaufwand dieser Methode hängt also ab von der Anzahl Aufträge und der jeweils zur Verfügung stehenden Stunden, da man diese immer wieder erneut neu betrachten muß.

Diese interne Umplanung wird ebenfalls mit Hilfe des Solvers erledigt. Hierfür generiert man sich bezüglich jeder Präferenzklasse einen Constraint, der dem Solver hinzugefügt wird und nicht gebrochen werden darf. Diese Bedingung stellt sicher, dass die zur Verfügung stehende Zeit bezüglich dieser Gewichtungsklasse für die bereits eingeplanten und umzuplanenden Aufträge nach hinten ausreicht.

5.3.3 Benutzeroberfläche

Kommen wir in diesem Abschnitt zur Beschreibung der Benutzeroberflächen, mit denen die natürlichen Disponenten oder Arbeiter mit dem System kommunizieren können und gewisse Prozesse starten oder steuern können.

Da das gesamte System auf Java-Technologie basiert und der Einsatz auf verschiedensten Betriebssystemen möglich sein soll, bietet sich eine Implementierung der Oberfläche basierend auf den etwas neueren *Swing-Komponenten* an. Diese haben den Vorteil, dass die Oberfläche zur Laufzeit generiert wird und vom Benutzer je nach Wunsch auch dem Betriebssystem entsprechend aufgebaut werden können. Potentielle Fehler wie bei den AWT-Komponenten (*Abstract Windowing Toolkit*) werden somit vermieden.

Für den Disponenten zeigt ihm diese Oberfläche die bisher eingeplanten Aufträge, sofern sie nicht schon abgelaufen sind, wobei in ihr folgende Informationen enthalten sind. Der Disponent kann hieraus erkennen, wann der Auftrag angenommen wurde, welchen Typ er hat, wie hoch der Gesamtumfang ist und welche Arbeiter dabei wieviel mit welcher Art von Wunschgewährung beiträgt. Desweiteren kann der Disponent auch neue Anfragen von Hand eingeben und starten, falls er zusätzlich auch Aufträge annimmt, welche er nicht aus dem elektronischen Markt erhält. Abbildung 5.6 zeigt diese Oberfläche, welche beim Disponenten direkt nach seiner Anmeldung auf der Agentenplattform erscheint. Bei dieser Anmeldung startet der Disponent auch direkt eine Suche nach allen Arbeiteragenten unabhängig vom Typ, welchen diese herstellen können, um sie in seiner Oberfläche darzustellen. Er hat somit jederzeit den kompletten Überblick über alle ihm bekannten Arbeiter mit ihren zu erledigenden Aufgaben.

Abbildung 5.6: Beispiel für Disponenten-GUI

Die Oberfläche für den Arbeiter-Agenten wurde ebenfalls mit den *Swing-Komponenten* implementiert. Für den Arbeiter öffnet sich hierbei ein Hauptfenster, in welchem alle anderen Fenster, die im Zuge des Prozesses zur Bearbeitung gebraucht wer-

den, als sogenannte *ChildFrames* geöffnet werden. Seine ihn betreffenden Aktionen werden somit einheitlich in einem eigenen sich abgrenzenden Rahmen zusammengefaßt. Während die Anmeldung an der Plattform beim Disponenten durch diesen eigenständig durchgeführt werden muß, werden alle Arbeiter bei FIPA-OS direkt geöffnet. Mit der Schaltfläche *Add Constraints* kann der Arbeiter den Kalender, in welchen er seine Präferenzen eintragen kann, öffnen. In diesen kann er dann wie im vorangegangenen Abschnitt erörtert seine Wünsche bezüglich jeder einzelnen Arbeitsstunde eintragen und sie dem System hinzufügen. Wie schon besprochen, wird das System darauf bei einem neuen Auftrag die für ihn bezüglich seiner Wünsche optimalen Gebote ohne weiteres Eingreifen des natürlichen Arbeiters berechnen. Seinen aktuellen Auftragsplan erhält er ebenfalls über diese Schaltfläche. Abbildung 5.7 soll die beschriebene Oberfläche nochmals graphisch darstellen.

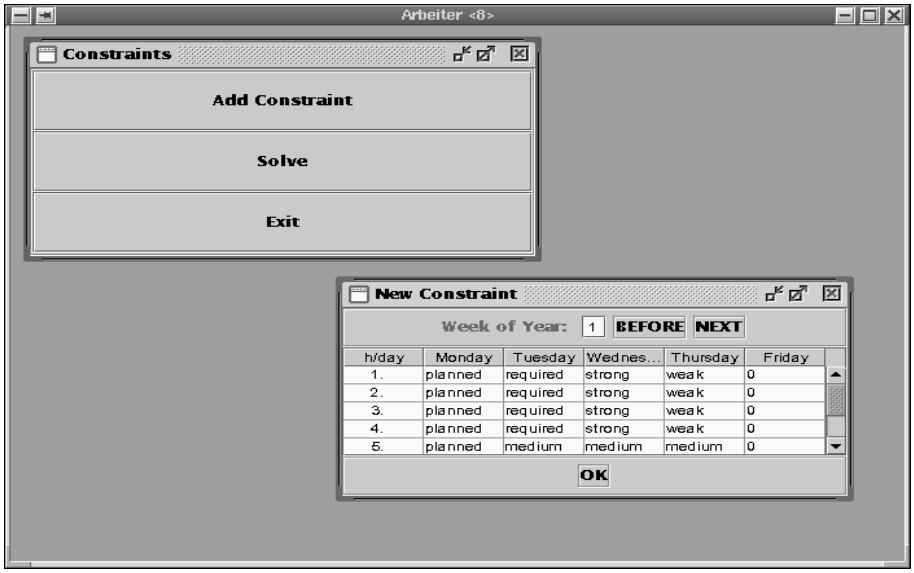

Abbildung 5.7: Beispiel für Arbeiter-GUI

5.4 Evaluierungskriterien

In diesem Abschnitt werden wir die Kriterien zum systematischen Belegen der Leistungsfähigkeit des implementierten Systems beschreiben. Wir werden dazu Testszenarien kreieren, welche einerseits die allgemeine Vorgehensweise des Softwarepaketes nochmals demonstrieren, aber andererseits auch die Mächtigkeit des Systems belegen werden. Das Verhalten des Softwarepaketes wird somit auch bei hoher Auslastung im Grenzbereich unter Beweis gestellt werden.

Nach der Methodik von Cohen (1995) bestimmen wir zuerst die unabhängigen und abhängigen Variablen, welche im Verlauf der Berechnung zur Problemlösung auftre-

ten. Eine unabhängige Variable kann extern manipuliert werden und hat Auswirkungen auf die abhängigen Variablen. Eine abhängige Variable repräsentiert Aspekte des Verhaltens vom gesamten System, wobei ihre Werte von den unabhängigen beeinflußt werden [Cohen, 1995].

5.4.1 Unabhängige Variablen

Betrachten wir hierzu nochmals die Ausgangslage in unserem Problem. Der Disponent erhält einen potentiellen Auftrag, der an einen Typ, eine Menge und einen Abgabetermin gebunden ist. Diese drei Größen sind demnach klar unabhängige Variablen, welche einen Einfluß auf das Ergebnis haben. Nun ist auch gefordert, dass das System den Prozeß nicht nur einmal, sondern fortlaufend neue Aufträge bearbeiten und einplanen kann. Um eine sinnvolle Simulation zu starten, ist somit auch die Anzahl der Aufträge eine unabhängige Variable.

Seitens des Arbeiters sind die Kapazität der Maschine, welche der Arbeiter bedient, sowie der Typ, den diese Maschine produzieren kann, extern vorgegebene und somit veränderbare Größen. Von besonderem Interesse sind hier die Wünsche der Arbeiter, welche ebenso unabhängig sind.

Desweiteren sind die Anzahl der Disponenten sowie der Arbeiter von außen bestimmbare Größen. Dabei ist die Anzahl der Disponenten im Rahmen der Evaluierung nicht von gesondertem Interesse, da sie, wie wir noch erörtern werden, keinen Einfluß auf die zu messenden und bezüglich der Perfomanz interessanten Eigenschaften des Softwarepaketes haben. Es genügt hier darauf hinzuweisen, dass das System jedoch auch mit mehreren Disponenten gleichzeitig fehlerlos läuft.

Gerade im Hinblick auf die Optimierung im Disponenten spielt die Anzahl der am jeweiligen Prozeß teilnehmenden Arbeiter aber eine entscheidende Rolle. Da diese Variable in dieser Optimierung einen exponentiellen Einfluß hat, ist sie somit auch diejenige Größe, welche die Perfomanz des gesamten Systems am meisten beeinträchtigen kann.

Fassen wir nun, bevor wir erörtern werden, was wir an dem System messen wollen, die eben bestimmten unabhängigen Variablen zusammen:

- Auftragskapazität: $Kapazität_{Auftrag} \in \mathbb{N}$
- Auftragstyp: $Typ_{Auftrag} \in \{A, B, C, D\}$ (In den Testläufen wird unter vier verschiedenen Auftragstypen unterschieden, Name und Anzahl der Möglichkeiten ist hierbei frei gewählt.)
- Abgabetermin: $Datum$
- Anzahl der Aufträge: $n \in \mathbb{N}$
- Disponentenanzahl: $d \in \mathbb{N}$ (In den Testläufen wird diese Größe, wie eben schon erläutert, vernachlässigt.)

- Arbeiteranzahl: $m \in \mathbb{N}$
- Maschinentyp: $Typ_{Maschine} \in \{A, B, C, D\}$ (passend zu potentiellen Auftragstypen)
- Maschinenkapazität $[\frac{Menge}{Zeiteinheit}]$: $Kapazität_{Maschine} \in \{10, 20, 30, 40, 50\}$ (Auch diese Werte sind frei gewählt, die Testläufe sind auf diese Größen abgestimmt.)
- Wünsche der Arbeiter: Der Arbeiter kann jede einzelne seiner $\frac{40}{Woche}$, was $\frac{8}{Tag}$ entspricht mit folgenden Prioritäten belegen:
  - *required*: Diese Wünsche des Arbeiters dürfen auf keinen Fall gebrochen werden. In der graphischen Benutzerschnittstelle (GUI) trägt er diese aus Gründen der Einfachheit mit 4 ein.
  - *strong*: Diese Wünsche sind dem Arbeiter sehr wichtig, können aber, wenn unbedingt erforderlich, gebrochen werden. In die GUI trägt er sie mit 3 ein.
  - *medium*: Diese Wünsche sind von gesteigertem Interesse, können jedoch gebrochen werden, er notiert sie als 2.
  - *weak*: Diese Klasse von Wünschen wird nur eingehalten, wenn wirklich genügend Spielraum im Plan des Arbeiters vorhanden ist, sie sind auch für ihn nicht unbedingt wichtig, er notiert sie mit 1:
  - "0": Diese Stunden können vom Disponenten frei eingeplant werden, da sie der Arbeiter mit keinerlei Priorität belegt hat.

Diese beschriebenen unabhängigen Variablen können also in den Testszenarien frei gesetzt werden oder zufällig generiert werden. Um diese Testläufe zu vereinfachen, haben wir hierfür einen sogenannten *Generatoragenten* kreiert, der beim Starten der Agentenplattform ebenfalls direkt gestartet wird und die Fähigkeit besitzt, andere Agenten zu starten. Er besitzt eine graphische Benutzerschnittstelle, über die alle oben aufgeführten unabhängigen Variablen bestimmt werden können. Abbildung 5.8 zeigt diesen Agenten mit seiner kreierten Oberfläche.

Die in diese GUI eingetragenen Werte benutzt der Generatoragent, um alle anderen Agenten, also Arbeiter wie Disponenten mit den hier übergebenen Parametern zu starten. Alle Werte, die nicht explizit gesetzt werden, werden dann randomisiert aus den gegebenen Domänen gesetzt. Eigenschaften des Arbeiters, dies sind der Maschinentyp, die Maschinenkapazität sowie die Generierung der Wünsche bleiben dann solange bestehen, wie der Agent auf der Plattform angemeldet ist. Ausnahme bilden hier die Wünsche, welche natürlich noch im laufenden Betrieb über die graphische Benutzerschnittstelle des Arbeiters verändert werden können. Wird in der Startkonfiguration des Generatoragenten die Wahl auf zufällige Wünsche gestellt, so generiert das System über den gesamten Kalender des betreffenden Jahres einen voreingestellten Prozentsatz bezüglich der Gesamtstunden jeder Klasse von Prioritäten, welche randomisiert über dieses Jahr verteilt werden. Sollen keine Wünsche zufällig generiert werden, so werden alle Stunden ohne Einschränkung eingeplant, sofern sie nicht über die GUI vom Arbeiter verändert werden.

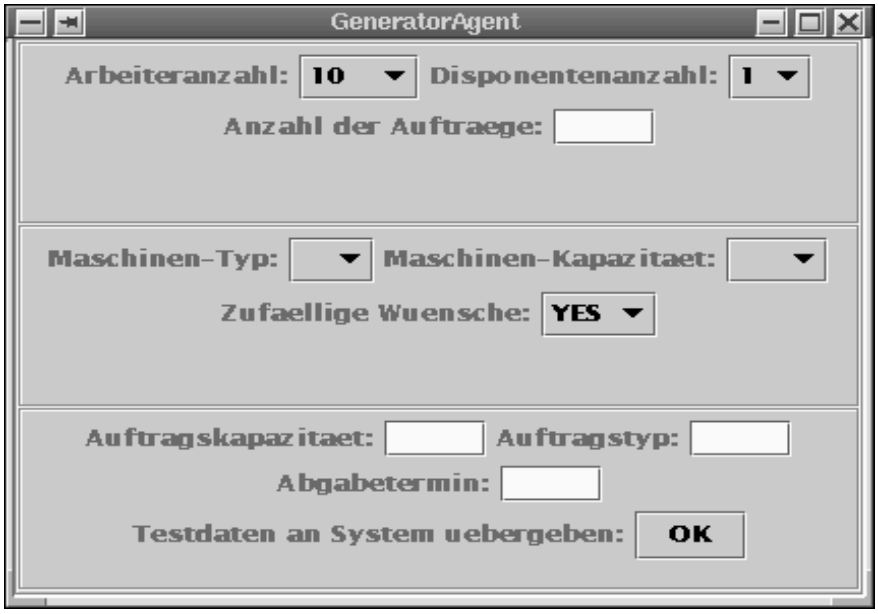

Abbildung 5.8: Generatoragent

Variablen, die den Auftrag betreffen, werden, sofern sie nicht explizit vom Benutzer vorgegeben werden, bezüglich ihrer Domänen dann zufällig für jeden neuen Auftrag gesetzt. Nach gewünschter Konfiguration kann das System dann mit dem OK-Button gestartet werden, die Simulation startet man über die GUI des Disponentenagenten mit dem TEST-Button.

5.4.2 Abhängige Variablen

Kommen wir nun zu den abhängigen Variablen des Systems, also den Werten, die das Verhalten des gesamten Softwarepaketes beschreiben. Hierfür werden wir zuerst festlegen, was dieses Verhalten ausmacht, also was wir demnach messen werden.

Von großem Interesse sind sicherlich die Antwortzeiten des Systems, also die Zeit, bis ein komplettes Protokoll in allen beteiligten Agenten durchlaufen wurde und somit ein Auftrag eingeplant beziehungsweise abgelehnt wurde, falls er nicht erfüllbar war. Da diese Antwortzeiten wohl am stärksten von der Anzahl der am Protokoll beteiligten Arbeiteragenten abhängt, werden mehrere Testläufe mit unterschiedlicher Arbeiteranzahl und Zeitintervallen gestartet und deren Ergebnisse verglichen.

Wenn man nun den Disponenten betrachtet, so ist es sicherlich sinnvoll, die Anzahl der von ihm eingeplanten Aufträge in Relation zu allen möglichen Aufträgen, welche er überhaupt hätte annehmen können, zu betrachten. Die Anzahl der potentiell möglichen Aufträge hängt hierbei natürlich noch von den einzelnen Auftragskapazitäten sowie den Kapazitäten der Maschinen und der Auslastung der betreffenden Arbeiter im betrachtetem Zeitraum ab. Dies ist insbesondere bei den Testszenarien zu beachten,

denn es könnte sein, dass der Disponent eine Vielzahl von Aufträgen ablehnt, trotzdem aber eine gute Auslastung hat.

Da in dieser Arbeit das *soft constraint solving* einen wichtigen Aspekt bildet, soll diese Auslastung ohne Präferenzen in den Plänen der Arbeiter gemessen werden und danach mit Präferenzen. Um aussagekräftige Ergebnisse zu erlangen, müssen alle anderen Variablen in den Testszenarien identisch sein. Es soll somit die Frage beantwortet werden, wie sich das Einbeziehen der Präferenzen auf das gesamte Planungsverhalten auswirkt. Damit ein vollständiger Überblick über dieses Planungsverhalten gewonnen werden kann, werden diese Testläufe bezüglich vieler verschiedener Konfigurationen im System, also Instanzen der anderen unabhängigen Variablen vollzogen.

Auch im Arbeiter ist diese Auslastung eine abhängige Variable, welche mit den selben oben angeführten Voraussetzungen und Umgebungen gemessen wird. Die Auslastung im Arbeiteragenten wird dabei exakt definiert durch die Anzahl der belegten Stunden, also eingeplanten Stunden im Arbeitskalender des Arbeiters, in Relation zu allen zur Verfügung stehenden Stunden im betrachteten Zeitintervall.

Diese beiden Auslastungen sind nicht, wie man im ersten Augenblick denken könnte, identisch, denn im Arbeiter wird so das Planungsverhalten über die konkrete Anzahl von Stunden mit konkreten Präferenzen festgehalten, im Disponenten wird eher die Gesamtheit betrachtet. Verschiedene Maschinenkapazitäten haben zum Beispiel dabei unterschiedliche Auswirkungen auf diese beiden abhängigen Variablen. Im Arbeiter wird also die Gewährung der Präferenzen konkret auf eine Person bezogen gemessen, im Disponenten das Scheduling anhand der zusätzlichen Bedingungen, welche durch die Präferenzen entstehen und in den Geboten der Arbeiter zum Ausdruck kommen.

Zum Abschluß fassen wir nochmals die abhängigen Variablen im System übersichtlich zusammen:

- Durchschnittliche Antwortzeiten $t^m_{Antwort}$: Dieser Wert hängt ab von der Arbeiteranzahl *m* und dem betrachteten Zeitraum, also den Abgabeterminen der Aufträge.
- Auslastung des Disponenten: $[\frac{|angenommene Aufträge|}{|bearbeitete Aufträge|}]$
  - Ohne Präferenzen: Hier ist zu beachten, dass die Anzahl der bearbeiteten Aufträge noch in Relation zu den Kapazitäten der Maschinen sowie der Auslastung der betreffenden Arbeiter im betrachtetem Zeitraum zu setzen ist.
  - Mit Präferenzen: Hier kann von dem Aspekt mit der Anzahl der bearbeiteten Aufträge abgesehen werden, da sowieso ein Vergleich zu der Lage ohne Präferenzen mit identischen Voraussetzungen gezogen wird und die Änderung von Interesse ist.
- Auslastung der Arbeiter: $[\frac{|eingeplante Arbeitsstunden|}{|alle Arbeitsstunden|}]$
  Dieses Maß gibt das Verhältnis der eingeplanten Stunden zu allen möglichen an. Das Augenmerk liegt dabei auf den Werten mit und ohne Präferenzen sowie

jeweils noch mit dem Einbeziehen unterschiedlicher Kapazitäten der Maschinen, welche die Arbeiter bedienen.

Nachdem wir die unabhängigen und abhängigen Variablen definiert und erläutert haben, können wir nun die Testszenarien beschreiben, an denen wir die erörterten Daten messen und im folgenden Abschnitt der *Perfomanzanalyse* auswerten werden.

Antwortzeiten

Um die durchschnittlichen Antwortzeiten $t^m_{Antwort}$ des Systems zu ermitteln, werden je 100 Protokolle mit den Arbeiteragenten gestartet, deren Anzahl sukzessive gesteigert wird, während dieser 100 Protokolle jedoch konstant bleibt. Desweiteren wird mit jeder festen Arbeiteranzahl auch der Abgabetermin der einzuplanenden Aufträge verändert, um festzustellen, wie sich längerfristig zu planende Aufgaben, bei denen trotzdem jede einzelne Stunde im Arbeiteragenten bezüglich der belegten Präferenz getestet werden muß, auf die Gesamtantwortzeiten der Protokolle auswirkt. Alle anderen unabhängigen Variablen haben keinen Einfluß auf diese Antwortzeiten, da sich deren Rechenaufwand nicht verändert bei unterschiedlichen Belegungen.

Disponentenauslastung

- Ohne Präferenzen: In diesen Testläufen bleibt die Anzahl der Arbeiter konstant. Es wird gemessen, wie der Disponent einkommende Aufträge auf die einzelnen Arbeiter verteilt, die hier ihre Gebote ohne ihre Wünsche geben müssen. Zu beachten sind die unterschiedlichen Kapazitäten der Maschinen und das Verhalten beim Verteilen im Verlauf der Protokolle und den zunehmend geringer werdenden Gebote der Arbeiter.

- Mit Präferenzen: Während dieser Testläufe sind die Werte zu den vorangegangene Szenarien identisch mit der Ausnahme, dass die Kalender der Arbeiter nun mit Präferenzen belegt sind. In diesem Teil der Evaluierung soll die Frage beantwortet werden, wie der Disponent die Aufträge auf- und verteilt speziell unter genauer Beobachtung der verschiedenen Kapazitäten und Wünsche der Arbeiter.

Auch hier werden wir jeweils mehrere Testläufe mit je 100 Protokollen starten, wobei immer nur der Wert einer unabhängigen Variablen geändert werden darf, um einheitliche Aussagen treffen zu können.

Arbeiterauslastung

Die Testdaten für die Evaluierung im Arbeiteragenten sind analog zu denen im Disponentenagenten. Es wird verglichen, wie sich der Arbeitsplan des Arbeiters ohne Wünsche gestaltet in Relation zu der Entwicklung, wenn er Präferenzen angibt. Somit wird die Frage beantwortet, ob die Arbeiter, welche an Maschinen mit hoher Kapazität arbeiten, im Vergleich zu denen mit geringer Maschinenkapazität mehr arbeiten durch die Zuteilung der Disponenten. Insbesondere soll dies auch unter Einbeziehen der Präferenzen der Arbeiter beobachtet werden.

Auch hier sei nochmal erwähnt, dass die beiden Auslastungen der Agenten unterschiedliche Aspekte sind. Ein Disponent hat keinerlei Kenntnisse über die Kapazität der Maschinen und die tatsächlich benötigten Stunden der Arbeiter, da er ihre Arbeitspläne nicht kennt, er bekommt verschiedene Gebote, an deren Daten er eine Entscheidung trifft. Dieses Entscheidungsverhalten wird beobachtet und bewertet. Im Arbeiter wird die Anzahl der belegten Stunden in Relation gesetzt werden zu allen möglichen Stunden. Dies kann man dann bezüglich verschiedener Kapazitäten und Präferenzlage vergleichen, man erhält also einen anderen Wert als im Disponenten.

5.5 Perfomanzanalyse

In diesem Abschnitt werden wir die aus den Testszenarien herauskommenden Daten beschreiben und analysieren. Sämtliche Testläufe haben wir auf einer *Sun Enterprise E 420 R* mit vier Prozessoren des Typs *Ultra Sparc 2* mit je 480 MHz Taktfrequenz und einem Arbeitspeicher von 4 GB durchgeführt, das installierte Betriebssystem war *Solaris 2.8*.

5.5.1 Durchschnittliche Antwortzeiten

In den Testläufen mit je 100 Protokollen, welche wir mehrfach bezüglich jeder Ausgangskonfiguration haben laufen lassen, wurden zahlreiche Daten gesammelt, von denen wir in diesem Abschnitt einen Teil darstellen werden, um zu demonstrieren, wie wir aus der Gesamtheit das im vorangegangen Abschnitt definierte Verhalten des Softwarepaketes bezüglich der Antwortzeiten analysieren. In sämtlichen Testläufen waren alle beteiligten Agenten auf der lokalen Maschine angemeldet, so dass sich die Antwortzeiten in der Praxis noch um die Zeit, welche für das Versenden der Nachrichten über verschiedene Plattformen benötigt wird, verlängert. Abbildung 5.9 stellt die

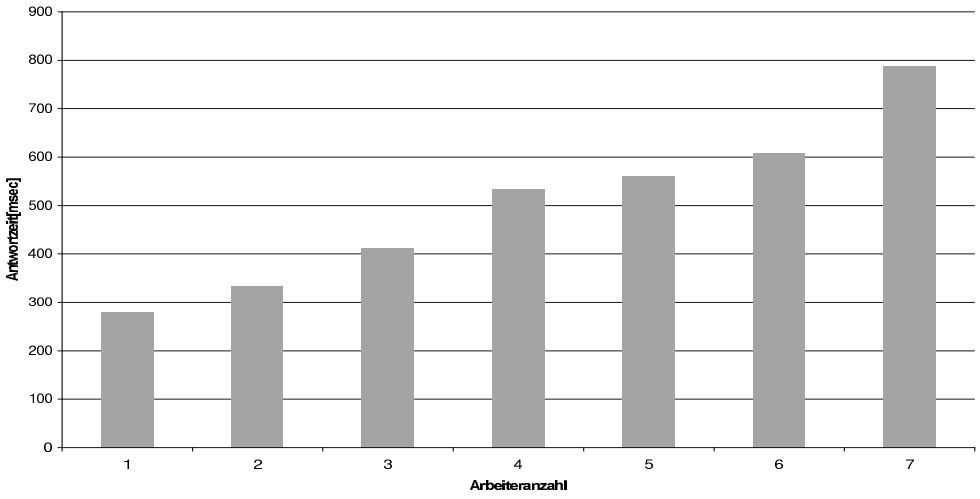

Abbildung 5.9: Antwortzeiten I

durchschnittlichen Antwortzeiten bezüglich einem bis zu sieben Arbeiteragenten gra-

phisch dar, welche sich auf Tabelle 5.1 beziehen. Man sieht, dass sich diese für jede Arbeiteranzahl noch unter einer Sekunde bewegt. Im Vergleich zu einer Spedition, in welcher ein natürlicher Disponent eine Fahrt planen muß, bei der er unter sieben möglichen Fahrern wählen kann und dies ”von Hand” macht, sind diese Antwortzeiten mehr als zufriedenstellend.

	Arbeiter										
Test	1	2	3	4	5	6	7	8	9	10	11
1	275	385	448	695	623	685	819	1709	4744	22483	103726
2	272	360	364	499	574	706	917	1435	4451	23757	102807
3	321	322	350	538	508	603	780	1463	4408	26523	103072
4	305	322	472	579	499	715	818	1325	4233	24149	104122
5	275	323	399	498	459	636	816	1470	4707	26082	102037
6	290	367	422	540	611	634	820	1494	4499	23744	104334
7	292	305	392	594	521	515	812	1670	4437	21987	103157
8	296	311	386	559	487	627	747	1527	4364	24403	104649
9	281	315	381	595	486	614	745	1289	4225	23613	102696
10	261	338	510	535	657	623	744	1340	4707	27441	102934
11	259	311	442	502	507	607	747	1318	4370	25088	
12	272	355	395	532	592	523	723	1397	4295	24428	
13	280	384	467	563	497	607	817	1315	4193	23928	
14	289	346	426	537	566	588	765	1411	4259	25537	
15	265	304	430	534	541	559	740	1401	4665	25856	
16	257	300	361	564	540	609	851	1403	4284	25378	
17	274	318	415	552	463	688	750	1425	4234	25384	
18	283	308	414	546	541	608	732	1254	4227	22968	
19	283	315	403	576	529	491	809	1692	4225	26394	
20	283	382	359	547	557	551	795	1413	4614	22756	
Ø	280	333	411	534	560	609	787	1437	4407	24594	103353

Tabelle 5.1: Durchschnittliche Antwortzeiten der Testszenarien

Abbildung 5.10 stellt die unter identischen Bedingungen zu vorangegangener Abbildung erhaltenen Daten für die Testläufe dar, in denen mehr als sieben Arbeiteragenten beteiligt waren. Diese Antwortzeiten bewegen sich bis einschließlich elf Agenten im akzeptablen Rahmen. Ab zwölf Agenten würde das System zwar genauso eine optimale Lösung berechnen, aber hier sind die Rechenzeiten zu hoch, dass eine Zeitschranke diese Berechnungen abbricht. Es bleibt anzumerken, dass die Testläufe ohne die im Disponentenagenten implementierte Zeitschranke durchgeführt wurden, welche nach zwei Minuten die Optimierungsphase abbricht und das bis dahin erreichte lokale Minimum als Lösung nimmt und dementsprechend die Arbeiteragenten informiert. Aus Gründen der Übersicht ist die Ordinate in dieser Abbildung logarithmisch skaliert.

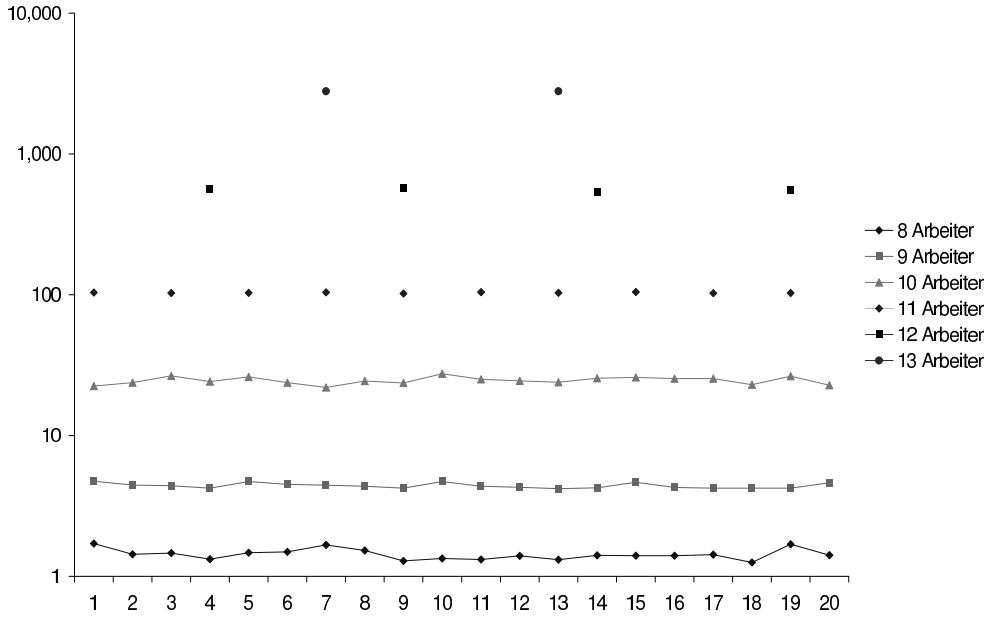

Abbildung 5.10: Antwortzeiten II

Die Anzahl der Testläufe mit mehr als zehn Agenten haben wir reduziert. Bei dreizehn Agenten stieg die Antwortzeit für ein einziges Protokoll im schlechten Fall auf über 45 Minuten, womit dies bei 100 Protokollen über drei Tage in Anspruch genommen hätte. Aus diesem Grund haben wir die Anzahl der Protokolle bei dieser hohen Agentenanzahl auf 50 reduziert. Bei zwölf Agenten benötigte das System ungefähr $15\frac{3}{4}$ Stunden für einen Testlauf mit 100 Protokollen.

Nachstehende Tabelle 5.2 enthält ausgewählte Daten aus den Testläufen unter Einbeziehung längerer Planungszeiträume. Beim Generieren der *soft constraints* wird im Arbeiteragenten dessen Kalender für den entsprechenden Zeitraum nach den unterschiedlich gewichteten Arbeitsstunden durchsucht und dem Solver in Form von Constraints übergeben. Je länger der Planungszeitraum dabei wird, um so größer wird die dabei durchzusuchende Datenstruktur. In den Testläufen blieben abgesehen von der Arbeiteranzahl und dem Planungszeitraum alle anderen unabhängigen Variablen konstant.

Die Antwortzeiten steigen dabei bei längerem Planungszeitraum erwartet linear an, was die Werte in etwa belegen. Sämtliche Zuwächse liegen unabhängig von der Arbeiteranzahl ungefähr bei 1,7 Prozent bis 2,0 Prozent bezogen auf eine Planungswoche beziehungsweise 52. Diese Schwankungen sind durch die unterschiedlichen Testumgebungen zu erklären, obwohl sie trotzdem noch in einem akzeptablen Rahmen liegen. Die Testläufe wurden auf der eingangs des Abschnitts beschriebenen Maschine durchgeführt, auf der zu unterschiedlichen Zeiten verschieden viele andere Prozesse liefen, worauf diese kleinen Abweichungen zurückzuführen sind. Generell kann man jedoch sagen, dass der Planungszeitraum einen nur sehr kleinen Einfluß auf die Gesamtant-

wortzeiten hat, was nicht unbedingt von vorne herein so zu erwarten war, da gerade im Arbeiteragenten erheblich mehr Vergleiche nötig sind je länger der Planungszeitraum wird.

	Planungszeitraum[Wochen]						
Arbeiter	1	8	16	24	32	40	52
1	280	280	282	283	283	284	285
2	333	332	334	335	337	337	339
3	411	412	415	415	416	417	418
4	526	528	531	533	533	534	535
5	557	559	562	563	564	566	568
6	609	612	617	617	618	619	621
7	778	780	785	787	789	791	794
8	1437	1444	1452	1455	1458	1460	1465
9	4407	4430	4471	4478	4483	4489	4506
10	24895	24912	25102	25197	25248	25299	25362
11	103353	103351	103849	104083	104339	104685	105201

Tabelle 5.2: Durchschnittliche Antwortzeiten [msec] bei längerem Planungszeitraum

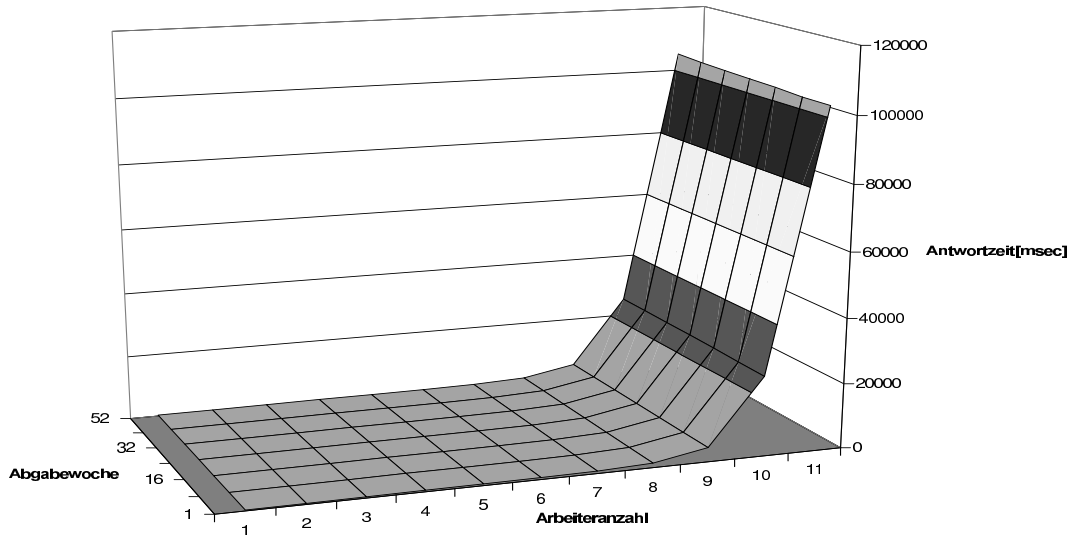

Abbildung 5.11: Antwortzeiten III

Abschließend wollen wir die gemessenen Antwortzeiten in Bezug auf die Arbeiteranzahl und den Planungszeitraum nochmals zusammen betrachten. Abbildung 5.11 stellt das Ergebnis in einem ausgewählten Teilabschnitt nochmals graphisch dar. Wie man sieht, liegen dabei die durchschnittlichen Antwortzeiten unterhalb von zehn Agenten im gesamten Planungszeitraum in einem sehr guten Bereich, nämlich unter einer Sekunde, bei neun am Protokoll teilnehmenden Arbeiteragenten liegt sie bei etwa 4,5 Sekunden, bei zehn immerhin noch bei ungefähr 25 Sekunden. Nachfolgend steigen

diese Zeiten der Problemstellung entsprechend exponentiell stark an. Bis zu zwei Minuten sind die Antwortzeiten seitens des Disponenten in einem akzeptablen Rahmen, so dass das System sogar noch für elf beteiligte Arbeiteragenten eine optimale Lösung bezüglich der Strafpunkte garantiert. Diese zwei Minuten sind im Vergleich zum ´´manuellen´´ Planen eines Disponenten in der Geschäftswelt mehr als eine gute Schranke.

Das System läuft auch mit beliebig vielen Agenten stabil, wir haben mehrfach Testläufe mit mehr als 50 Agenten verschiedenen Typs über einen längeren Zeitraum laufen lassen, ohne dass es zu fehlerhaften Protokollen gekommen ist.

5.5.2 Disponentenauslastung

In diesem Teil der Evaluierung werden wir die Auslastung des Disponenten analysieren. Wir werden dabei speziell die Testläufe miteinander vergleichen, bei denen die Konfigurationen bezüglich des Generatoragenten identisch waren ausgenommen der Wünsche der Arbeiter. Durch das Einbeziehen der Wünsche der Arbeiter verändern sich deren Gebote und wir werden beobachten, welche Auswirkungen dies auf die Auftragsvergabe hat.

Die Anzahl der Arbeiter spielt in dieser Evaluierungsphase keine Rolle, wir haben sämtliche Testläufe mit jeweils acht Arbeiteragenten durchgeführt. Vielmehr ist das Verhältnis der Auftragsgröße zu der Gesamtkapazität über alle Arbeiter ein wichtiger Aspekt. Um aussagekräftige Resultate zu bekommen, haben wir die Summe der Kapazitäten $Kapazität_{Auftrag}$ aller Aufträge *n* über den Planungszeitraum *Datum* so gewählt, dass sie gerade durch alle Arbeiter *m* mit ihren jeweiligen Kapazitäten $Kapazität_{Maschine}$ über dem Planungszeitraum zu erfüllen waren.

$$\sum_{i=1}^{n} Kapazität_{Auftrag} \leq \sum_{i=1}^{m} \sum^{Datum} Kapazität_{Maschine}$$

Durch diese Wahl konnten wir zum einen feststellen, wie sich die Verteilung bezüglich dem Planungszeitraum und den damit einhergehenden geringer werdenden Gebote der Arbeiter entwickelte, zum anderen konnten wir so direkt messen, wieviele der einzuplanenden Aufträge durch Hinzunahme der Präferenzen der Arbeiter abgelehnt werden mußten.

An dieser Stelle sei nochmals erwähnt, dass der Disponent keine Kenntnisse über die Präferenzen und die Arbeitspläne der Arbeiter hat. Er entscheidet lediglich anhand der ihm zugesandten Gebote, welche die Präferenzen indirekt zum Ausdruck bringen, und den zu zahlenden Strafpunkten für die jeweiligen Zuschläge, welcher Arbeiter wieviel zu produzieren hat. Ein hohes Gebot im Bereich *weak* eines Arbeiters kann durch eine hohe Maschinenkapazität zustande kommen aber ebenso durch eine relativ geringe Präferenzlage für den betrachteten Zeitraum.

Die Ergebnisse zeigen, dass der Prozentsatz der angenommenen Aufträge des Disponenten in etwa mit dem Prozentsatz der zufällig verteilten *required*-Wünsche korreliert. Bei zufälliger Verteilung von 5 Prozent harter Constraints, also Wünschen, die

nicht gebrochen werden dürfen, lag die Auslastung bei etwa 95 Prozent im Vergleich zu den 100 Prozent bei oben beschriebener Konfiguration ohne Präferenzen der Arbeiter.

Dabei haben wir die Größe eines einzelnen Auftrags so gewählt, dass er gerade von mehr als einem Arbeiter im betrachteten Planungszeitraum erfüllt werden muß. Ein einzelner Auftrag von der Größe der Gesamkapazitäten aller Arbeiter hätte das Resultat gehabt, dass der Disponent alle Aufträge, obwohl es nur einer war, hätte ablehnen müssen. Dieser Fall ist durchaus möglich, spiegelt jedoch nicht das normale Verhalten auch im Hinblick auf den anwendungsspezifischen Hintergrund des implementierten Systems wieder. Abbildung 5.12 stellt eine generelle Beobachtung im Disponenten

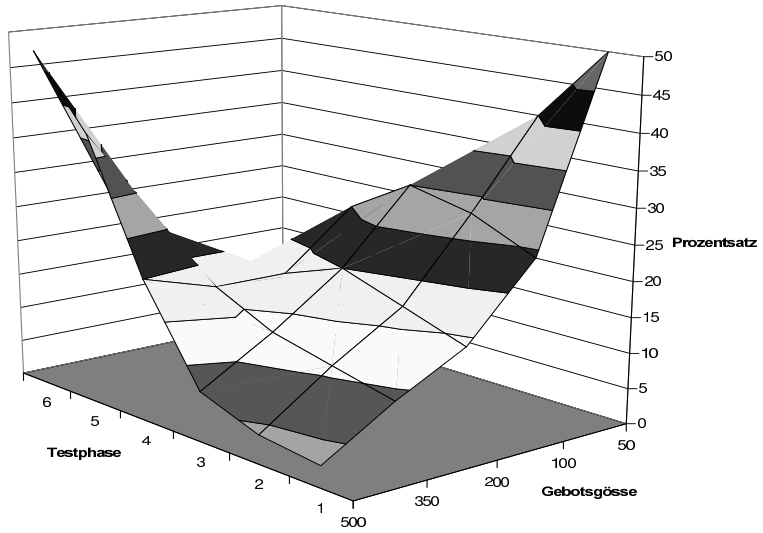

Abbildung 5.12: Disponentenverhalten

unter ausgewählten Parametern dar. Zu Beginn einer Simulation sind die Pläne der Arbeiter noch leer, deswegen sind ihre Gebote sehr hoch, zumal ihre mit Präferenzen belegten Stunden im Verhältnis zur noch freien Gesamtstundenanzahl ebenso noch sehr gering ist. Prinzipiell wählt der Algorithmus im Disponenten die Kombination mit geringster Strafpunktanzahl und möglichst großer Produktionsmenge. Diese beiden Tatsachen führen dazu, dass zu Beginn einer Simulation zunächst Gebote mit hoher Kapazität gewählt werden. Nach mehreren Protokollen verschiebt sich dieses Verhältnis. Dies hat zwei Ursachen. Zum einen sind die Pläne der Arbeiter nun schon recht belastet und lassen sowieso nur wenig Spielraum für neue Aufträge, zum anderen haben sich durch den Fortlauf der Simulation die Anzahl der Stunden, die der Arbeiter mit *weak* präferenziert hat, verringert. Der Disponent versucht trotzdem, die Strafpunkte so gering wie möglich zu halten, weswegen die Aufträge immer mehr in Teilaufträge ”gesplittet” werden und auf viele Arbeiter verteilt werden. Deswegen verschiebt sich die hohe Verteilung gegen Ende einer Simulation zu den Geboten geringerer Größe.

	Auftrag[Typ X; 2000 ; 2 Wochen]					
Maschinenkapazität	1	2	3	4	5	6
10	0	0	0	0	0	800
20	0	0	0	0	1600	0
30	0	0	2000	0	0	400
40	0	2000	0	0	400	800
50	2000	0	0	2000	0	0

Tabelle 5.3: Beispiel für Auftragsverteilung

Das Beispiel aus Tabelle 5.3 belegt den Verlauf der Abbildung 5.12 nochmals tabellarisch mit Werten. Aus Gründen der Übersicht haben wir hier ein Beispiel mit nur fünf Arbeitern unterschiedlicher Maschinenkapazität gewählt und betrachten einen Planungszeitraum von nur zwei Wochen, wobei die Auftragsgröße konstant bei 2000 liegt. Desweiteren sind in dem Beispiel keine Präferenzen berücksichtigt. Die Tabelle zeigt die Verteilung der Aufträge auf die chronologisch eintreffenden Aufträge 1 bis 6. Auch hier sieht man, dass zu Beginn der Simulation größere Zuschläge erteilt werden als gegen Ende der Simulation.

Die Verteilung der Einzelkapazitäten durch den Disponenten richtet sich nach der Größe der Gebote und der damit verbundenen Strafe, welche der Disponent für das Brechen von Wünschen ”bezahlen” muß. Da der Algorithmus diese minimiert, verteilt der Disponent demnach auch die Aufträge an die mit den höchsten Geboten mit geringster Präferenzlage. Da ein Arbeiter mit momentan niedriger Auslastung im Verhältnis zu denen mit höherer Auslastung größere Gebote abgibt, bekommt er auch den Zuschlag eher zugeteilt als andere, dies hängt jedoch stark von seiner Maschinenkapazität ab. Bei gleicher Kapazität erhält ein Arbeiter mit geringer Auslastung eher den Zuschlag als einer mit hoher Auslastung respektive der jeweiligen Wünsche. Dies entspricht dem in Kapitel 2 vorgestellten Approximierungsalgorithmus bezüglich Planungsaufgaben, da so ein Auftrag im Normalfall immer an die am wenigsten ausgelastete Maschine oder Arbeiter verteilt wird.

Zusammenfassend können wir festhalten, dass die Präferenzen der Arbeiter Auswirkungen auf auf die Gesamtauslastung entsprechend der Art haben, wie ihnen *required*-Präferenzen von vorne herein gewährt werden. Gewährt man ihnen 5 Prozent harte Constraints, welche also nicht gebrochen werden dürfen, im Verhältnis zu der Gesamtstundenanzahl, so vermindert sich auch die Annahme von Aufträgen im selben Rahmen. Ist der letzte einzuplanende Auftrag einer Simulation im Verhältnis zu den anderen recht groß, so kann dieser Wert der Minderung natürlich auch über dem des gewährten Prozentsatzes der harten Constraints liegen.

5.5.3 Arbeiterauslastung

Im letzten Abschnitt der Evaluierung werden die Auslastungen des Arbeiters gemessen. Diese hängen ab von der Präferenzlage, welche in den Testläufen randomisiert auf die einzelnen Stunden generiert wurden, und der Kapazität der jeweiligen Maschine. Die Beobachtungen und Ergebnisse werden im folgenden erläutert.

Die Gebote der Arbeiter verringern sich mit zunehmender Anzahl von Wünschen über die verschiedenen Prioritätsklassen. Abbildung 5.13 stellt zwei identische Arbeiter mit einer Kapazität von ”10” und ihren daraus resultierenden potentiellen Gebote für eine Woche mit 40 Arbeitsstunden dar. Bei einem Agenten werden diese ohne Berücksichtigung von Constraints gemacht, bei dem anderen berechnet Cassowary bezüglich immer mehr werdender Präferenzen die Gebote. Man sieht, dass sich die Gebote bei dem Agenten mit Wünschen unter Betrachtung immer weiterer Wünsche zunehmend verringern. Entscheidend ist jedoch die Menge, welche bezüglich der *required*-Constraints berechnet wurde (in der Abbildung: 390), da sie gewissermaßen eine obere Schranke bildet, welche der Disponent durch seine Zuschläge nicht überschreiten darf. Alle anderen Werte kann er durch Bezahlen von ”Strafen” überschreiten.

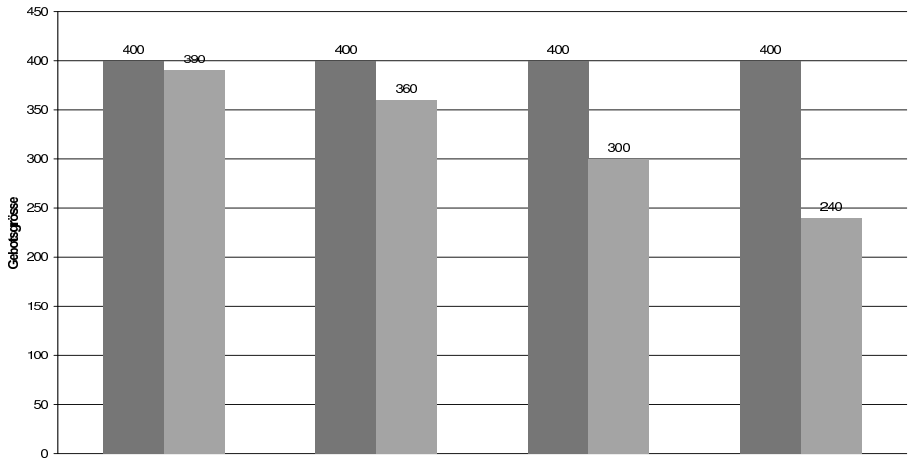

Abbildung 5.13: Gebote der Arbeiter

Die Auslastung eines einzelnen Arbeiters mit Präferenzen und ohne ist der im Disponenten bezüglich der Präferenzen ähnlich. Dort sank die Gesamtauslastung in etwa um den Prozentsatz, den man auch als harte Constraints, also Wünsche, die nicht gebrochen werden dürfen, zugelassen hat. Im Arbeiter entspricht diese exakt dem Prozentsatz zugelassener hoher Prioritäten. Hängt diese Auslastung im Disponenten direkt von der Größe der einzuplanenden Auftragskapazitäten, so ist dies hier nur mittelbar der Fall. Im Arbeiter bilden die Arbeitsstunden die Basis für diese Auslastung, also nicht die Menge, auch wenn sie mittelbar über die Maschinenkapazität auch auf den Arbeiter anzuwenden wäre. Da der Arbeiter jedoch keine fixen Mengen wie der

Disponent einzuplanen hat, sondern Gebote abgibt, welche sich auf einen Zeitraum beziehen, und zudem auch noch von mehreren Disponenten kontaktiert werden kann, gehen wir bei ihm von einem vollen Plan unter Gewährung der jeweiligen Anzahl von harten Constraints aus.

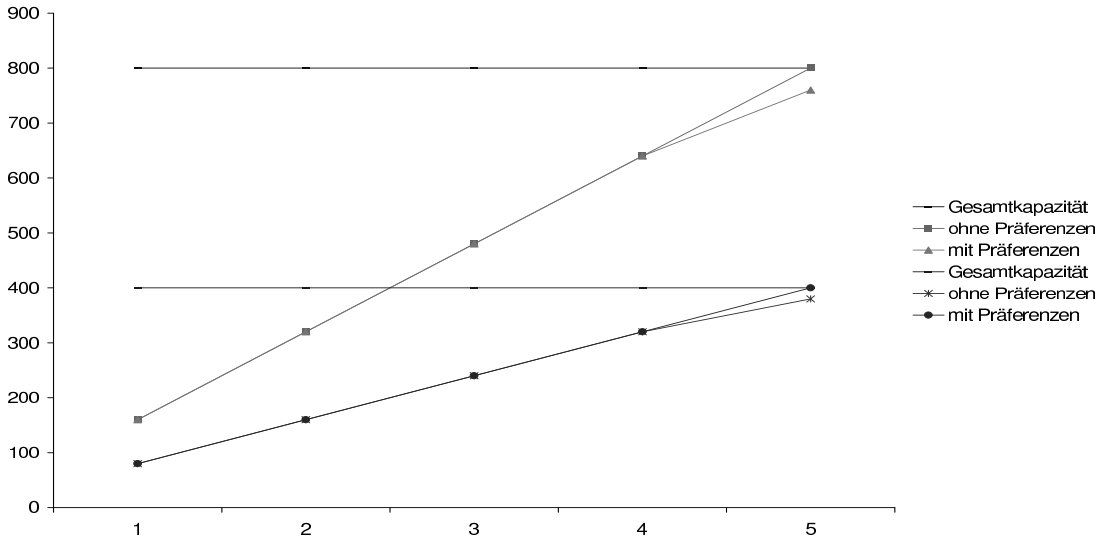

Abbildung 5.14: Auslastung im Arbeiter

Abbildung 5.14 stellt die Auslastung im Arbeiter nochmals graphisch dar. Wir haben zwei Arbeiter dargestellt, welche Maschinenkapazitäten von ”10” beziehungsweise ”20” haben. Betrachtet wird eine Woche, also fünf Arbeitstage, welche auf der Abszisse eingetragen sind. Bei maximal acht Arbeitsstunden am Tag hat der Arbeiter mit Kapazität ”20” eine Gesamtkapazität von 800 Stück, der mit Kapazität ”10” 400. Diese sind auf der Ordinate der Abbildung dargestellt. Falls den Arbeitern keine Wünsche gewährt werden, erreichen sie nach einer bestimmten Anzahl von Zuschlägen, welche wir bezüglich der Menge hier konstant gehalten haben, ihre Gesamtkapazität, also eine hundertprozentige Auslastung. Im Fall mit Präferenzen werden diese 100 Prozent nicht erreicht. Die Ursache, dass sich die Präferenzen erst im letzten Zuschlag, also erst gegen Ende und hoher Auslastung bezüglich eines Planungszeitraumes bemerkbar machen hängt wiederum an dem Disponenten. Dieser versucht, die Wünsche bestmöglich in Betracht zu ziehen, was anfangs einer Simulation dann auch keine Auswirkungen auf den Zuschlag hat, da die Gebote der Arbeiter zu Beginn sowieso recht hoch sind. Erst mit zunehmender Dauer einer Simulation machen sich dann die harten Constraints bemerkbar, was natürlich wiederum von den Auftragskapazitäten abhängt.

Bezüglich einem Auftrag gilt für den Arbeiter, dass er, je höher seine Maschinenkapazität ist, desto höher ist die Wahrscheinlichkeit, dass er einen Zuschlag erhält bei gleichem Planungszeitraum. Dies hat jedoch keine Auswirkungen auf seine Arbeitszeit. Ein Arbeiter mit hoher Maschinenkapazität arbeitet nicht unbedingt mehr als einer

mit niedriger Maschinenkapazität über einen längeren Zeitraum gesehen. Dies ist nur der Fall, falls im laufenden Betrieb nicht genügend Aufträge angeboten werden, so dass die Pläne der Arbeiter nicht voll ausgelastet werden.

Zusammenfassend können wir für den Arbeiter festhalten, dass sich seine Auslastung stark nach der Kapazität seiner Maschine und der ihm gewährten Präferenzen richtet. Desweiteren hängt sie natürlich auch von der Anzahl der eintreffenden Aufträge und Auftragskapazität ab, diese Größen sind jedoch nicht durch ihn beeinflußbar. Prinzipiell wirken sich die Wünsche der Arbeiter natürlich negativ auf deren Gebote aus. Es hängt also davon ab, wieviele Wünsche man bezüglich jeder Prioritätsklasse maximal zuläßt, obwohl entscheidend natürlich die *required*-Constraints sind. Nur sie haben letztlich entscheidenden Einfluß auf den Zuschlag, da durch sie gewissermaßen eine obere Schranke definiert wird.

Kapitel 6

Ergebnis

In diesem Kapitel werden wir die Erkenntnisse, welche sich aus dieser Arbeit ziehen lassen, nochmals kurz zusammenfassen. Hierfür geben wir einen Überblick über die Ergebnisse aus jedem Abschnitt dieser Arbeit. Desweiteren geben wir eine Reihe von möglichen Erweiterungen für das implementierte System an, welche sich im Verlauf dieser Arbeit als weitere erstrebenswerte Ziele herausgebildet haben.

6.1 Zusammenfassung

In Kapitel 3 dieser Arbeit haben wir die Problemstellung genauer analysiert und somit die speziellen Herausforderungen der einzelnen Teilaufgaben definieren können. Auf Basis der in Kapitel 2 erarbeiteten theoretischen Grundlagen konnten wir diese Teilaufgaben auch den Forschungsgebieten zuordnen, deren Erkenntnisse sie sich zu nutzen machen konnten. Zudem konnten wir mit dieser detaillierten Beschreibung auch die praktische Relevanz des Problems unterstreichen.

Nach dieser exakten Problemdefinition haben wir in Kapitel 4 eine Spezifikation angefertigt, welche sich in seinem Aufgabu nach der Methodik von Balzert richtet [Balzert, 1998]. Diese Spezifikation kommt einer Instanzaufnahme des behandelten Problems gleich. Wir haben die Ziele genau definiert, die Umgebung beschrieben als auch die Funktionalität sowie die Produktdaten erörtert, also die Anforderungen an das System ausführlich beschrieben. Diese Spezifikation mit der Beantwortung der Frage, *was* man mit dem System erreichen will, diente als Basis für die Implementierung und Evaluierung in Kapitel 5.

In der Implementierung haben wir beschrieben, *wie* wir das Problem gelöst haben. Zunächst haben wir anhand der in der Spezifikation erarbeiteten Umgebung eine begründete Wahl für einen Solver gegeben, welcher die bei der Lösung entstehenden Constrainthierarchien entscheiden kann. Danach haben wir auf FIPA-OS basierend die beiden benötigten Softwareagenten entworfen und ihre wichtigsten implementierten Methoden, welche zur Lösung des Problems notwendig waren, so wie wir sie in der Spezifikation erörtert haben, explizit erklärt.

In der Evaluierung haben wir schließlich systematisch gezeigt, dass unser implementiertes System auch tatsächlich die geforderten Ziele erreicht und dabei die allgemeingültigen Zeitschranken für diese Problemklassen einhält. In vielen verschiedenen Testreihen haben wir dabei die unabhängigen Variablen des Systems abwechselnd verändert und deren Auswirkungen auf die abhängigen Variablen gemessen. Anhand der Ergebnisse haben wir das Verhalten des gesamten Systems in besonderen Situationen und im Normalbetrieb somit beschreiben können.

Zusammenfassend können wir sagen, dass das implementierte System zeigt, dass man die Forschungsgebiete von *Multiagentensystemen* und des *soft constraint solving* effizient in einem anwendungsspezifischem Rahmen nutzen kann.

6.2 Ausblick

In diesem Abschnitt werden wir potentielle Folgearbeiten, welche sich während der Bearbeitung des gestellten Problems als erstrebenswerte Ziele herauskristallisiert haben, anführen.

Wenn man den Disponenten betrachtet, so sieht man, dass er möglicherweise sehr gute Aufträge ablehnen muß, da seine berechnete Lösung die geforderte Kapazität nicht erreicht. In seinem Plan sieht man jedoch, dass er durch die Abgabe eines bereits angenommenen und eingeplanten Auftrages diesen neuen sehr wohl erfüllen könnte, womit er seine Auslastung erheblich verbessern würde. Doch hierfür müßte er mit einem anderen Disponenten in Kontakt treten und versuchen, den bereits eingeplanten Auftrag an diesen abzutreten. Diesen Aspekt haben wir im Rahmen der Spezifikation unter den Abgrenzungskriterien schon behandelt und gezeigt, dass dies einen kaskadierenden Effekt haben kann und zudem zu *deadlocks* führen kann. Im Rahmen dieser Arbeit war es nicht gefordert, ein einmal eingeplanter Auftrag kann in unserem System nicht mehr abgegeben werden.

Trotzdem ist dies ein erstrebenswerter Faktor, den man möglicherweise mit dem Ansatz des *Simulated Trading*, einer Heuristik, welche im Rahmen der Behandlung von *Vehicle Route Problems* entstand, lösen kann [Bachem et al., 1996].

Dieser Aspekt ist auch für den Arbeiteragenten denkbar, wir haben im Rahmen dieser Arbeit zwar eine Umplanung im Arbeiteragenten implementiert, diese ist jedoch intern auf diesen Arbeiter beschränkt. Wenn ein solches Umplanen auch von Arbeiteragent zu Arbeiteragent möglich wäre, könnte man deren Auslastung sicherlich noch weiter verbessern.

Im Hinblick auf die Anwendungsmöglichkeiten des Systems ist eine Integration in das bestehende System *TeleTruck* am DFKI in Saarbrücken denkbar [Bürckert et al., 1998]. Das Softwarepaket *TeleTruck* plant innerhalb einer Spedition die Routen der einzelnen Fahrer in Abhängigkeit des bisher erstellten Plans des Fahrers mit seinem LKW, der Ladekapazität sowie weiteren einschränkeden Bedingungen. Auch hier ist das Einbeziehen von Wünschen der Fahrer ein denkbarer Aspekt, der das gesamte System erweitern würde.

Literaturverzeichnis

[Aiba, 1991] Aiba, K. S. . A. (1991). Computing soft constraints by hierarchical constraint logic programming. Technical Report ICOT Technical Report: TR-610, ICOT. Institute for New Generation Computer Technology.

[Alan Borning and Wilson, 1989] Alan Borning, Michael Maher, A. M. and Wilson, M. (1989). Constraint hierarchies and logic programming. In *Proceedings of the Sixth International Logic Programming Conference*, pages 149–164, Lisbon, Portugal.

[Bachem et al., 1996] Bachem, A., Hochstättler, W., and Malich, M. (1996). The simulated trading heuristic for solving vehicle routing problems. *Discrete Applied Mathematics*, 65(1-3):47–72.

[Badros and Borning, 1998] Badros, G. and Borning, A. (1998). The cassowary linear arithmetic constraint solving algorithm: Interface and implementation. Technical Report TR-98-06-04.

[Balzert, 1998] Balzert, H. (1998). *Lehrbuch der Software-Technik, Bd. 1.: Software-Entwicklung*. Lehrbücher der Informatik, Spektrum, Akad. Verlag, Heidelberg.

[Barták, 1998] Barták, R. (1998). Inter-hierarchy comparison in hclp. In *Proceedings of PAP/PACT '98*, pages 461–474, London.

[Bond and Gasser, 1988] Bond, A. and Gasser, L. (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, Los Angeles, CA.

[Borning et al., 1996] Borning, A., Anderson, R., and Freeman-Benson, B. N. (1996). Indigo: A local propagation algorithm for inequality constraints. In *ACM Symposium on User Interface Software and Technology*, pages 129–136.

[Bürckert et al., 1998] Bürckert, H.-J., Fischer, K., and Vierke, G. (1998). The TeleTruck System. Research Report, DFKI.

[Chaib-draa and Levesque, 1994] Chaib-draa, B. and Levesque, P. (1994). Hierarchical models and communication in multi-agent environments. pages 119–134, Odense, Denmark.

[Cohen, 1995] Cohen, P. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press, Cambridge, MA.

[Dantzig, 1963] Dantzig, G. (1963). *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey.

[Dechter and Frost, 1998] Dechter, R. and Frost, D. (1998). Backtracking algorithms for constraint satisfaction problems; a survey.

[FIPA, 2001a] FIPA (2001a). Fipa abstract architecture specification. http://www.fipa.org/specs/fipa00001/XC00001J.html.

[FIPA, 2001b] FIPA (2001b). Fipa agent management specification. http://www.fipa.org/specs/fipa00023/XC00023H.html.

[FIPA, 2001c] FIPA (2001c). Fipa agent message transport service specification. http://www.fipa.org/specs/fipa00067/XC00067D.html.

[FIPA, 2001d] FIPA (2001d). Fipa communicative act library specification. http://www.fipa.org/specs/fipa00037/XC00037H.html.

[FIPA, 2001e] FIPA (2001e). The fipa contract net interaction specification. http://www.fipa.org/specs/fipa00029/XC00029F.html.

[Fischer et al., 1994] Fischer, K., Müller, J. P., and Pischel, M. (1994). Unifying control in a layered agent architecture. Technical report, DFKI.

[Fishman, 1986] Fishman, G. (1986). A comparison of four monte-carlo methods for estimating the probability of s-t conectedness.

[Georgeff and Ingrand, 1990] Georgeff, M. and Ingrand, F. (1990). Real-time reasoning: The monitoring and control of spacecraft systems.

[Gerber and Klusch, 2001] Gerber, A. and Klusch, M. (2001). Casa: Agents for mobile integrated commerce in forestry and agriculture. In Proceedings of the Workshop of the seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001) on E-Business and the Intelligent Web.

[Guide, 2001] Guide, F.-O. D. (2001). Fipa-os developers guide. citeseer.nj.nec.com/477218.html.

[Hannebauer, 2000] Hannebauer, M. (2000). Their problems are my problems - the transition between internal and external conflict.

[Henz et al., 1993] Henz, M., Smolka, G., and Würtz, J. (1993). Oz - a programming language for multi-agent systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 404–409, Chambery. Morgan Kaufmann Publishers, Inc.

[Hiroshi Hosobe, 1996] Hiroshi Hosobe, Satoshi Matsuoka, A. Y. (1996). Generalized local propagation: A framework for solving constraint hierarchies. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*.

[Hosobe,] Hosobe, H. Theoretical properties and efficient satisfaction of hierarchical constraint systems.

[J. Blazewicz, 1993] J. Blazewicz, K. H. Ecker, G. S. J. W. (1993). *Scheduling in Computer and Manufacturing Systems*. Springer-Verlag.

[Kumar, 1992] Kumar, V. (1992). Algorithms for constraints satisfaction problems: A survey. *The AI Magazine, by the AAAI*, 13(1):32–44.

[Mackworth and Freuder, 1985] Mackworth, A. and Freuder, E. (1985). The complexity of some polynomial consistency algorithms for constraint satisfaction problems.

[Mackworth, 1977] Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118.

[Menezes et al., 1993] Menezes, F., Barahona, P., and Codognet, P. (1993). An incremental hierarchical constraint solver. In Kanellakis, P., Lassez, J.-L., and Saraswat, V., editors, *PPCP'93: First Workshop on Principles and Practice of Constraint Programming*, Providence RI.

[Nemhauser and Wolsey, 1988] Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*. John Wiley and Sons, 1988. Wiley Interscience Series in Discrete Mathematics and Optimization.

[Neveu, 1996] Neveu, B. (1996). Houria iii: A solver for hierarchical systems of functional constraints. planning the solution graph for a weighted sum criterion.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[Sannella, 1994] Sannella, M. (1994). SkyBlue: A multi-way local propagation constraint solver for user interface construction. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 137–146, Marina del Rey, CA, USA.

[Sannella et al., 1993] Sannella, M., Maloney, J., Freeman-Benson, B. N., and Borning, A. (1993). Multi-way versus one-way constraints in user interfaces: Experience with the deltablue algorithm. *Software - Practice and Experience*, 23(5):529–566.

[Schillo and Klein, 2001] Schillo, M., F.-K. and Klein, C. (2001). Multi-agent based simulation: Second international workshop on multi-agent based simulation.

[Searle, 1969] Searle, J. R. (1969). *Speech Acts*. Cambridge University Press.

[Seidel, 1996] Seidel, R. (1996). *Vorlesungsskript Optimierung*.

[Smith, 1979] Smith, R. G. (1979). The contract net protocol: High-level communication and control in a distributed problem solver. In *Proceedings of the 1st International Conference on Distributed Computing Systems*, pages 186–192, Washington, DC. IEEE Computer Society.

[Smolka, 1998] Smolka, G. (1998). Concurrent constraint programming based on functional programming. In Hankin, C., editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, vol. 1381, pages 1–11, Lisbon, Portugal. Springer-Verlag.

[Weiss and Sen, 1996] Weiss, G. and Sen, S., editors (1996). *Adaption and Learning in Multi-Agent Systems*. Springer-Verlag.

[William D. Harvey, 1995] William D. Harvey, M. L. G. (1995). Limited discrepancy search. In Mellish, C. S., editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95); Vol. 1*, pages 607–615, Montréal, Québec, Canada. Morgan Kaufmann, 1995.

[Wilson and Borning, 1993] Wilson, M. and Borning, A. (1993). Hierarchical constraint logic programming. Technical Report TR-93-01-02.

[Wilson, 1993] Wilson, M. A. (1993). *Hierarchical Constraint Logic Programming*. Technical report 93-05-01, Dept. of Computer Science and Engineering University of Washington.

[Wooldridge and Jennings, 1995] Wooldridge, W. and Jennings, N. R. (1995). *Intelligent Agents*. Springer-Verlag.