Self-Adaptation and Scalability in Multi-Agent Societies

Christian Gerber

Dissertation

zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Technischen Fakultät
der Universität des Saarlandes

Saarbrücken, Dezember 1999

Kurzzusammenfassung

Die vorliegende Dissertation präsentiert Selbstadaptierungsmechanismen zur Effizienzoptimierung und Skalierbarkeit von Multiagentensystemen. Zur Umsetzung der Mechanismen wird eine Agentenentwicklungsumgebung vorgestellt, die den Entwurf von selbstadaptierenden Agentengesellschaften erleichtert. Weiterhin wird gezeigt, wie es durch eine Modifikation der Autonomie der Mitglieder eines Agentensystems möglich ist, den Adaptierungsansatz zu verfeinern, um so *holonische* Agentengesellschaften zu konfigurieren, bei denen die Mitgliedsagenten Teile ihrer Autonomie aufgeben und eine Gesellschaft nach außen wie ein einzelner Agent auftritt. Sowohl für den allgemeineren Fall als auch für den Fall einer holonischen Agentengesellschaft werden in dieser Arbeit Industriefallstudien herangezogen, die die Anwendbarkeit des Ansatzes untermauern.

Abstract

This thesis presents methods to allow multi-agent systems to configure themselves to any application scale and nature. This self-adaptation is the key to achieve and maintain efficiency and scalability in multi-agent societies. Theoretical foundations are elaborated and then realized in an agent development environment that allows the convenient design of self-adapting agent societies. If the autonomy of the members of a society can be changed significantly, this society can be perceived as a single entity - a *holon*. The self-adaptation approach is applied to configure a holonic agent society. A collection of very different case studies is chosen to evaluate the presented approach, in both the more general case, and the specialized holonic case.

Zusammenfassung

Skalierbarkeit ist ein wichtiger Aspekt für das Design von Software jeglicher Art: Ein System, das in einer kleinen Umgebung effizient arbeitet, kann in einer großdimensionierten Umgebung versagen, wenn beispielsweise das Laufzeitverhalten der verwendeten Algorithmen exponential zur Größe der Umgebung ist. Daher ist die sorgfältige Untersuchung des Laufzeitverhaltens und der auftretenden Problemen beim Programmieren im Großen von besonderer Wichtigkeit.

Multiagentensysteme bieten diesbezüglich mehrere Besonderheiten: Viele Multiagentensysteme sind *offen* konzipiert, d.h., neue Komponenten (sowohl Softwareteile, wie z.B. Agenten, als auch neue Hardware) können zur Laufzeit jederzeit in das System integriert werden. Weiterhin sind viele Multiagentensysteme nicht konzipiert, um zu einem bestimmten Zeitpunkt zu terminieren, so etwa die permanente Kontrolle eines Echtzeitsystems rund um die Uhr. Durch die Offenheit und die dadurch resultierende Veränderung der Systemumgebung kann jedoch ein System, das ursprünglich sehr effizient kalibriert wurde, über die Zeit seine Effizienz verlieren. Die Erhaltung von Effizienz ist daher ein wichtiges Thema und eine Voraussetzung für den erfolgreichen Einsatz von Multiagentensystemen.

Der wissenschaftliche Beitrag der vorliegenden Dissertation liegt in der Entwicklung von Mechanismen, die eine Selbstadaptierung von Multiagentensystemen ermöglichen. Diese Selbstadaptierung ist der Schlüssel, um Skalierbarkeit und Effizienzerhaltung von Multiagentensystemen zu gewährleisten.

Diese Arbeit konzentriert sich auf Gesellschaften von benevolenten Agenten. Hierbei ist die Existenz von mindestens einem gemeinsamen Ziel aller Agenten innerhalb der Gesellschaft Voraussetzung für den Einsatz des in der Arbeit vorgestellten Selbstadaptierungsmechanismus *GRAIL (**G**eneric **R**esource **A**llocation & **I**ntegration a**L**gorithm)*, der alle Mitgliedsagenten der Gesellschaft integriert.

Der GRAIL-Mechanismus basiert auf der effizienten Allokation von Ressourcen. In diesem Zusammenhang wird der Begriff der *beschränkt-rationalen Agentengesellschaft* als Erweiterung des beschränkt-rationalen Agenten eingeführt: Es wird nicht nur das Verhalten aller Agenten nach ihren individuellen Ressourcen optimiert, sondern auch durch eine höhere Kontrollinstanz eine möglichst effiziente Ressourcenverteilung auf der Gesellschaftsebene angestrebt. In GRAIL werden eine oder mehrere solcher Instanzen durch spezielle *Monitoragenten* implementiert.

Auch der Ressourcenbegriff wird für den Multiagentenfall erweitert: eine *abstrakte Ressource* beschreibt eine Entität der Umgebung einer Agentengesellschaft, deren Benutzung für die Mitglieder der Gesellschaft einen Vorteil beim Erreichen ihrer Ziele darstellt und dabei Abhängigkeiten zwischen den Mitgliedsagenten modelliert.

In der Arbeit werden Methoden zur Ressourcenallokation präsentiert und diskutiert: hierbei wird diese Allokation als ein Optimierungsproblem angesehen, dessen Zielfunktion durch die Performanz des Systems definiert wird, während der mehrdimensionale Suchraum die Menge aller möglichen Ressourcenkonfigurationen darstellt. Dabei steht jede Ressource für eine Dimension im Suchraum.

Zur Umsetzung des GRAIL-Ansatzes wird eine Agentenentwicklungsumgebung *SIF (**S**ocial **I**nteraction **F**ramework)* zur einfachen Generierung von Agentensystemen vorgestellt und zu einem Werkzeug *SIFIRA (**S**ocial **I**nteraction **F**ramework for **I**ntegrated **R**esource **A**daptation)* erweitert, das den Entwurf von ressourcenadaptierenden Agentengesellschaften erleichtert. Auf diese Weise wird das Paradigma des *agenten-orientierten Programmierens* in Richtung eines *ressourcen-orientierten Programmierens* erweitert. Beide Umgebungen werden ebenfalls in der Arbeit vorgestellt.

Zur Verwendung des Selbstadaptierungsmechanismus ist der Aspekt der *Agentenautonomie* von hoher Relevanz: Durch eine Modifikation der Autonomie der Mitgliedsagenten einer Gesellschaft ist es möglich, den GRAIL-Ansatz weiter zu verfeinern und auf *holonische* Agentengesellschaften anzuwenden: in einer solchen Gesellschaft geben Mitgliedsagenten Teile ihrer Autonomie auf und vereinigen sich zu einem *Holon*, das - zumindest nach außen - wie ein einzelner Agent auftritt. Ein Holon ist selbstähnlich strukturiert, da die Mitgliedsagenten ihrerseits selbst Holonen sein können. In der vorliegenden Arbeit wird dieses Phänomen genauer untersucht: Es werden Merkmale für holonische Agenten und Anwendungen herausgearbeitet und mit Eigenschaften herkömmlicher Agentensysteme verglichen. Weiterhin wird gezeigt, wie Ressourcenverteilung im Allgemeinen und GRAIL im Speziellen geeignet sind, holonische Agentensysteme zu konfigurieren.

Sowohl für den ersten, allgemeineren Fall einer kooperativen Agentengesellschaft, als auch für den Fall einer holonischen Agentengesellschaft werden in dieser Arbeit Industriefallstudien herangezogen, die die Anwendbarkeit des GRAIL-Ansatzes untermauern: In der ersten Fallstudie wird das mobile agentenunterstützte Informationssystem MoTiV-PTA auf Verhalten im Großeinsatz untersucht und gezeigt, dass GRAIL auf dieses Szenario anwendbar ist und zu einer

Erhöhung der Skalierbarkeit des Systems führt. Weiterhin wird der GRAIL-Ansatz an zwei Fallstudien für holonische Agentengesellschaften, dem Transportplanungssystem TeleTruck und dem Fertigungssystem IFMS demonstriert. Für beide Systeme wird gezeigt, wie GRAIL eingesetzt werden kann, um eine Selbstorganisation innerhalb eines Holons zu ermöglichen.

Preface

Scalability is a crucial aspect for the design of software of any kind: A system that runs efficiently in a small environment may fail in a large size environment, if for instance the run time behavior of the system is exponential to the size of the environment. Therefore, the thorough examination of the problems of programming in the large is inevitable.

Multi-agent systems exhibit several peculiarities: Many multi-agent systems are designed to be *open*, in the sense that new components (software components, for instance agents, as well as hardware components) may be added to the system at any time during its run. Furthermore, many systems are not designed to terminate at a particular time point, and a system which was originally adjusted to work at a very high performance level has to react dynamically to new inputs so that it might lose its performance over time as the environment changes. Maintenance of high performance is therefore a vital issue for the design of large-scale agent societies

The scope of this work focuses on scalability of multi-agent systems and on the development of methods to allow multi-agent systems to configure themselves to any application scale and nature. This self-adaptation is the key to achieve and maintain efficiency in multi-agent societies.

The thesis presents the self-adaptation scheme *GRAIL (**G**eneric **R**esource **A**llocation & **I**ntegration a**L**gorithm)* for a society of benevolent agents. The concept of a *bounded-rational agent society* is introduced as an extension of a bounded-rational agent: in such a society, agents optimize their behavior to their given individual resources, and in addition, higher-level control instances optimize the allocation of societal resources. In GRAIL, such control instances are implemented through special *monitor agents.*

The GRAIL mechanism bases on the efficient allocation of resources. For this purpose, the resource concept is extended to the multi-agent case: an *abstract resource* describes an entity of the environment of an agent society, which expresses inter-dependencies among the society members and whose use is to the advantage of the members.

In this thesis, methods for resource allocation are presented and discussed. Here, the task to organize such a society of artificial agents is perceived as an optimization problem by characterizing a multi-dimensional *search space* and an *objective function* to be optimized. The objective function has to denote the system's performance while the search space must describe the system's set of possible configurations.

These theoretical foundations are implemented in an agent development environment. The *SIF (**S**ocial **I**nteraction **F**ramework)* system supports rapid prototyping of efficient multi-agent scenarios while its extension *SIFIRA (**S**ocial **I**nteraction **F**ramework for **I**ntegrated **R**esource **A**daptation)* is designed to allow a convenient design of self-adapting agent societies. Both frameworks are described in this thesis.

One deliverable for the design of GRAIL has been generality: the GRAIL approach makes only few assumptions on the nature of the agents in the society. In particular, the autonomy of the member agents is modified as little as possible. If however, the autonomy of members of a group, or of even all society members can be changed more significantly, the GRAIL approach can be refined to a *holonic* agent society: in such a society, members give up part of their autonomy and unite to a *holon*, which appears to the outside as one single entity. A holon is organized in a self-similar way, where member agents can be holons themselves. This thesis examines this issue in detail; in particular, properties of holonic agents are elaborated and compared to properties of regular agents. Furthermore, it is demonstrated how resource allocation and in particular the GRAIL approach can be used to configure holonic agent systems.

A collection of very different case studies is chosen to evaluate the GRAIL approach, in both the more general case, and the specialized holonic case: In the first case study, the mobile information system MoTiV-PTA is examined on its behavior in large-scale applications and it is shown that GRAIL is suitable for this scenario and that its use leads to an increased scalability. Furthermore, the GRAIL approach is applied to two case studies for holonic agent societies, the transportation scheduling system TeleTruck, and the flexible manufacturing system IFMS. For both systems, the applicability of GRAIL for the self-adaptation of a holon is demonstrated.

Statement

Work on the general GRAIL scheme has been published in [Ger97, Ger98a]. Some aspects are the result of joint work with Christoph Jung [GJ97, GJ98]. Extensions have been published in [LJG99].

The standard SIF system has been derived together with Michael Schillo, Petra Funk and Jürgen Lind [FGLS98]. A detailed system description and user's guide is given in [SLG+99]. Extensions have been published in [JLG+99, JLG+00] and [Ger98a].

The holon framework, elaborated in cooperation with Gero Vierke and Jörg Siekmann, has been published in [GSV99a, GSV99b].

The work on the MoTiV-PTA case study has been published in [GSB99, Ger99]. Results in the TeleTruck domain have been published in [FGCd+99, GRV99]. As joint work with Thorsten Bohnenberger and Klaus Fischer, IFMS has been presented in [BFG99a, BFG99b].

Some future extensions to the GRAIL approach have already been discussed in [GI98, Ger98b].

Acknowledgments

First of all I would like to thank my thesis adviser Prof. Jörg Siekmann, not only for providing this research opportunity and environment to me, but also for raising my interest in artificial intelligence and guiding my academic career over more than eight years, ranging from term projects over Master's thesis to this dissertation, including two student and research exchanges in the United Kingdom and the United States of America.

I am grateful to the responsible officers of the Fulbright exchange program, in particular the staff of the German Fulbright commission who made my stay at Rutgers University possible which not only served as a preparation for my dissertation endeavor, but also enriched my understanding of the American culture. Many thanks to Prof. Alex Borgida who was my adviser during this year.

I am also indebted to my cooperation partners from Siemens AG, not only for funding this work, but also for their friendly support over the last three years: The topic of this work was originally motivated by Dr. Kleinschmidt; I am grateful to Dr. Donald Steiner for the advice and fruitful discussions, and to Dr. Bernhard Bauer who implemented the MoTiV-PTA prototype and never got tired of maintaining the tips and tricks-hotline from Munich.

I would like to thank all my colleagues at the multi-agent systems group at DFKI for the very fruitful research environment; in particular I am grateful to Dr. Klaus Fischer for his advice and support, and to Christoph Jung and Gero Vierke for the great collaboration and for the friendly and productive office atmosphere. I am indebted to Dr. Hans-Jürgen Bürkert for his very valuable advice on this thesis and on organizational issues at DFKI. Furthermore, I would like to thank Thorsten Bohnenberger, Andreas Gerber, Christian Ruß, and Michael Schillo for their implementational support on various systems, Petra Funk and Jürgen Lind for experiences in the framework of social interaction, and of course Walter Bieniossek for his commitment and system support, even on weekends.

Jörg Siekmann, Hans-Jürgen Bürkert, Gero Vierke, Andreas Gerber, Alex Borgida, and Jay League have read draft versions of this thesis. I am grateful for their constructive suggestions.

The greatest debts of gratefulness I owe my parents, my brother, and my girl friend Anja Weisbrodt who all have given me encouragement and support in every life situation.

This thesis is dedicated to my father.

Contents

List of Figures

Chapter 1

Introduction

Scalability is crucial for the design of software systems as it is well known that programming in the large differs significantly from small or medium size programming: the design and structure of a small system may not be appropriate at all for a very large if for instance the run times of the employed algorithms behave exponentially to environment size. Hence, an examination of the problems accompanying programming in the large is nowadays a core issue in software engineering. Unfortunately, in many cases the control flow in a very large system can become so complex that the system's run time cannot be described in an exact mathematical framework. Results from traditional complexity theory may therefore not always be appropriate as the complexity classes are of a granularity which is too coarse to be useful. So, other more empirically oriented approaches are asked for.

Investigating *large-scale multi-agent* applications is not only of scientific, but also of commercial interest: Multi-agent software is nowadays used for many applications, such as telematics, transportation planning, electronic commerce, manufacturing, etc. Even if a multi-agent system (MAS) runs perfectly in some test suites, it is not a priori clear that it will perform well in every day practice of very large scale applications. Many multi-agent systems have been designed to be *open*, in the sense that new components can be incorporated or removed from the system at any time. Components can either be new pieces of software, such as agents or information sources, or hardware, such as new computers in a network. Also, in many cases, multi-agent applications are intended to work ad infinitum. A system which was originally adjusted to work well, might loose its performance over time as the environment changes. *Maintenance of high performance* is therefore a relevant issue for the successful use of multi-agent technology in industrial applications.

Contribution of the Thesis

The contribution of this dissertation is the development of mechanisms for the self-adaptation of multi-agent systems, in particular, of applications with a very large number of agents. This self-adaptation is the key to achieve scalability of multi-agent systems.

In a multi-agent environment many kinds of societal settings can occur. Among others, systems can be described according to the following criteria:

- **Benevolence:** Agents of a society share the same goals or their goals conflict.
- **Heterogeneity:** All agents have the same architecture or there are agents of different types.
- **Openness:** New agents may or may not be allowed to enter or leave the system. New hardware components may or may not be added to the system during run time.
- **Autonomy:** Agents are fully autonomous in their decision making and action or they give up (some of) their autonomy.

This work focuses on a society of *benevolent* agents, where members do not betray each other and where the existence of a common overall goal for all members of an agent society allows for an adaptation scheme that integrates all members of the society. This scheme bases on the distribution of *abstract resources* that model representations of interdependencies between agents. The concept of a *bounded-optimal agent* [RW91, RS95] is extended to the notion of a *bounded-optimal agent society.*

Whereas heterogeneity and openness have only minor effects on the self-adaptation scheme to be presented in this thesis, a second, further refined scheme will be derived for agent societies in which member agents give up parts of their autonomy and form a greater entity, a *holon.* We extend the paradigm of *agent-oriented programming (AOP)* towards *holon-oriented programming* and *resource-oriented programming*, by proposing a generic agent development kit for building resource-adapting agent societies. Case studies from industrial projects demonstrate our approach for both, the general case of a fully autonomous agent society, and the refined case of a holonic agent society with members of restricted autonomy.

Overview of the Thesis

Chapter 2 places this dissertation within the field of distributed artificial intelligence (DAI). It gives an overview on existing agent definitions and describes the InteRRaP [Mül96] and MECCA [Ste92, SMH90] agent architectures, since they serve as underlying models of the agent systems of the case studies later in this work.

The chapter also surveys approaches to structure societies of natural and artificial agents, collected from organization theory, biology, social sciences and from DAI. These approaches are discussed and used to derive a collection of requirements for a generic self-adaptation scheme.

Chapter 3 presents a generalization of bounded rationality for multi-agent systems in which autonomous, but benevolent agents try to achieve a common goal. A ***G****eneric* ***R****esource* ***A****llocation and* ***I****mplementation a****L****gorithm (GRAIL)* is developed which links the members of an agent society through resource distribution. The introduction of societal structures provides the key for breaking the high complexity of a search for optimal solutions.

Chapter 3 also presents and discusses a variety of resource allocation mechanisms: Resource allocation is regarded as an optimization problem by characterizing a search space and an objective function to be optimized. The objective function has to denote the system's performance while a multi-dimensional search space must describe the set of possible resource configurations of the system: Each modifiable resource of the system reflects one dimension of the search space.

Chapter 4 implements the theoretical concepts of Chapter 3. The agent-oriented *Social Interaction Framework (SIF)* is presented whose key design aspects are rapid-prototyping, a broad implementation platform, the control of avatar agents by human users and easy access to the internal data of every agent in the simulation. A theoretical framework for the agent interaction is given as well as a detailed description of the Java^{TM}-based realization.

The extension *SIF with Integrated Resource Allocation (SIFIRA)* provides all facilities needed for constructing resource-aware agent societies: it allows for an explicit representation of resources, built-in resource-adaptation mechanisms and a clear separation of object-level agent tasks from meta-level issues that concern the structural adaptation. Both the theoretical foundation for the resource adaptation scheme, as well as the implementation are specified.

Chapter 5 proposes the *holonic agent* paradigm, in which agents consent to give up part of their autonomy and unite in order to form a "super-agent", a *holon*, that acts as a single agent when seen from the outside. A holonic agent is structured in a self-similar fashion; sub-agents may again be decomposed into holonic agents.

In this chapter, the entire spectrum of this new paradigm is explored, ranging from classification of possible application domains over definitorial issues to implementation aspects: Domains are examined on their suitability for holonic agents and general criteria are elaborated for the distinction between holonic and non-holonic domains. Requirements and characteristics of holonic agents in comparison to traditional agents are investigated. GRAIL is applied to holon management for holon formation and on-line re-configuration. Finally, the implementation of holonic agents in a collection of prototypical scenarios is sketched.

Chapter 6 presents a case study where the GRAIL approach is evaluated in an industrial application supported by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung; BMBF). The project *MoTiV (Mobilität und Transport im intermodalen Verkehr; Mobility and Transportation in Inter-modal Traffic)* [Sie95] is a united effort of major German car companies (e.g., BMW, Daimler-Chrysler, and Volkswagen) and supplier companies (Siemens, Bosch, etc.) to develop a travel planning support system where agents are used to represent traffic participants and service suppliers. Scalability is most important for the success of the project as the system is to be used (in the best case) by every traffic participant.

Using SIF, a suitable environment is created to simulate large-scale processes in a real-world environment, and then SIFIRA is applied to realize the GRAIL approach for this domain.

Chapter 7 evaluates the work of Chapter 5 on the basis of two case studies for holonic agent systems, the TeleTruck system [BFV98a, BFV98b], a fleet management system for the transportation domain and the *Intelligent Flexible Manufacturing System (IFMS)*, an agent-based approach for a dynamic re-scheduling of tasks on production units.

In the first study, a transportation company carries out transportation tasks which are given to the company dynamically anytime during the run of the TeleTruck system. The company has a limited set of trucks and its goal is to maximize its profit, i.e., to accept as many promising offers as possible given its restricted capacity. Furthermore, the optimal distribution of the load as well as optimal routes for each truck have to be found. The agent-oriented approach provides not only plan generation, but also monitoring of plan execution: if a step of the plan fails or if some new task is incorporated during the execution phase, a decentralized re-planning procedure is started on-line to adjust current plans to the new situation.

The IFMS approach has been developed in cooperation with experts from Daimler-Chrysler to design an optimal production flow for the automobile production business. IFMS employs two strategies to optimize production: an agent-based technique is used to monitor the production process and to re-schedule it in the case of failures. Furthermore, a high-level optimization of the plant topology is performed with respect to machine failures.

Chapter 8 summarizes the results of this dissertation. It gives an outlook to future work, based on the sociological concept of *generalized media of interaction* in order to extend our approach to societies of non-benevolent agents. Finally, the use of an evolution-based approach for resource allocation is discussed.

Chapter 2

Background

In this chapter, we provide the scientific background for this work. In the first part, we describe relevant aspects of results in distributed artificial intelligence, including selected agent definitions and architectures. In the second part, we refer to research results on building up structures and organizations. Here we present results from other sciences that originally used to have only little impact on artificial intelligence but which are of continuously growing relevance. We also present existing work in the MAS area. All these issues are then used as inspiration for a self-adaptation model for a society of artificial agents which we present in Chapter 3.

2.1 Distributed Artificial Intelligence and Multi-Agent Systems

Research on multi-agent systems has become an increasingly important area of research in artificial intelligence. The rather young community of researchers in *distributed artificial intelligence (DAI)* was founded in the early 1980's. The first international workshop was held in Boston in 1980, while the first larger journal release was a special issue of IEEE Transaction on systems [IEE81] in 1981. It was almost one decade later before the first workshop in DAI was held in Europe (titled *Modeling Autonomous Agents in a Multi-Agent World MAAMAW* in 1989.) It took another year before the first German workshop was organized.

In artificial intelligence, entities in a distributed environment are called *agents*. Research in DAI follows two main directions:

- **Multi-agent systems theory** addresses how the core of “intelligent” agents (e.g., their ability to interact with the environment and other agents, to represent and infer knowledge and to make decisions) has to be modeled in order to construct universal agents which can be used in as many domains as possible.

- **Distributed problem solving theory** focuses on the development of decentralized problem solving approaches by using agents as local decision making and executing units.

Although global optimality cannot be guaranteed, these methods have been proven useful: Even during the execution of a previously computed plan it is possible for the system to accept task changes and to adjust the current plan accordingly. Another aspect is robustness: If one participating unit fails another one may take its task, possibly with only small extra effort.

2.1.1 Models of Agency

There is a great variety of agent definitions in the literature, ranging from philosophically and sociologically inspired concepts to logical definitions and to definitions that focus on implementational aspects such as software architecture, efficiency, or tractability.

The sociologist Parsons [Par69] takes an *actor* to be an agent who has goals. In his definition, an *agent* is an individual who shows behavior. *Behavior* is the ability to change the state of the world. The *world* is differentiated into the agent itself and its environment. The *environment*, as it is perceived by the agent, defines the situation the agent is in. A *goal* is a certain state of the world. *To act* means to behave in such a way as to achieve a goal. In general, an agent can choose from a set of actions, on which he has certain expectations how they will change the world. An actor selects a specific action from his options according to his goals, the means at his disposal and his situation. Additionally, agents can use a common language in order to communicate with other agents.

Bratman [Bra87] connects three mental categories *belief*, *desire*, and *intention* by postulating certain requirements for mental capabilities of an intelligent agent which base on his analysis of rational human behavior on these categories. Based on this concept, Cohen and Levesque [CL90] and Rao and Georgeff [RG91] found a logical theory of belief, desire and intention (nowadays often called *BDI theory*) which ascribes these mentalistic notions to artificial agents as well.

Shoham [Sho91] characterizes the term agent as "an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments". He proposes a computational framework for *agent oriented programming (AOP)* that extends the object oriented programming paradigm by these "mentalistic" notions.

Russell and Norvig [RN95] define an agent as "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors."

Wooldridge and Jennings [WJ95] distinguish between *weak* and *strong* agency. They characterize a weak agent by the following traits: *autonomy* (the state of an agent is only influenced by its previous state and its perception), *social ability* (the agent communicates with other agents via a common language), *reactivity* (the agent responds to changes in their environment which it can perceive), and *pro-activeness* (the agent displays a goal-directed behavior based on deliberation in addition to their direct reaction to the environment).

Due to Wooldridge and Jennings, a stronger notion of agency includes *mobility* (the agent can move on a network), *veracity* (the agent does not communicate information that contradicts to its knowledge), *benevolence* (the agent does accept guidelines) or *rationality* (the agent acts in a way that, to the best of his knowledge, leads to one of its goals.)

Lange [Lan98] provides a more pragmatic definition that is oriented towards industrial demands: He defines an agent as a software object that has the following properties: *situatedness*, *reactivity*, *autonomy* with respect to its actions, and *pro-activity*. Furthermore, an agent should be continuously executing. Optionally, an agent can be communicative, mobile, believable or able to learn.

Our Perspective

In this work, we use a rather general notion of an agent as we derive a mechanism to be suitable for any agent architecture. For our purposes, we speak of an agent if the requirements for weak agency are fulfilled. In Chapter 5 however, we shall introduce the notion of a *holonic agent* which refines the traditional agent concept. In that chapter we shall provide a set of requirements for traditional agency which shall be extended to the holonic case.

There are many different agent architectures in the DAI literature that implement the above models (see [JW98] for an overview). In this work, we focus on two architectures: In Chapter 5, we shall use the INTERRAP agent architecture as a basis for holon-oriented programming. The case study of Chapter 6 employs agents of the MECCA architecture. So we briefly describe these two architectures.

2.1.2 INTERRAP

INTERRAP (*INTEgration of Reactivity and RAtional Planning*) [Mül96] is an agent architecture implemented in the programming language Oz [SSR95]. An

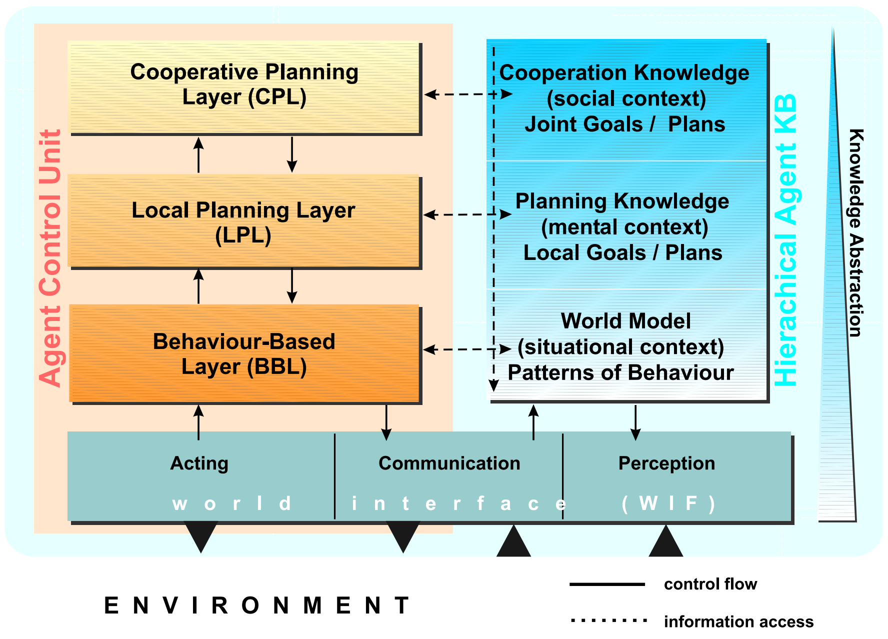

Agent Control Unit
Cooperative Planning Layer (CPL)
Local Planning Layer (LPL)
Behaviour-Based Layer (BBL)
Cooperation Knowledge (social context) Joint Goals / Plans
Planning Knowledge (mental context) Local Goals / Plans
World Model (situational context) Patterns of Behaviour
Hierachical Agent KB
Knowledge Abstraction
Acting
Communication
Perception
world interface (WIF)
ENVIRONMENT
control flow
information access

Figure 2.1: The INTERRAP agent architecture

important feature of the INTERRAP agent architecture is the combination of reactivity with goal-directed behavior. This is achieved by modeling the different types of behavior in different layers.

In INTERRAP an agent (Figure 2.1) consists of its *World Interface*, where its perception and action is modeled, a *Knowledge Base* (KB) and a three-layered control unit. This unit consists of the *Behavior-Based Layer* (BBL), the *Local Planning Layer* (LPL) and the *Cooperative Planning Layer* (CPL).

According to these three layers, the knowledge base of an agent is partitioned into three units: a *world model*, a *mental model* and a *social model*. The BBL is designed to give the agent the ability to react quickly to exceptional circumstances and to cope with routine situations. Such reaction is triggered by processing information from its world interface and world model. The purpose of the LPL is to enable the agent to create long-term plans. This is achieved not only by using the world model of the agent, but also by its mental models which reflect its intentions and goals. The CPL is responsible for creating *joint plans* with other agents. To do so, the CPL does not only use information on its world model and its mental model; it also processes information about goals, skills and commitments of other agents stored in its social model.

Mental states of an INTERRAP agent are described by the following components: the current *perception* ($\mathcal{P}$) of the agent, a set of *beliefs* ($\mathcal{B}$) describing its informational state, a set of *goals* ($\mathcal{G}$) and a set of *intentions* ($\mathcal{I}$) defining some target states of an agent: If such a state is reached, a given goal is fulfilled. Furthermore, in InteRRaP intentions define which *action* is performed next. Using these concepts, three basic functions for updating the mental states can be defined:

- *BR:* $\mathcal{P} \times \mathcal{B} \mapsto \mathcal{B}$ is a *belief revision and knowledge abstraction function*, deriving new beliefs from perception and old beliefs.
- *SG:* $\mathcal{B} \times \mathcal{G} \mapsto \mathcal{G}$ is a *situation recognition and goal activation function*, mapping the goals and beliefs of an agent into a new set of goals.
- *PS*: $\mathcal{B} \times \mathcal{G} \times \mathcal{I} \mapsto \mathcal{I}$ is a *planning and scheduling function*, deriving new intentions from current beliefs, goals and intentions.

In the above figure, the flow of control inside the model is shown. The functions *SG* and *PS*, evaluated in a certain layer, influence activities on that layer as well as on neighboring layers: The *situation recognition and goal activation* process results in the creation of new goals. These goals may trigger *planning and scheduling* processes on the same layer. Such a process generates new intentions by planning the steps which have to be taken to achieve a focused goal. After execution of these steps, the situation and goals relevant for the *SG* function of the next upper layer may have changed. Hence, the *SG* function on that level has to be executed. Applying these techniques leads to two basic flows of control:

- **Upward activation requests:** If a particular layer is not competent for solving a task (i.e., it cannot find a plan or schedule to achieve a given goal) it sends a request to the next upper layer. This layer may be able to solve the task since it has more access to the knowledge base and has keener planning facilities. The upper layer reports the result of its *PS* process to the lower one.
- **Downward commitment posting:** Whereas activation of layers is performed bottom-up, acting is organized in a top-down fashion: partial plans, derived at a given layer are posted to the underlying layer which has to integrate them into its schedule.

This leads to several problem-solving strategies: In a *reactive path* the situation is recognized in the BBL and directly addressed in that layer (example: avoiding a collision of two agents). A *local planning path* denotes a situation where a solution could not be found in the BBL, so that the LPL has to be activated in order to find a solution executed in the LPL (example: planning a transportation order). Finally, in a *cooperative planning path*, first the LPL had to be addressed by the BBL and the CPL had to be activated by the LPL. Once a joint plan has

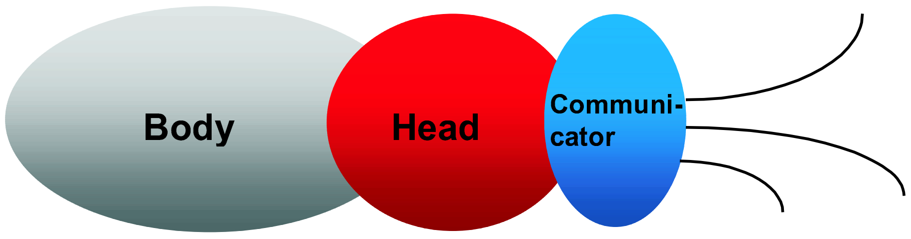


Figure 2.2: The MECCA agent architecture

been found it is processed in the LPL into a local plan which is then executed in the BBL (example: negotiation on a joint plan solving a blocking conflict).

Jung [Jun99] refines the InteRRaP architecture even further; in particular the control flow between the layers is now modeled in a much finer-grained way. However, for our case studies, the original architecture is still used as these latest developments are not of relevance there. Therefore we do not present them in detail.

2.1.3 MECCA

MECCA (*Multi-agent Environment for Constructing Cooperative Applications*) [Ste92, SMH90] is a Java^{TM}-based agent development environment for the construction of multi-agent systems using the *FIPA* (*Foundation for Intelligent Physical Agents*) [FIP98] standard.

One major focus in the development of MECCA has been to create a system which integrates human and artificial agents. Such an approach is classified under the paradigm of *Human Computer Cooperative Work (HCCW)*. A main trait of the HCCW paradigm is to regard human and artificial agents as entities of equal rights. Thus, a major requirement to such a unifying system is to provide adequate user interfaces and powerful and flexible cooperation methods.

MECCA comes with a library of pre-defined agent types. It also provides an *Agent Management System (AMS)* which allows the user to control the initialization and termination of agents. Furthermore, MECCA incorporates a yellow pages service called *Directory Facilitator (DF)*: agents register automatically at the directory facilitator upon creation which allows for retrieving agents with special names or service descriptions. For system monitoring, a special environment exists to view the performed communication between agents.

Furthermore, MECCA contains classes for describing *interaction protocols* based on a plan description language, and has a library of pre-defined negotiation protocols at its disposal.

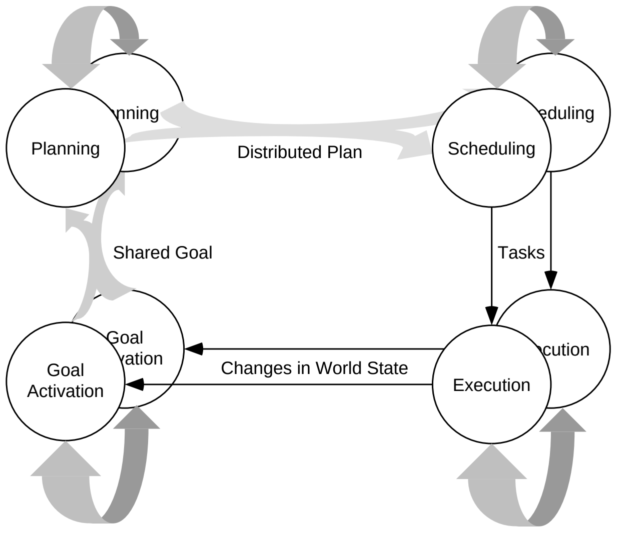


Figure 2.3: Problem solving with MECCA

Technical Details

The MECCA agent architecture consists of three parts (see Figure 2.2):

- **The body** is used to model basic problem solving capabilities of an agent. The body defines the application-specific functionality, for instance a database. This capability can be pursued by the agent body without an integration into any cooperation structure.
- **The head** enables an agent to participate in cooperative processes. It maintains the goals and plans of an agent and controls the body via the application interface. Moreover, the state of a negotiation is maintained.
- **The communicator** is responsible for the physical communication between agents via interaction protocols that employ local method invocation, TCP/IP or GSM. The communicator provides channel communication and information about other agents, (for instance their addresses), to the head of the agent.

Based on this partitioning, different components can be exchanged without major changes to the system, e.g., new protocols can be introduced without changing the existing MECCA system. Plans and interaction protocols can be constructed and executed by a plan interpreter in the MECCA system. In contrast to the InteRRaP model, where reactivity is a possible kind of behavior, a MECCA agent behaves strictly goal-oriented; problem-solving is realized in a four-step manner (visualized in Figure 2.3):

- **Initialization:** During this step, a goal is established for one or several agents. Negotiation between agents may be necessary to achieve this goal.
- **Planning:** During the planning phase, alternative action sequences are developed and evaluated in order to determine the optimal one.
- **Execution:** While performing and monitoring the planned actions they are monitored in order to recognize differences between the expected and current outcomes.
- **Evaluation:** Evaluating the result of a plan execution helps to determine weaknesses of current plans which may lead to new goals.

As mentioned above, one major focus of MECCA is to provide an integrating system for both, human and artificial agents. Hence, a sophisticated *cooperation model* is implemented as it is defined in the FIPA standard.

FIPA

FIPA is an international foundation whose goal is to develop standards for communication among agents to ensure inter-operability in industrial applications.[1] Currently, about fifty industrial and academic institutions world-wide are members of FIPA whose main goal is accomplished through an open international co-operation, the incorporation of existing standards as well as through verifying the usefulness of the specifications with concrete field trials.

The main focus of FIPA lies on the interface between the agents and the units of their environment, e.g., human beings, physical surrounding and existing software. Based on speech act theory [Aus62], the cooperation model of FIPA consists of two levels: *cooperation primitives* and *cooperation methods.*

Cooperation primitives are structured messages which are sent from one agent to another. A cooperation primitive consists of a keyword and a message contents where the keyword defines how to interpret the contents. Examples of keywords are `propose`, `accept`, `reject` or `refine`. The planning components of the agents treat primitives as actions, i.e., their semantics can be described by preconditions and effects. This may enable planners to reason about communication with other agents.

Cooperation methods can be built up by composing cooperation primitives to larger pieces. Methods then serve as well-defined communication protocols between agents. Using cooperation primitives allows to model cooperation methods such as *master-slave cooperation*, *contract net cooperation* or negotiations.

[1]For further details see `http://www.fipa.org`

[FIP98] progresses the FIPA specification along two dimensions: The *technical specification* and the *field trial specifications.* The technical parts address the following issues:

- **The agent management** does not only specify the management and administration of agents but also the entire agent system with agent name services, yellow pages with service descriptions and communication through firewalls.
- **The agent communication language** (FIPA ACL) defines the interaction between agents based on speech-act-based communicative acts and interaction protocols for information exchange, task distribution and negotiation.
- **The agent software integration** specifies how existing software can be interfaced to the agent system (using agent wrappers) and how agents are enabled to query an agent resource broker to locate, connect and use such software systems.
- **The agent management support for mobility** handles the states of mobile agents as well as their configuration and handling.
- **The agent security management** tackles security aspects concerning the communication between agents, agent authentication and trusted agent platforms based on existing standards.
- **The ontology service support** simplifies the specification of the semantics of domain specific content in messages exchanged between agents.
- **The human/agent interaction support** provides interfaces of different modalities to humans, as well as integration of user profiling and learning services.

The field trial specifications define the following applications based upon the technical specifications:

- **Personal travel assistance** (PTA) specifies a multi-agent system that provides individualized automated assistance in trip planning and on-route guidance. The MoTiV-PTA system addresses this issue as an application of the MECCA architecture. This system will be described in Chapter 6.
- **Personal assistant** (PA) specifies an electronic secretary agent that performs standard operations automatically for its user. The aim of the field trial is to provide a distributed appointment scheduling across different organizations using different personal schedulers.

- **Audio/video entertainment and broadcasting** bases on the increasing demand for efficient means of information filtering and retrieval, specifically for digital broadcasting networks such as the individual selection of programs.
- **Network management and provisioning** defines a framework for supporting dynamic virtual private networks, where a user can set up a multimedia connection with several other users. A dynamic market for telecommunication is assumed, so that contracts have to be negotiated at the time the service is delivered.

2.2 Structures in Natural and Artificial Societies

We shall now present different types of structures in industrial companies, human and animal societies. We also sketch work on structuring of groups of artificial agents. All these results are then used to point out relevant issues to a self-adaptation scheme for an agent society.

2.2.1 Business Administration: Organization Theory

Organization theory has developed basic organizational forms for companies (a survey is provided by Wöhe [Wöh81]). These forms provide a collection of "patterns", so organization managers have to decide which form to choose for a certain division of the company in question. Organizational forms developed in business administration may be used to model suitable substructures in a complex agent society. Wöhe defines a collection of possible organizational forms:

Single-line system: In a classic straight-line organization (as shown in the left side of Figure 2.4) every unit is only subordinated to one higher unit, as it is typical for example in military or in plan-based economies. Thus, tasks can be distributed vertically over the hierarchy.

The advantages of such a structure are a clear order of authority, transparency of the line of command and clear delineation of powers. Disadvantages lie in the length of that line of command, its inflexibility and in the danger of communication overload of intermediate and higher units.

Multi-line system: In this functional organization (displayed in middle part of Figure 2.4) a unit is subordinated to several higher units.

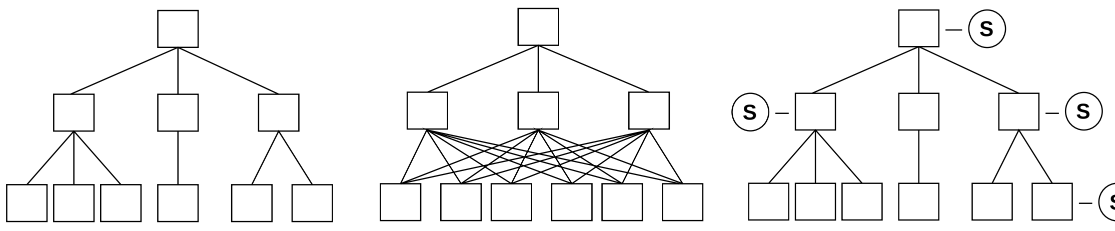


Figure 2.4: Single-line, multi-line, and staff-line systems

The pros of this approach lie in a fast and short information flow between the units and the possibility of specialization since this structure can be used to subordinate units to *several specialists*, each of whom, however, is only entitled to give directives restricted to his sphere of authority. The cons lie mainly in the unclarity of lines of authority and responsibility separation.

Staff-line system: This system is a modification of the single-line system: decision making units are augmented with a panel of experts who have consulting functionality but no decision making competence. The right part of Figure 2.4 models this system.

On the one hand, competence areas and the order of authority are clearly defined, on the other hand, problems may occur since the panel has strong influence on decisions it is not responsible for.

Divisional organization: In this type of organization a company is organized according to two criteria: business lines and subdivisions. Two single-line systems are combined: On the highest level the company is split according the business line criterion, on the lower levels according to the subdivision criteria. The left part of Figure 2.5 displays this approach.

Technically speaking, this model is a single-line system. Hence, it inherits all properties of such a system. In practice, however, for each product almost a whole independent enterprise has to be built up. This leads to decentralization, accompanied by the effect that subdivisions have to be constructed multiple times: e.g., in the left part of Figure 2.5 a sales department has to be built up twice. On the other hand, a company is represented in a clear fashion as no interweavings occur. This model is mainly realized in very large enterprises which can plan over a great product variety.

Matrix organization: This organizational form is also designed for integrating two organization criteria, business lines and subdivisions. In contrast to the model

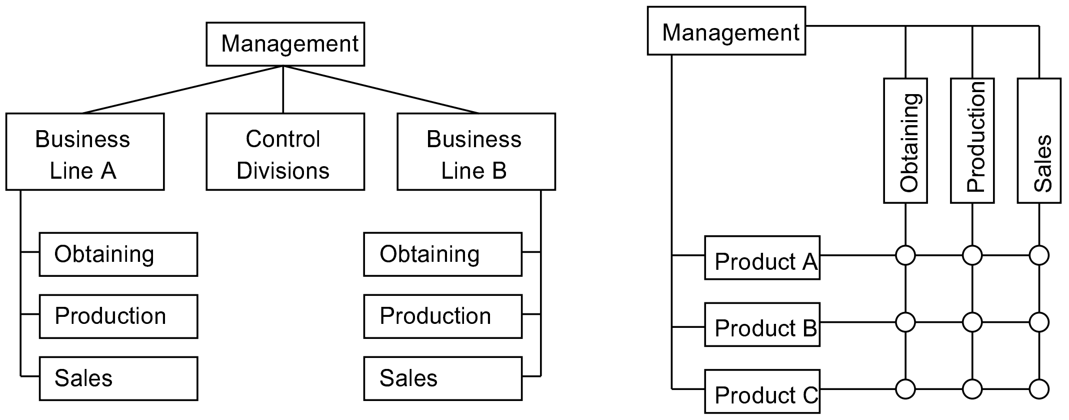


Figure 2.5: Divisional and matrix organizations

above, this organizational form is a variation of a multi-line system: every unit is subordinated into two hierarchies, one for each criterion. The right side of Figure 2.5 shows the correlations.

As an instance of a multi-line system, it inherits all its properties. In contrast to a divisional organization, no subdivision has to be built up twice which reduces structural overhead but may lead to competence confusion. After being en vogue in the 1980's as a favorite organizational form for very large enterprises, currently it is rejected since a clear order of authority is not visible to employees.

These organizational forms provide a collection of possible structure elements an enterprise consultant has at his disposal in order to form a company. Picot [Pic93] describes four additional organizational forms and characterizes their usage according to two properties of the task to be performed: Specificy and changeability of the task: in a *market*, unspecific tasks are processed in a standardized way. If the tasks processing varies quite often, a *strategic network* (where smaller units are clustered and communication between clusters is restricted) may be better used since it efficiently supports cooperation with internal or external suppliers or customers. Specific tasks are usually coordinated in a *hierarchical* way; if tasks vary frequently, *clans* (where each entity can communicate to any other) have shown useful. Figure 2.6 shows the classification.

2.2.2 Social Sciences: Group Behavior

The formation and development of human groups is one issue that social psychologists work on, inter alia trying to find the optimal size of a certain group. Furthermore, researchers in social psychology are interested in reasons why people form or join certain groups. Group theorists have been studying behavior

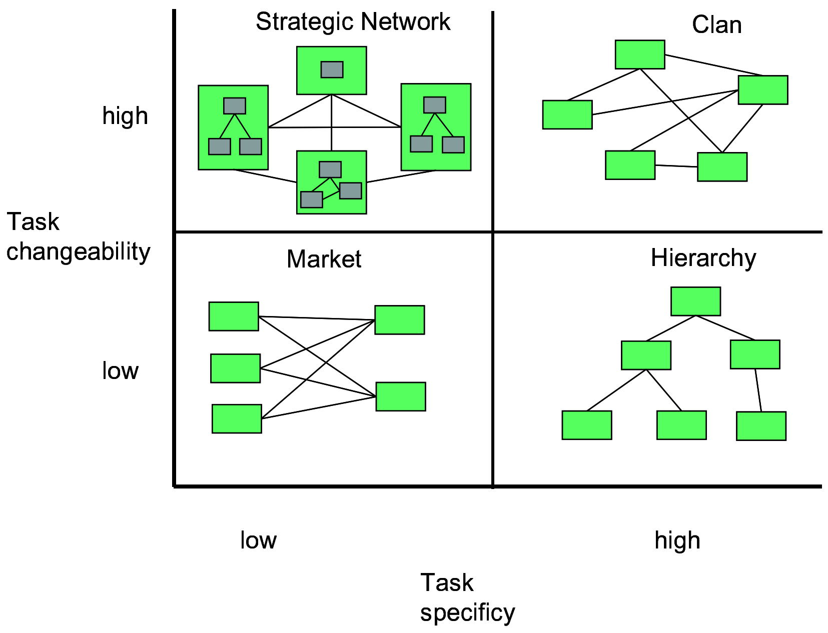


Figure 2.6: Four different organizational forms

of human groups that have to solve a given task. In particular, scientists are interested in determining for a certain task the size and degree of diversity by which a group performs best.

According to Moreland [Mor96] there are several paradigms that provide useful information about the best group size. First, *observations* of social interaction in public places have shown that the average group is quite small, containing just two or three people, and only few groups contain more than five or six people. Another popular paradigm involves *artificially creating groups* of different sizes and then noting which group experiences the fewest problems. Some researchers simply *ask* people to describe the ideal sizes for various types of groups. Experiments have shown that the optimal size of a cooperating group varies from five up to a dozen members (see [Nas88], [Sch89] or [CML93] for details), depending on the task and on properties of the test group.

These results are rather vague and they show that there is no simple way to determine the perfect size of a human group. Hence, a better approach is to study some of the correlates of group size: Moreland compares characteristics of larger groups to those of smaller ones: Generally speaking, larger groups enjoy several advantages: They have access to more resources, including time, money, and expertise, they tend to be more diverse. On the other hand, larger groups suffer from disadvantages: they often experience coordination difficulties which may decrease problem solving performance. Furthermore, there is more conflict among members of larger groups. In general, they are less willing to cooperate.

A great diversity of a group can improve its problem solving performance as specialists might be available to perform a certain task. The main risk of a great diversity is that it can produce conflicts among members. Moreland concludes that all these factors make it difficult to specify the optimal size or degree of diversity of a human group for pursuing a certain task. He suggests rather than worrying about the best group size and degree of diversity, it might be wiser to maximize the advantages and minimize the disadvantages of a group, whatever size that group has reached.

Tuckman and Jenson [TJ77] examine the formation of groups. They regard group formation as a continuous process moving through a five-stage life cycle: In the *forming stage*, members seek to orient themselves to the group. In the *storming stage*, members try to alter the group to satisfy their personal needs. In the *norming stage*, members endeavor to resolve the disagreements and tensions which threaten group success. In the *performing stage*, members attempt to maximize group performance and productivity. Finally, in the *adjourning stage*, members disengage from the group emotionally and behaviorally which leads to the end of the group life cycle.

2.2.3 Biology: Sociology of Insect Societies

Insect behavior and intelligence is structured rather differently from those of mammals: Insects seem to act much more socially in the sense that the prosperity of an individual appears to carry much less weight than the society as a whole. A single insect has a neglectable intelligence; however, the whole colony shows remarkable traits of intelligence. Understanding its cause has been and still is a major goal for scientists working in this field.

The examination of the sociology of animal societies, in particular of phenomena such as emerging functionality in insect societies, has lead to interesting insights into how a goal for such a society is split into subgoals and how they are achieved. Insect species can be classified by the degree of their social behavior: Wilson [Wil71] distinguishes between six different types of social behavior:

- **Solitary:** Adults do not care about their breed, do not live in a common hive and do not share labor.
- **Sub-social:** Adults care for their own nymphs or larvae for some period of time.
- **Communal:** Members of the same generation use the same common nest without cooperating in brood care.

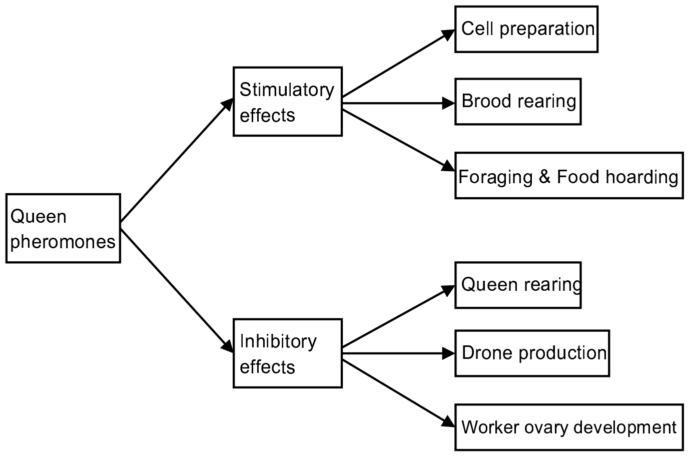


Figure 2.7: Effects of pheromone transmission

- **Quasi-social:** Members of the same generation use the same common nest and also cooperate in brood care.
- **Semi-social:** Members behave as in the quasi-social case, additionally there is also a reproductive division of labor, i.e., a worker caste cares for the young of the reproductive caste.
- **Eusocial:** Members behave as in the semi-social case, but there is also an overlap in generations so that offspring assist parents.

Of course, insects of the latter type are of the most interest for this work. The insect probably most investigated is the *Apis mellifera*, the honey bee (a survey can be found in [See85].) Typically, a honey bee colony consists of one queen, several thousand workers (between 5000 and sometimes more than 10 000) and drones whose population varies over the seasons, in summertime roughly 1500).

Starr [Sta79] explains bee behavior as a result of their genetic relationship: individuals prefer cooperating with close relatives rather than cooperating with other colony members.

In the past, some insect researchers (e.g., Wheeler [Whe11]) created the notion of a *super organism*: A colony is compared to a real body. Analogies which have caused this analogy are replication, import and export of materials, control of the inner environment, such as temperature, and finally, response and orientation to the external environment. Although the super organism concept failed in biology, mainly for the reason that in a metazoan body all cells are genetically identical which is not true for members of an insect colony, we will use it as an inspiration for the *holonic* agent society in Chapter 5.

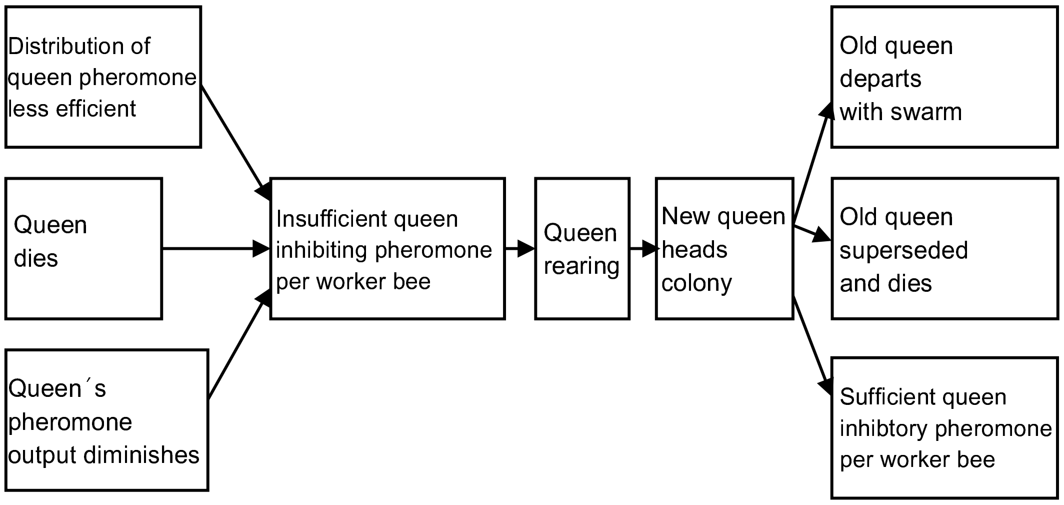

Distribution of queen pheromone less efficient
Queen dies
Queen´s pheromone output diminishes
Insufficient queen inhibiting pheromone per worker bee
Queen rearing
New queen heads colony
Old queen departs with swarm
Old queen superseded and dies
Sufficient queen inhibtory pheromone per worker bee

Figure 2.8: Pheromones regulating the rearing of honey bee queens

Furthermore, it has been found that colony members have individual goals which partly contradict to those of other members. Neither pursuing only one of these goals, nor adding them together would lead to a strategy allowing the colony to survive; the successful survival strategy is a result of a very complex communication and action behavior: Communication between queen and other colony members is realized via *pheromones*, i.e., chemicals that secreted by the queen and spread by workers as liquids or gas. Using pheromones, the queen influences the behavior of workers who might have goals inhibiting the queen's goals. Figure 2.7 gives an overview; a detailed description can be found in [Fre87]. This method allows the colony as a whole to react on changes of the internal and external environment, such as death of the queen, critical size of the colony, etc.

Feedback loops regulating the size and structure of insect societies have been developed to represent such phenomena. For instance, Free [Fre87] describes the cyclic influence of queen rearing on pheromone availability: in a colony that originally had a queen pheromone deficiency (due to the death or weakness of the queen) more queen larvae are being raised; the additional pheromones of these larvae re-establish the balance and an lead to the destruction of additional queen cells (Figure 2.8).

Such a feedback loop system has been proven very robust for achieving the prime goal of the colony, to survive. This approach, where centralized control meets decentral decision finding in a fashion that turns out to be very satisfying for all colony members, influences the design of an adaptation scheme for artificial agent societies in Chapter 3.

2.2.4 Models of Structured Artificial Agent Societies

By definition, multi agent system theory is concerned with effects of the interaction of a collection of autonomous agents. If the society of agents is very large, it is inevitable to introduce structures in order to keep the system feasible. In this section, we present structuring approaches that exist in the literature. Some of them focus on the pure description of structures while others concentrate on methods to introduce or optimize structures. A main distinguishing feature is the type of representation: structures in agent societies can be modeled implicitly or represented explicitly in the knowledge or belief of member agents.

In this context, the term *emergent behavior* is often used. Nagel [Nag61] defines a property of an object as emergent, if it is impossible to deduce this property from even a complete knowledge of the components.

Implicit Models of Agent Societies

Gasser et al. [GRHL89] view an agent *organization* as a set of questions on beliefs and actions of member agents. The authors believe that a structure should not be explicitly represented as a structural relationship between a set of agents. They argue that if agents experience "organization" they do not see global structures or fixed constraints. Rather, the agents' expectations concerning other agents define the organization.

Castelfranchi [Cas90] introduces implicit structures among agents by defining *power relations.* He defines the following relations which can then be used to derive a social structure for a set of agents:

- **Power of:** An agent α_1 has the power of a goal g if he somehow can reach that goal.
- **Dependence:** Agent α_1 depends on agent α_2 if α_1 does not have the power of a goal g while α_2 does.
- **Power over:** Agent α_1 has power over α_2 if α_1 can help α_2 to reach its goal or prevent α_2 from reaching the goal.

Chaib-draa [Cd96] introduces different structure degrees of *agent collectivities.* He classifies social collectivities into roles, groups and organizations. In this context, Chaib-draa identifies a *role* with the expected behavior of an agent. A *group* refers to a set of agents that consider themselves as a unit. In an *organization*, members are tied much closer to another; however they need not necessarily have to do something in common. In order to run a collectivity, norms and rules are needed. *Norms* are expectations of the group members on the appropriateness of certain actions. *Rules* have been derived in order to coordinate the activities of the members.

Doran and Palmer [DP95] describe an agent's *social model* of a society as its beliefs about the existence of agent groups, their structures, and the identities of their leaders and sub-leaders. According to Doran and Palmer, the social model of the entire society is not explicitly available; group membership is an emergent property of the behavior of the agents.

Explicit Models of Agent Societies

Werner [Wer89] develops an integrated view of agent *states*, *communication*, and *social structure* in a multi-agent society in order to describe the complex relationship between these issues in a uniform fashion. Among others, he defines a *social structure* as a set of social roles $\{r_1, \ldots, r_n\}$. A *social role* r_i characterizes an agent which is defined as the tuple $< I_r, S_r, V_r >$ of the state *information*, *permissions* and *responsibilities*, and *values* of that agent according to that role. Werner extends this definition by a temporal index which allows to express dynamics in such a structure.

Ferber and Gutknecht [FG98] also use roles in their framework. For them, a role is an abstract representation of an agent function within a group. A *group structure* is then defined as a tuple $S =< R, G, L >$ where R denotes a set of role identifiers. G is an *interaction graph* specifying valid interactions between two roles and the *interaction language* L is a formalization of the individual interaction definitions. Furthermore, Ferber and Gutknecht define a *organizational structure* as the set of group structures expressing the overall design $O =< S, Rep >$ where *Rep* is a representative graph where each edge points for one group structure to another one.

Tambe [Tam98] describes a formal model for *joint intentions* of members of an agent group by extending BDI theory. He defines that a *joint persistent goal* to achieve the completion of a team action p holds if and only if

- all team members mutually believe that p is currently false,
- all team members mutually know that they want p to become true eventually, and
- all team members mutually believe that until p is mutually known to be achieved, unachievable or irrelevant, they mutually hold p as a goal.

Tambe uses this framework to model groups in which all member agent share a joint intention. The system designer can hence model hierarchical or interleaving structures by introducing suitable joint intentions.

Group Formation Procedures

Numaoka [Num92] proposes communication between autonomous agents as an instrument for group configuration. In his model, an agent group is a *mapping* from a group name to a set of agents. If an agent wants to join a group, the current members run a simple voting procedure. Similarly, if an agent intends to leave a group the same voting mechanism is applied.

Ketchpel [Ket93] proposes a market-oriented approach for group formation. In this approach, originally single agents iteratively cluster to larger groups. Each iteration consists of four steps:

1. **Communication phase:** All agents (or agent groups) collect information in order to estimate the usefulness of possible partners.
2. **Calculation phase:** Each agent ranks the possible partners.
3. **Offer phase:** Based on that ranking, agents contact others and offer them to join together.
4. **Unification phase:** Agents commit the group formation.

Ketchpel evaluates the quality of a derived solution according to the criteria of *stability* (agents assign a high utility to a certain configuration) and *efficiency* (the coalition forming process is efficient in terms of computational cost and number of communication acts between agents).

Aeken and Demazeau [AD98] provide an approach to (re-) balance a hierarchy of agents. In their framework, *atomic* agents are always clustered in groups of two by the introduction of a new *composite agent* on an upper level. Such higher-level agents are recursively structured in the same way. However, not all atomic agents necessarily have to be positioned at the same depth of the graph, which may possibly lead to imbalanced tree structures. Aeken and Demazeau introduce a balance measure which bases on the *entropy* $E = \sum p_i log_2(p_i)$ of the system where p_i represents the probability of accessing an atomic agent when descending from the root of the tree, given a probability of 0.5 to descend in one of the two agents in each composite agent. The authors show for certain applications that a balanced agent society (i.e., one with a small entropy) performs better that an imbalanced one.

Goldman and Rosenschein [GR97] examine the evolving of organizational agent structures by mapping the problem domain to the well-known *Game of Life* [Gar83] domain. After the application of the Game of Life algorithm, the resulting pattern is mapped back to the original domain in order to describe a structure of the agent society.

2.2.5 Inspirations for a Self-Adaptation Scheme

For the development of a self-adaptation scheme, we put special emphasis on the following requirements:

- The environment and the agent system itself are open: new agents may enter the system, new constraints on the optimality of a configuration may be imposed by the environment, etc.
- Both the environment and the system are dynamic: Changes may occur at any time.
- The environment is unpredictable: in general, no probabilities of occurrences of certain phenomena can be assumed.
- The system is inherently complex.
- The setting consists solely of benevolent agents.
- The scheme shall be used in a universal way, in particular it shall not depend on one application domain or agent architecture.

The study of insect societies, in particular the topic of the super-organism and the pheromon transmission mechanism have inspired the holonic model of regarding a whole society as one entity from an outside perspective (Chapter 5). This area has also motivated the use of a feedback-loop as it will be presented in the next chapter: A central entity (such as the bee queen) sends guidelines that constrain the otherwise autonomous behavior of society members who may have partially conflicting goals.

Theories from organization theory provide structural basic components to be incorporated into our approach and exemplified in a case study in Chapter 7: A possible organization can be regarded as a point in a search space of all possible configurations. Using pre-defined larger modules (consisting of several atomic units) as basic entities for the topology design can dramatically reduce the search complexity.

Most approaches to structurize societies of artificial agents define some sort of roles or power relation between agents. This of course is inevitable, but such a role or relation needs to be more flexible than just an a priori tuple-based definition. There must be some sort of meaning behind a role definition. We consider a power relation to hold if one agent is able to constrain the behavior of another one. In the next chapter, we present our approach where such a power relationship is instantiated through the *distribution of resources*.

With respect to the above requirements, it is necessary to explicitly represent structures, since an implicit model puts requirements to the knowledge base of an agent, which contradicts to our aim to derive a universal approach. In contrast to some of the presented approaches, we do not restrict the structure to a tree where each node has only a limited number of children nodes.

Regarding our requirements, a societal structure needs to be flexible and modifiable due to changes in the environment. Therefore, we unify the *group formation* process with the *re-configuration* process. Whereas some of the presented group formation procedures are only able to form an initial society of agents, our approach also re-configures a society during the run of the system if needed. In contrast to some other approaches, we use a de-centralized technique, where several special agents are assigned to control the performance of different sub-sets of the agent society. These special agents can then re-configurate these sub-groups in a more local fashion which leads to the desired flexibility.

Bottom Line

In this chapter we have surveyed some scientific background for this thesis and presented a range of definitions for agency. We will call entity an agent if it fulfills Wooldridge and Jennings' requirements for weak agency. We have also sketched two agent architectures, InteRRaP and MECCA, which we will use later on.

We have collected approaches to structurize natural and artificial societies. We have derived a list of requirements for a society configuration scheme which we have used to compare existing approaches and to extract vital elements for our approach.

Chapter 3

A Generic Resource Allocation & Integration Algorithm (GRAIL)

This chapter presents the theoretical foundation for the self-adaption of agent societies. The chapter is structured as follows: First, we will characterize "scalability" and its correlation to resource adaptation. In the following section we will discuss different resource concepts and present an abstraction of that notion. The second part of the chapter shows the realization of a ***G**eneric **R**esource **A**llocation & **I**mplementation a**L**gorithm (GRAIL)*[1]. We present a collection of ***R**esource **A**llocation a**L**gorithms (RALs)* to decide how to assign resources, and we specify a ***R**esource **I**mplementation a**L**gorithm (RIL)* to distribute resources in an agent society.

3.1 Definition of Scalability

Bamberg and Baur [BB89] define scale as a *classification curve to assign numbers to certain entities*; three different types of scales can be distinguished:

- **A nominal scale** is used to simply distinguish between entities by assigning different numbers to them.
- **An ordinal scale** is a nominal scale where an order is introduced between the numbers representing entities. This order is used to sort entities with respect to a certain property.
- **A cardinal scale** is an ordinal scale where additionally a metric is introduced which allows to determine *how far* two entities differ according to the property in question.

[1]The word grail is from the old French *greal* meaning a kind of dish.

Furthermore, we can distinguish between *discrete* and *continuous* scaling dimensions: In a discrete dimension an entity can only be assigned to a value out of a discrete set of numbers whereas in a continuous dimension all possible values between two extreme points may be assigned.

The following definition of a scale has been taken from Meyers Konversationslexikon [Mey90], a German lexicon of the 19th century. This definition views the term "scale" from its historical perspective.

> Auf physikalisch-mathematischen Instrumenten wie z.B. Barometern, Thermometern etc. angebrachter Maßstab, bestehend aus einer in gleiche Teile geteilten geraden Linie, deren einzelnen Teile Grade genannt werden.[2]

Although this definition focuses on a physical environment, the main property of a scale can be seen from a more general perspective: the object in question can be divided into equal parts, hence, exact up- and down-sizing is possible. Thus, the concept *scalability* denotes the *possibility to exactly up- and down-size an object.*

In the field of software systems, the degree of scalability of a system architecture can be used to describe how its problem solving behavior reacts to resource modifications. This behavior can be measured by the introduction of a *performance function.* One may achieve *optimal scalability* if performance is directly proportional to the use of resources. However, optimal scalability can hardly be achieved in complex applications: in general, scaling up one or several resource quantities by some factor n does not imply a performance improvement by the factor $c * n$ for some constant c.

However, there may be an *optimal resource distribution* which can be viewed as the optimum of the performance function in a multi-dimensional search space where each scalable quantity denotes one of these dimensions. We shall now look at the concept of a "resource" more closely.

3.2 The Resource Concept

During the 1970's and 1980's, various theories on resources have been developed in psychology, mainly to explain performance decrease in human behavior due to cognitive resource restrictions. Overviews can be found in [Neu92] and [Rum96]. In the late 1980's research in AI stipulated that a cognitively adequate behavior

[2]Translated into English: *A range put on physical-mathematical instruments, such as barometers or thermometers; consisting of a straight line separated into equal parts which are called degrees.*

of artificial agents can only be achieved if their currently available resources are taken into consideration. Hence, models of resources have been derived in AI and cognitive science, but also in business sciences where the theory of the *economic principle* is the underlying foundation of economic theory.

In Section 3.2.1 we introduce the term *goods* from the area of economics. which nicely correlates to the resource concept and is hence used as one of the foundations for our definition of abstract resources. The subsequent sections give an overview of resource concepts in AI and cognitive science. In Section 3.2.5, we define our resource concept and illustrate it with a collection of examples.

3.2.1 Economic Theories on Limited Goods

According to Wöhe [Wöh81], rational behavior (in the economic sense) is due to the fact that humans usually have a large number of desires or needs while they have only a *limited supply of means* to satisfy the desires. In economic terms, these means are called *goods.*

Müller and Peters [MP82] state that from the perspective of an economical rational (human) agent, three criteria must be fulfilled, in order to perceive an entity as a good:

- The agent must have a desire.
- The agent must regard the entity to be useful to satisfy his desire.
- The entity must be under the agent's right of disposal.

In economy theory, goods are distinguished into *private* and *public* goods. Free goods are not subject to any quantitative limitations (e.g., the air to breathe). Private goods are available only in limited supply; the degree of satisfaction depends on every unit of the stock. Hence such goods are subject to trading between agents.

Depending on its nature, a good might be consumed (i.e., it vanishes after being used, such as fuel) or it might allow for repeated use (e.g., a hammer). Economic theory calls goods for single use *consumables* while multiply usable goods are called *consumer durables.* Special *regeneratable consumables* can be consumed, but they may regenerate (e.g., wood).

The economist Menger has set up an order of goods according to the criterion whether a good satisfies an agent's needs directly or indirectly. *Consumables* (e.g., food) are first-order goods, since they satisfy the need directly. *Production goods* are second-order goods if they are needed to produce first order goods (e.g., fertilizer). Inductively, production goods are n^{th}-order goods if they are needed to produce goods of order $n-1$.

In general, human desires are not satisfied by using one single good, but by a variety of different goods. Economy theory distinguishes between *complements* and *substitutes.* Two or more goods are complements if they have to be used *together* in order to satisfy the need of an agent (e.g., hammer and nails). Goods are substitutes, if one good can be replaced by another one in order to meet the same desire (for instance potatoes and rice).

If an agent has a limited stock of goods available for satisfying a number of desires, he has two options to follow the *economic principle*: he can either try to satisfy a maximum number of needs with the given resources or he can pursue a certain goal by minimizing the amount of used goods, weighted by a *utility function* that denotes the agent's preference structure for each quantity of each good.

3.2.2 A Resource Definition by Jameson and Buchholz

Jameson and Buchholz [JB98] characterize the term "resource" in a first step simply as *tools and sources for such tools* that help an agent to reach its goals. Typical types of resources are physical items, human capabilities, information, energy, and time. Jameson and Buchholz distinguish resources among three categories:

- **Consumable vs. usable resource:** Some resources may be *consumed*, i.e., each use of that resource decreases the amount of stock available. Other resources may be *used*, in principle arbitrarily often.
- **Purpose of use:** Resources may be used on purpose under certain considerations or they may be used unreflected and automatically (see also [Sch87]).
- **Decomposability:** Some resources can be subdivided into smaller portions (e.g., time) while other resources can only the used as a whole (e.g., the information about a phone number).

Jameson and Buchholz state that often some *restrictions* for the use of a resource apply, e.g., a maximum quantity of the resource per use. Furthermore, often the *cost* of using a resource has to be taken into consideration, i.e., the utility of using a resource not only depends positively on the success of an agent's action, but also negatively on the success of an agent's action if he had chosen another resource. This in fact is the exact definition of *economic costs* (see [Var87]). In order to measure success, *success metrics* and *success profiles* can be introduced:

- A success metric for a task is a measure for determining to what extent the execution of that task has been successful.
- A success profile for a task is a function denoting the dependency of the execution success of a task from the use of resources.

Often, the success of using a resource is not a priori clear; if it can be estimated, success profiles can express the *expected success* of the task execution.

3.2.3 A General Framework for the Use of Resources

Jameson [Jam97] sketches a very universal framework for resource allocation for a single agent: An agent has to perform a number of tasks $T_j \in \mathcal{T}$. To do so, the agent has a set of resources $R_i \in \mathcal{R}$ at its disposal. The success of an action depends, among other things, on the allocation of resources to the tasks. Hence, a resource allocation $A_{ij} \in \mathcal{R} \times \mathcal{T}$ is a relation between tasks and resources. For all resources, a set of possible *allocation expressions* must be specified. An allocation expression may have different meanings, for instance

- **a boolean value**, denoting whether an R_i has been allocated to T_j, or
- **a nonnegative real number**, denoting a certain amount of R_i allocated to T_j, or
- **a subset S_i of R_i** denoting that some elements of the resource have been allocated, provided that each R_i can be viewed as a set.

Hence, for a complete scenario, the meaning of all allocation expressions must be specified. Furthermore, restrictions or constraints of the resource allocation have to be defined: For boolean-valued allocation expressions there might be upper and lower thresholds for the overall number of tasks to be allocated. For real-valued allocation expressions the overall sum or product might be limited. For specifications of subsets of R_i there might be a restriction that no overlapping among these subsets should occur, etc.

For the specification of performance profiles a success metric M_j has to be defined: In the simplest case, a boolean value denotes the success or failure of an allocation. A value in $[0;1]$ may stand for a success probability of the allocation or for a quality value, etc.

According to Jameson this framework rests upon the following assumptions:

1. The success and the performance of a task allocation are completely independent from the success or performance of other task allocations.

2. There is no temporal order in the allocation of several resources.

3. This framework does not specify meanings of allocations, constraints or performance profiles. Since the framework is that general, these specifications are due to the "user" of the framework.

4. There are no mechanisms provided, *how* to actually do the allocation decisions. The "user" of the framework has to derive a solution on his own.

In particular the first assumption restricts the applicability of the framework, especially for the multi-agent case. In Section 3.5, we will show how to guarantee independence. We will also present a number of allocation procedures of our settings that implement solutions for the fourth issue.

3.2.4 Resource-sensitive Behavior

Once resources and their relations to the various tasks of an intelligent agent have been identified, it has to be specified or examined how these definitions influence the behavior of an agent. Wahlster and Tack [WT97] distinguish between three different types of *resource-sensitive* behavior, i.e., behavior of an agent that takes resource limitations under consideration.

- **Resource-adapted behavior:** The resource allocation of an agent has been optimized prior to the actual run of the system. If the environment changes in an unexpected way, the agent cannot modify its resource allocation.

- **Resource-adaptive behavior:** An agent pursues a given goal according to a pre-defined strategy, which is however, parameterized by resource allocation that can be dynamically adjusted to the current situation.

- **Resource-adapting behavior:** For a given problem setting, an agent has different problem solving techniques at its disposal. These techniques differ in the quality of the produced results and in the degree of resource consumption. During the run of the system, the agent reasons on how much of its resources to be spent by each problem solver.

The first type of resource-sensitive behavior is not desirable in large and open systems we assume in this work, since it is not flexible enough. Depending on the task an agent has to perform, resource-adaptive behavior might already be sufficient, but in many cases we need to ensure resource-adapting behavior.

3.2.5 A new Concept: Abstract Resources

We need a resource concept that subsumes different aspects of the above resource and generalizes the traditional resource notion in computer science, where resources are mainly *computational time* and *memory space.*

Russell et al. [RW91, RS95] distinguish between internal and external resources, between environmental and architectural constraints. Internal resources only affect the *application* of the agent program, while external ones are the subject of the agent's reasoning capabilities. This distinction between architectural and environmental constraints does not seem to be reasonable in the multi-agent case: *interdependencies among agents* affect a group or society of agents, regardless if these dependencies are of internal nature, (e.g., agents running on the same hardware compete for computational resources) or of external nature (e.g., agents compete for items that are part of their perceived and represented environment).

For achieving resource-adapting agent societies, our resource concept has to consider macro-level aspects of an agent society, i.e., interdependencies among agents. We model resources as parts of an abstracted environment; we call an environmental entity an *abstract resource* if the following criteria hold:

- **Usefulness:** The expected success/performance of the agent's action must depend on the use of a resource.
- **Inter-agent or intra-agent dependency:** The entity must either cause a dependency of an agent on another agent or agent group, or a dependency of a sub-module of an agent to another sub-module.
- **Local perspective:** The entity can be regarded as a resource by one agent while this may not hold for another agent.
- **Limited availability:** The entity must be available to the agent, but not in unbounded capacity.
- **Restricted use:** For our purposes we require that the entity can only be used by one agent at a time.

A resource can enhance, or by its absence, constrain the agent's (or agent group's) choice of action (information, perception, capabilities are good examples) or it can constrain the effects of the agent's (or agent group's) action execution in a quantitative or qualitative fashion, for instance, computational time assigned to an agent in a pseudo-concurrent environment, such as JAVA^{TM} Virtual Machine.

The construct of a *semaphore* is a classical control mechanism to handle resources: only one of the agents is able to allocate a resource and is therefore allowed to reason (internal use) or act (external use). Assigning an abstract resource thus amounts to putting indirect "guidelines" or constraints on the behavior of the affected agents.

Categories of Abstract Resources

In order to illustrate the concept, we present some examples of abstract resources which can be classified into *societal* and *individual* resources. In this work, we focus on the former category. In general, societal abstract resources express certain *rights* that some agents have while others have not. Individual resources denote *capabilities* of an agent.

Examples of Societal Abstract Resources

The following examples are adjustable *macro-level* traits of an agent society. Due to our definition, they are abstract resources *for the whole agent society* since they induce interdependencies between member agents in the society and the society itself.[3] The examples are always perceived from the *perspective* of the agent society. Similarly, the criteria *usefulness*, *limited availability* and *restricted use* hold straight-forwardly, so we will explain only the dependency property for each resource.

In accordance with [Jam97], we also specify the meaning and the constraints of the allocation of each resource. Furthermore, we point out whether the scale of such an allocation is discrete or continuous, nominal, ordinal, or cardinal. For the use of a resource allocation mechanism (Section 3.4), we need at least an ordinal scale; hence for nominal cases we introduce an artificial order.

Right to decide on the number of member agents in a society: Obviously, the more agents are added to the system, to more tasks can be performed. However, introducing too many agents may lead to communication overheads which may actually decrease the overall performance. For an agent society, the control of this parameter can be taken as an abstract resource since member agents are integrated into the society inducing a dependency of the member agents to the group as a whole (implemented by the representative agent). The allocation of that resource ranges from 1 to infinity on a discrete and cardinal scale.

[3]In Section 3.5 we introduce a special agent to represent an agent society; this agent then operates on the macro-level resources.

Right to define organizational forms: Similar to organizational forms built up in business or nature, agent societies need some organizational structure. This structure can be used to determine communication and cooperation partners, it may define a command hierarchy, etc. Several different models are available by applying organizational forms presented in Section 2.2 to the MAS domain. In the big picture, the organizational structure of a multi-agent application determines a degree of centralization. The more centrally the society is organized, the more the members depend on the higher institution; hence the authority to decide on a organizational form can be seen as an abstract resource. Each possible organizational form may be represented as one point on a discrete and nominal scale.

Right to determine agent specialization: Heterogeneous societies (i.e., societies with agents of different architectures or with different capacities) may have wider ranges of expertise at their disposal which may enable them to perform better; on the other hand, such a society may tend to be susceptible to failure since a specialist may not be replaced easily. In a heterogeneous society, members may depend on the special capabilities of other agents; therefore these capabilities are abstract resources. The allocation of that resource ranges from 0 (a totally heterogeneous group) to 1 (a group where all members are equal) on a (in principle) continuous and cardinal scale.

Right to assign migration: In a traditional approach, communication among geographically distributed agents is performed long-distance over the network. This procedure can be rather time consuming in case the network is heavily loaded and the communication process consists of some complex negotiation procedures. In a *migration* approach, an agent is transmitted over the net in order to communicate with its partner on the local server. This technique can be fast if the receiving server has an accurate model of the traveling agent. In this case, only data describing the agent's internal states have to be transmitted; a copy of the agent can be generated locally.

The decision power to force members of a group to migrate can be implemented as follows: A threshold is specified that determines a certain spatial distance between two agents. If the current spatial distance between communication partners is greater than the threshold, one of the agents is forced to migrate. Since the actions of this agent depend on the assigned threshold value, the right to specify the threshold value is an abstract resource. This realization leads to a continuous and cardinal scale ranging from 0 to possibly infinity.

Right to select communication channels: Depending on the organizational form, communication and cooperation within a certain group may be restricted strongly or weakly: For instance, in a hierarchical structure members of a group may only communicate with the head of the group; in other organizational forms communication among group members may also be allowed. Furthermore, communication and cooperation *between groups* has to be specified: It has to be determined whether all group members may communicate with neighbor group members or only selected agents (e.g., the heads of the groups) may do so. Since communication restrictions clearly lead to dependencies of the agents in question, the right to assign such restrictions amounts to an abstract resource.

Communication may be interpreted on a discrete, nominal scale, where each possible communication type denotes one point in that scale. However, it may be possible to introduce an order which leads to a ordinal scale. This order may for instance reflect the degree of decision making autonomy of lower-ranked agents.

Choice of the communication protocol: Communication among artificial agents is usually performed in a structured form, e.g., by using speech act protocols. However, several protocol types are possible, ranging from fast master-slave communication over various types of auctions to complex negotiation protocols which may be quite time consuming if applied widely. Similar to the previous point, this issue induces dependencies and is hence an abstract resource. The allocation of this resource can be modeled on a discrete, ordinal scale where each communication type is represented by one point.

Examples of Individual Abstract Resources

Whereas the previous quantities express macro-level resources that lead to dependencies among agents or agent groups, the following resources concern the individual agent on the *micro-level.* These issues are of minor relevance for this work since we do not focus on a certain agent architecture; they are just listed to give a better feeling for the term "abstract resource".

Capability to model the complexity of knowledge representation and inference capabilities of an agent: Efficient integration of these capabilities is a crucial aspect of an agent design. A powerful inference mechanism enables an agent to draw complex conclusions. However, complexity theory has shown that powerful reasoning algorithms easily become intractable. The capability to find an efficient trade-off between tractability and expressiveness can be modeled as an abstract resource on a continuous and cardinal scale ranging from 0 (a very simple mechanism is used) to 1 (a most complex procedure is applied).

Capability to model the complexity of an agent's perception functionality: If an agent has a perception module at its disposal, its performance can be influenced by introducing sophisticated methods which enable the agent not only to perceive passively its environment, but also to focus actively on certain circumstances. If such a functionality is given, the question has to be addressed on how extensively it should be used. Similar to the case above, the capability to adjust this functionality can be modeled as an abstract resource on a continuous and cardinal scale ranging from 0 (a very simple mechanism is used) to 1 (a very complex procedure is applied).

Capability to model the complexity of communication among agent modules: As the various modules of an agent have to cooperate to achieve an agent's goals, communication among internal modules can be scaled leading to an abstract resource on a cardinal and continuous range between 0 for a low complexity and 1 for high complexity.

This enumeration is not complete; on the contrary, the examples have rather been chosen to illustrate the abstract resource concept. In Chapters 6 and 7, we present concrete instances of the resource adaptation in industrial applications. As mentioned above, abstract resources are used to calibrate a society of artificial agents, which is discussed in the next section.

3.3 The Bounded-Optimal Agent Society

After having defined our resource concept, we now return to our original goal, the specification of a self-adaptation scheme for agent societies. We lean on ideas and concepts derived in the fields of bounded optimality and anytime algorithms. However, we cannot directly adopt these concepts, since they have been designed for the single-agent case. In the following we shortly summarize the most relevant work in these areas, before we extend the concept to the multi-agent case in Section 3.3.2.

3.3.1 Bounded Optimality and Anytime Algorithms in the Single-Agent Case

The research areas of bounded rationality and anytime algorithms are rather new topics, derived in the late 1980's and the 90's. The key idea of *bounded rationality* is to regard problem solving itself as a (meta-level) action to be performed by an agent. Hence, it has to be incorporated into the deliberation process to determine the next action of an agent. *Anytime algorithms* are mechanisms whose application can be interrupted at arbitrary time (in the ideal case) and feasible, but not necessarily optimal solution can be returned.

Bounded Rationality

For optimization purposes it is desirable that in a given situation, an agent behaves "optimally", i.e, it performs the best action. Good [Goo71] calls the behavior rationality of *Type I*. Rationality of *Type II* additionally considers the deliberation cost: after the deliberation and performance of an action of an agent of that type, its utility is maximized in comparison to all possible deliberate/act combinations.

Russell and Subramanian [RS95] define a whole range of different measures for optimal behavior:

- **Perfect rationality:** In this case, the agent always "does the right thing", i.e., it performs the action that maximizes the expected quality of the result of the action. This expectation bases on the current knowledge of the agent.
- **Calculative rationality:** In contrast to the previous notion, this form of rationality regards the continuously passing *time*: a calculative-rational agent incorporates the temporal situation in the beginning of the deliberation into the state definition, but the determination of the next step is not timely bounded. Therefore, an agent of this type returns after a while a solution which would have been optimal in the beginning of the reflection, but not at the time of the output of the answer.
- **Meta-level rationality:** In order to overcome the above problem, a meta-level-rational agent performs a two-leveled optimization: on the lower *object level* the given problem is solved, i.e., the next action is determined, while on the higher *meta-level* the computation sequence to derive that action is optimized. Since the meta-level reasoner also needs computational resources, approximations to meta-level rationality have been applied and have shown successful in practice. Russell and Wefald [RW89] distinguish between *uniform* and *non-uniform* meta-level architectures of agents. In the former case, the meta-level reasoner employs the same mechanisms as the object-level reasoner, which is not true in the latter case.
- **Bounded Optimality:** Whereas in the previous approach, the meta-level component optimizes the sequence of object-level computation steps, a bounded-optimal agent optimizes the entire agent program; thus a bounded-optimal agent behaves as well as possible given its computational resources.

Zilberstein [Zil93] introduces the more technical term of *operational rationality*, by distinguishing between problem solver and control instance: The control instance of an agent optimizes the distribution of computational resources to submodules of the problem solver. The resource distribution is not subject of the actual problem to be solved.

Anytime Algorithms

Horvitz [Hor87b] introduces a first approach called *flexible computation* in which he proposed to apply different problem solving strategies depending on the resources at hand. Considering the ratio between the solution utility and the costs for computational time, a sequence of feasible results is produced where the quality of these results increases monotonically. Horvitz calls this a *refinement* of the solution. In particular, he poses the following requirements for a refinement scheme:

- **Continuity of utility and quality:** The utility and quality of the results that are derived during the refinement, are continuous functions.
- **Monotony of quality:** The quality increases monotonically according to the use of resources.
- **Convergence of quality:** The quality converges against the optimal result which would have been computed if unlimited resources had been available.
- **Dominance of utility:** During the run of such an algorithm, there have to be time intervals in which the utility increases monotonically according to an increasing use of resources.

Based on this work, Dean and Boddy [DB88] introduce the concept *anytime algorithm* with the following properties:

- Anytime algorithms may be interrupted and continued at any time with very little administrative overhead.
- At any time, such an algorithm provides a solution whose quality increases monotonically according to the invested computational time.
- *Performance profiles* log information about the trade-off between the invested time and the achieved quality of the solution. This information may be used for an optimized distribution of computational time to several different anytime algorithms.

Zilberstein [Zil93] uses *off-line compilation* to realize a composition of several anytime algorithms to one system. He introduces *conditional performance profiles* which are used to describe the quality of a result of an anytime algorithm under consideration of a certain input quality. This is needed as Zilberstein employs multiple anytime algorithms in a sequential fashion where each anytime component uses the results of its predecessor. During run time of the system, the resource allocation can be modified; however, the total computation time must be determined prior to the start of the system making a flexible reaction to changes of the resource limitations impossible.

Zilberstein and Russell [ZR96] propose the use of the following metrics for quality measures to validate performance profiles:

- **Certainty:** This metric expresses the probability that a result is correct.
- **Accuracy:** This metric reflects the accuracy of the return value, i.e., it indicates the quality difference between the found solution and the optimal one.
- **Specificy:** This metric denotes the level of detail in the result for cases where results are always correct but differ in granularity.

Zilberstein distinguishes between *contract anytime algorithms* and *interruptible anytime algorithms*. The latter return at once a feasible solution at any break of the system, while algorithms of the former type need after an interruption some more time to return a feasible result. The approach proposed by Dean and Boddy can then be classified under the more general latter case. Although an algorithm of the former type is less desirable, in practice it is much easier to be built.

3.3.2 Bounded Optimality in the Multi-Agent Case

We now extend the concept of bounded optimality to a society of agents. A simple adaptation of the presented mechanisms is hardly possible, not only since these mechanisms are designed for the single agent case, but also because they focus mainly on *time* as the only resource to be distributed which is too narrow-minded for our purposes. Zilberstein's concept of compiling several anytime algorithms to one main system cannot be applied since he orders them in a chain which is too static for our purposes. Although we cannot apply the derived mechanisms, we use the proposed methodologies and concepts which we will extend for our needs.

Russell and Subramanian define *bounded optimality* as a property that specifies the optimal *agent program* rather than the optimal *action* of an agent. The

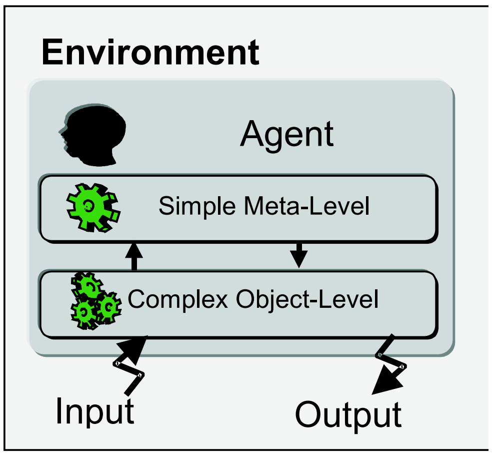


Figure 3.1: Architecture for a bounded-optimal agent

bounded-optimal agent solves the combined optimization problem of simultaneously finding the optimal agent architecture and the optimal agent action. Transferring this specification to the multi-agent case, we characterize a bounded-optimal agent society as a collection of agents that solves best the complex optimization problem of simultaneously finding the optimal architecture and the optimal action *for each agent in correlation with all other agents*, for a given amount of (abstract) resources.

The most important assumption we make is *benevolence of the member agents*: All agents of a *society* must have a common goal and they must accept to follow guidelines to reach that goal. Not all agents in a MAS must obey the two requirements, agents can also exist outside such a society. These are the only restrictions we have for our resource allocation scheme. In Chapter 5 however, we pose stricter requirements onto a society of agents. Such a special case is then called a *holonic agent system.*

Architectural Aspects

Russell and Subramanian propose a two-fold architecture to approximate the optimal behavior of a single agent (Figure 3.1). They argue that a rational behavior cannot be reached in a domain-independent manner without solving the higher-order problem of additionally approximating the optimal agent program. This justifies the use of meta-reasoning in the single-agent case. Since the meta-reasoning component also consumes resources, it should be of neglectable complexity. Fast and simple resource assignment is therefore only locally optimal, i.e., for a single time step. Thus it does not necessarily lead to the global optimum, but hopefully to a satisfying solution.

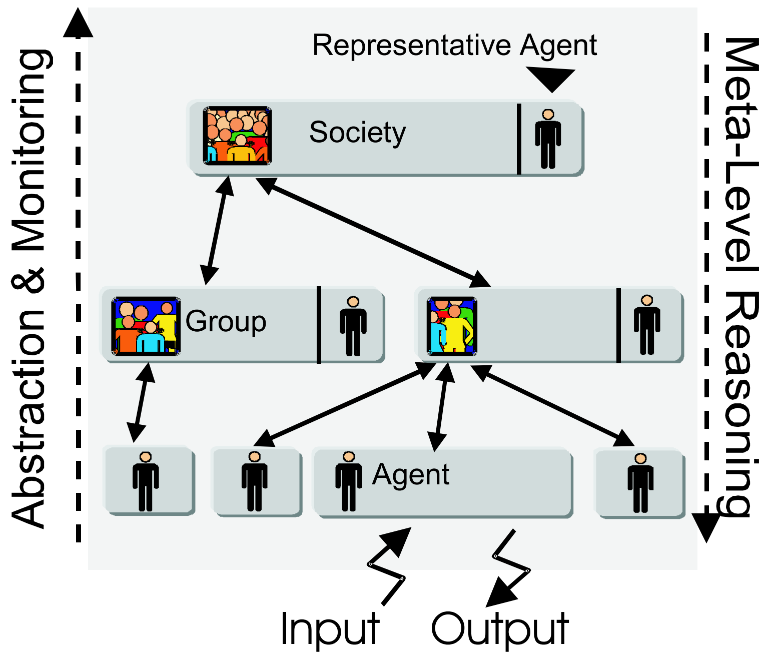


Figure 3.2: Architecture for a bounded-optimal agent society

A first idea to build a system that approximates the bounded-optimal agent society is to adopt this technique to the multi-agent case by equipping every agent with such a two-fold architecture. We argue that for the case of an agent society such an architecture for every member agent is insufficient. An architecture to approximate the bounded-optimal society does not only have to elaborate micro-level aspects of the individual agent to ensure optimality. It also has to incorporate macro-level aspects of the society:

- Even, if every single agent behaves bounded-optimally, it cannot be guaranteed that the whole agent society behaves globally bounded-optimally, since every agent has only a local perspective and knowledge.

- Interactions among agents have to be taken into consideration for an optimization procedure. Hence, for our purposes, reasoning over resources in the traditional sense is not enough; we require to also reason about interdependencies between agents, i.e., abstract resources.

In order to express that some states are more desirable for a situated agent society than others, we assign a value to each state of the environment. As this utility value has to be computed by the society, it depends on its local perspective and therefore, can be subjective, and can possibly include errors.

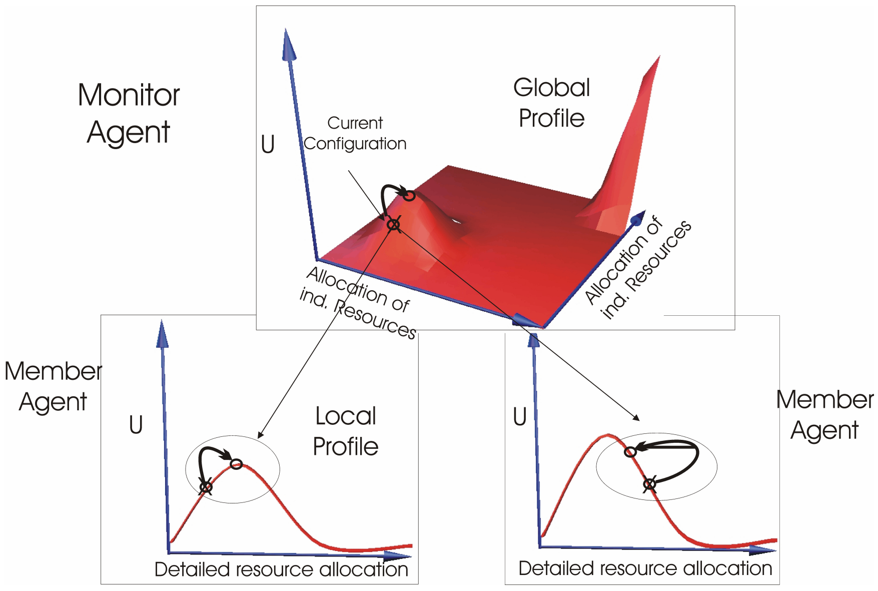

Monitor Agent
U
Current Configuration
Global Profile
Allocation of ind. Resources
Allocation of ind. Resources
Member Agent
U
Local Profile
Detailed resource allocation
U
Member Agent
Detailed resource allocation

Figure 3.3: From local to global profiles

In order to monitor the utility of a state of the whole society, we introduce a centralized unit, representing the whole society. This top-most *decision stage* has to measure the utility or performance of the whole society at a current or future state. In large societies, this measuring is an infeasible task for only one control unit. Hence, we introduce a flexible hierarchy of decision stages, such as the individual agents, groups, and the society (Figure 3.2). At each of these stages resource assignment to members of the subordinate stages is performed, based on their performances/utilities in the current situation.

Implementation of the Architecture

There is a wide range of possibilities to implement this resource assignment procedure at a given stage: In a *de-centralized realization*, resource allocation is performed solely through negotiation among stage members. In a *centralized setting*, the autonomy of the member agents is heavily restricted or even further, the former individual agents are replaced by a new agent. We choose a *hybrid setting*, where for each stage a new, *representative agent* (or *monitor agent*) is introduced. The only task of that new agent is to perform resource allocation on a *meta-level*; the original *object-level* tasks still have to be performed by the members of the group. We prefer this hybrid approach over the other possibilities because it avoids the communication overhead of a fully distributed setting, and is also much more suitable for a dynamic reconfiguration of the agent society than a centralized setting.

On every stage, the representative agent achieves a current profile of its social stage through frequent *performance monitoring.* By setting *guidelines* for the behavior of the lower stages an adaptation to changes in the environment is achieved. Guidelines are spread through the allocation of abstract resources.

In order to assign resources efficiently, the dependency between resource allocation and performance has to be determined. In most cases, this dependency cannot be computed, but has to be observed: subordinate representative agents therefore monitor *local profiles* and report them to the next higher representative agent which in turn uses this information to build a more *global profile* (see Figure 3.3). An example is the knowledge whether a semaphore which has been previously allocated to a member agent has already been released and is now free for further use by one of member agents.

In the following, we propose a *Generic Resource Allocation & Implementation aLgorithm (GRAIL)* which consists of two nested parts: *a hierarchical* ***R****esource* ***I****mplementation a****L****gorithm (RIL)* for resources defining resource-based interaction among agents and, embedded, a ***R****esource* ***A****llocation a****L****gorithm (RAL)*, defining a resource configuration in the space of possible configurations. In the next two sections, we will present the two parts in detail.

3.4 Resource Allocation Algorithms (RALs)

The task of a resource allocation algorithm is to discover an *optimal abstract resource distribution* which can be viewed as the optimum of the performance function in an m-dimensional search space where m denotes the number of scalable resources and each point in that search space corresponds to one particular system configuration: Each resource is represented by a dimension in the search space, whose domain ranges from one extreme point of that quantity to the other.

Following the methodology of operations research (OR), the problem can be characterized as follows:

max	Overall Performance
subject to	Distribution of abstract resources
	Minimal and maximal boundaries for the use of each resource
	Maximal boundary for the overall use of resources

Figure 3.4 shows a simple, two-dimensional example: here performance depends only on the number of agents and the usage of a sophisticated knowledge representation (KR) component. For the sake of simplicity, this quantity is scaled equally for all agents, i.e., a modification of this parameter affects all agents in the same way. Whereas the agent number is represented by a discrete dimension

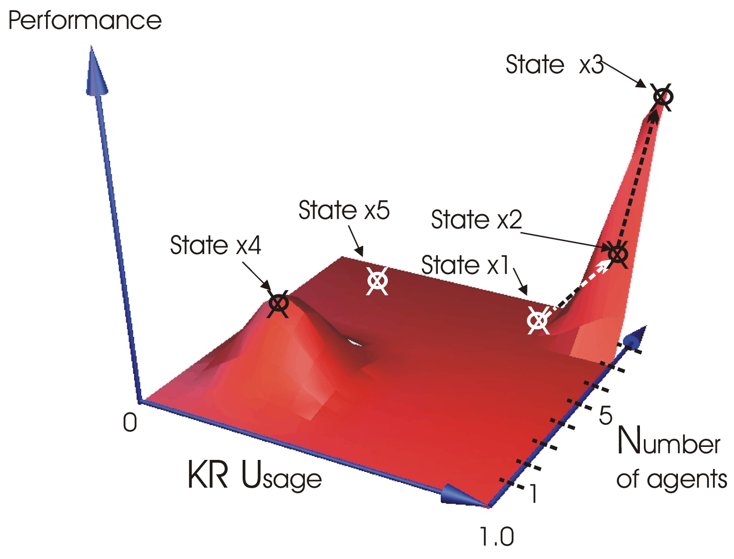


Figure 3.4: A simple example of a system performance relation

(whose domain ranges from 0 to possibly infinity)[4] the other dimension is modeled as a continuous one, its domain ranging from 0 to 1; a number between those extremes indicates how much percentage of the agents' computational time can be used in their KR components.

If no information about the problem characteristics is available, finding a global optimum is a hard problem whose solution can barely (if at all) be found in reasonable time. Furthermore, in many cases, a *high enough local optimum* will do, considering that the list of tasks the system has to perform may change rapidly: current configurations may lose high performance and hence, have to be discarded quite frequently.

3.4.1 A Greedy Resource Allocation Algorithm

The steepest descent method [Zou76] for finding local optima is in principle well suited for a RAL strategy since it is able to react fast to situation changes in order to maintain high, but not necessarily optimal performance during the complete run of the application. For our purposes, we slightly modify the approach; differences are discussed in Section 3.4.4.

An efficient configuration (i.e., a (locally) optimal point in the search space) can be found by moving from some arbitrary starting point in the search space iteratively to the optimal point.

[4]The figure shows also performance values for value fractions on that scale which are actually not defined. This is done only for a more intuitive display.

Algorithm 1 `Greedy-RAL (Agent` a`, SearchSpace` ss`)`

```
 1: previousPerformance := currentPerformance
 2: currentPerformance := getCurrentPerformance()
 3: if currentPerformance > previousPerformance
 4:    currentStepDirection := previousStepDirection          /* step direction remains */
 5: else
 6:    undoStep()
 7:    currentStepDirection := findNewStepDirection()     /*that has not been tested yet */
 8: endif
 9: if currentStepDirection = void                           /*no untested direction left*/
10:    boolean solutionFound := true
11: else
12:    performStep(ss, currentStepDirection)                  /*modify resource allocation */
13:    boolean solutionFound := false
14: endif
15: SubSearchSpaceVector sssv := computeSubSearchSpaceVector(ss)
16: return (sssv, solutionFound)
```

In every step, first the direction is determined on which to move. In case the relation of performance towards resources can be expressed by some differentiable function, the best direction can be determined using partial differentials. Otherwise the direction has to be determined empirically. One way to do so is to traverse the search space in a *depth-first search* strategy: A *marginal step*[5] in a certain direction of the search space is performed. If the performance has increased, a further step in this direction is made; otherwise the step is undone and a step into a new direction made. This procedure is repeated until a point is reached where all steps towards promising directions have resulted in performance decrease, and hence an optimum has been reached.

In the previous example of Figure 3.4, a starting point may be x_1. A maximal performance gain is achieved by adding more resources to the KR components: x_2 may be achieved. By increasing the number of agents to eight, the optimal configuration (represented by x_3) is found.

Algorithm 1 describes this technique in a rather abstract and general way in order to provide a widely applicable technique. A RAL is embedded into the hierarchical RIL scheme (see Algorithm 6 in Section 3.5); on the higher societal stages of the agent hierarchy, the RIL calls a RAL to perform only one step in its search space. A RAL has to return a list of search spaces for the sub-ordinate resource-adapting agents and a flag which denotes whether a local optimum has

[5]If the search space is not *convex* (i.e., not all possible intermediate points between two arbitrary points in the search space do also belong to the search space), a "marginal step" denotes the smallest step possible to reach again a point in the search space. For instance, in the above example the discrete dimension "Number of Agents" induces non-convexity. A marginal step along that dimension can be achieved by increasing or decreasing the number of agents by one.

been found. In the *Greedy-RAL* algorithm this flag is set to *true*, if all possible step directions have been tested (the procedure *findNewStepDirection* returns void in Line 9 of Algorithm 1) and no improvement could have been found. The list of sub-search spaces is generated in dependence from the current position in the search space *ss* (Line 15).

The procedure *getCurrentPerformance()* always returns the current performance of the agent group the monitor agent is heading. The scenario designer has to actually implement this domain-dependent procedure. In Line 7, the computation of a new direction in the search space always returns a choice that has not been tested before for a certain position, if there exists one. At the moment we always assume a marginal step size. Later, we will refine the scheme by introducing a variable larger step size.

We assume that all data relevant for the RAL (in particular, the previous position in the search space) is stored locally for each agent that performs a RAL. Initially, the value of *currentPerformance* is set to 0, and *previousStepDirection* is set to an arbitrary direction, leading to a random move as an initial step.

In Sections 3.4.2 and 3.4.3 we will further elaborate the direction finding strategy of this procedure. In Chapter 4 we will show some modifications of the implemented systems from the theoretic framework that make the system more applicable. In Chapter 6 we will instantiate this RAL concretely to the MoTiV-PTA application where we refine the scheme in greater detail.

Properties of Greedy-RAL

Some assumptions have to be made in order to effectively use that simple algorithm: If all of the following assumptions hold, we can can show that the proposed Greedy-RAL in combination with a RIL scheme will find an optimal solution. However, we argue that some small violation of these requirements does not reduce the quality of the achieved solution too much. For our purposes we will relax them.

1. *The performance function P is concave*, i.e., for arbitrary points x and y in the search space and for $0 \leq \lambda \leq 1$ holds: If an intermediate point $z = \lambda x + (1-\lambda)y$ is an element of the search space, then $P(z) \geq \lambda P(x) + (1-\lambda)P(y)$. If this assumption is violated, the algorithm may run into a local optimum, or even worse, a plateau: for instance in the previous example, moving from x_5 to any other direction does not lead to a performance gain; by beginning from some other starting point, the algorithm might return x_4 as a result. However, we are not interested in finding a global optimum in the search space; a relatively high local optimum will do as well, in particular under the consideration of a continuously changing environment.

2. *During the search for optimization neither the search space changes, nor does the definition of the performance function.* This assumption cannot hold in general, as e.g., new tasks may be incorporated anytime which may change for instance the optimal number of agents. However, due to the hierarchical RIL scheme, the system can adapt to minor changes in the search space or the utility function by redefining subparts of the search space and then running a RAL locally on that sub-space.

3. *Performance impacts of the resource quantities are independent from each other.* Again, this is an assumption which does not hold in the general case. However, due to the choice of abstract resources and the resulting design of the agent hierarchy, we *can* assume such an independence for our case (see Section 3.5).

4. *All dimensions are scaled cardinally or at least ordinally*, i.e., a partial order can be introduced. This assumption is crucial for the algorithm above to decide in which direction to move next. However, it also cannot hold in general; In such cases an order has to be introduced artificially (e.g., by using some heuristics).

5. *All dimensions are scaled discretely.* Of course, this requirement does not hold in general, but we we discretize any continuous dimension on a suitable level of granularity.

A setting for which all five requirements hold is called *admissable.*

Corollary 3.4.1 *For an admissable setting, Greedy-RAL embedded in a RIL that keeps calling a RAL step until a solution is found, will eventually find an optimal position in the local search space.*

A proof of this corollary can be found in Appendix A. For our purpose, we cannot guarantee all assumptions to hold; however, as already discussed above, some small violation to the one or other requirement will only lead to minor disturbances.

Extensions of Greedy-RAL

Two issues of Algorithm 1 have not been clarified yet: how to find an efficient direction to move (Step 7) and how to determine an efficient step size (in Step 12), since moving only a marginal step per iteration is too inefficient. For the specification of the step size we employ a *bisection technique*: The system designer has to specify a *regular step size* which is usually taken for steps in the search

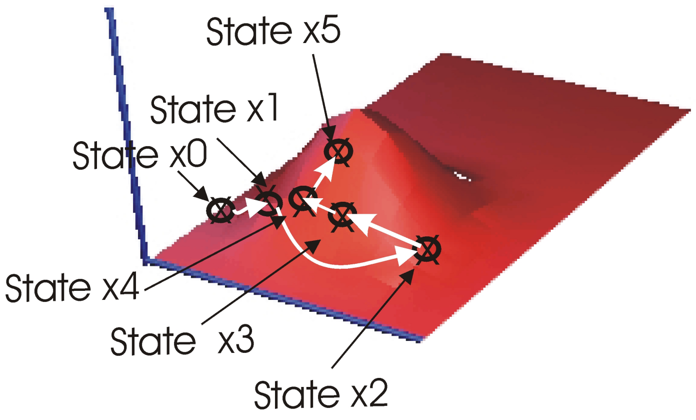


Figure 3.5: Bisection in Greedy-RAL

space, except for two cases: a boundary of the search space is reached, or a performance decrease is measured, which means that the last step (or even the step before that) has been too large and the maximum in the search space has been exceeded.

Consider the example of Figure 3.5 after having moved from x_1 to x_2. There must be a point in the interval between the current point (here x_2) and the previous one (x_1), or between the point x_1 and its predecessor x_0 which has a higher performance value than both, x_0 and x_2. Stepping to the mid-point $x_3 = 0.5 * x_0 + 0.5 * x_2$ splits the search interval, which then ranges in the example from x_0 to x_3. The search for the optimal point is repeated on the new interval, until the size of the interval has become significantly small (x_4 is found). Then a new step direction is determined and the best point is searched among this direction (x_5).

Algorithm 2 shows the technique in more detail. This technique basically cuts down a multi-dimensional search space into a sequence of one-dimensional searches. Of course we have to take into consideration that after having optimized among some dimension, the position according to a previously optimized dimension may no longer be optimal and has to be re-optimized again. For this algorithm we ussume that the variables *oldPosition* and *previousPosition* have initially been set to the start value of *currentPosition*.

In Lines 5 and 10, the step size is computed by taking the minimum of the regular step size and the distance to the boundary of the search space. The procedure *bisection* is applied whenever an optimum has been exceeded (Line 8). During the recursive run of that procedure, a step is taken to the middle point of the interval (Line 25), until the interval has become small enough. Afterwards,

Algorithm 2 `Bisection-Greedy-RAL (Agent` a`, SearchSpace` ss`)`

```
1: $previousPerformance := currentPerformance$
2: $currentPerformance :=$ getCurrentPerformance()
3: if $currentPerformance > previousPerformance$
4:    $currentStepDirection := previousStepDirection$
5:    $stepSize :=$ computeStepSize()                    /* are boundaries reached? */
6: else
7:    undoStep()
8:    bisection ($oldPosition, currentPosition$)          /* moves to optimal intermed. point */
9:    $currentStepDirection :=$ findNewStepDirection()
10:   $stepSize :=$ computeStepSize()
11: endif
12: if $currentStepDirection =$ void
13:    boolean $solutionFound :=$ true
14: else
15:    performStep($ss$, $currentStepDirection$)
16:    boolean $solutionFound :=$ false
17:    $oldPosition := previousPosition$
18:    $previousPosition := currentPosition$
19:    $currentPosition :=$ computeNewPosition()
20: endif
21: SubSearchSpaceVector $sssv :=$ computeSubSearchSpaceVector($ss$)
22: return ($sssv$, $solutionFound$)
23:
24: Procedure bisection ($firstPoint, secondPoint$)
25:    stepTo($firstPoint + secondPoint$) / 2
26:    if performance($firstPoint$) > performance($secondPoint$)
27:       $secondPoint :=$ getAverage($firstPoint, secondPoint$)
28:    else
29:       $firstPoint :=$ getAverage($firstPoint + secondPoint$)
30:    endif
31:    if $|(firstPoint - secondPoint)| > threshhold2$
32:       bisection($firstPoint, secondPoint$)
33:    endif
34: endprocedure
```

a new step direction and size are determined (Lines 9 and 10), and the step is executed (Line 15). For the sake of orientation, In this and the following algorithms only new modifications are labels with comments.

In the model above a local step has to be applied in the search space described previously. Obviously, pure uninformed search will lead in Step 9 to a random choice of direction in case a new step is *not* taken in the same direction as the previous one. However, in such cases, *heuristics* can be incorporated to determine the next step (see Lines 10-12 of Algorithm 3) converting Greedy-RAL from a blind depth-first search strategy to a *best-first search strategy.* If there is no other indication to decide between several choices, relying on a heuristic keeps Greedy-RAL from making a random choice.

Algorithm 3 `Heuristic-based-Greedy-RAL (Agent` a`, SearchSpace` ss`)`

```
previousPerformance := currentPerformance
currentPerformance := getCurrentPerformance()
previousStepSize := currentStepSize
if currentPerformance > previousPerformance
  currentStepDirection := previousStepDirection
  stepSize := computeStepSize()
else
  undoStep()
  bisection (oldPosition, currentPosition)
    (currentStepDirection,currentStepSize) := applyHeuristics(currentPosition,
                                              (previousStepDirection, previousStepSize),
                                              currentPerformance - previousPerformance)
    if currentStepDirection = void                /* if no heuristic could be applied: */
      currentStepDirection := findRandomStepDirection()          /* random move */
    endif
endif
if currentStepDirection = void
  boolean solutionFound := true
else
  performStep(ss, currentStepDirection)
  boolean solutionFound := false
  oldPosition := previousPosition
  previousPosition := currentPosition
  currentPosition := computeNewPosition()
endif
SubSearchSpaceVector sssv := computeSubSearchSpaceVector(ss)
return (sssv, solutionFound)
```

It is the nature of a heuristic to give a rough guideline or indication to find faster a better solution. Depending on the application, such a guideline needs not to be exact or correct for all cases. Additionally, an indication may be interpreted in different ways leading to possibly different suggestions what step to perform next.

Among others, heuristics can be gained from observations made in related sciences presented in Section 2.2, such as social psychology or management theory. In the next section, we focus on heuristics that can be gained from data collected from the members of the agent society, while Section 3.4.3 presents an the integration of a machine learning approach to draw conclusions from the usefulness of previous modifications. Note that these heuristics are kept generic on purpose, since our approach is designed to be integrated into a framework of heterogeneous agents. They will be instantiated in detail in the case study of Section 6.

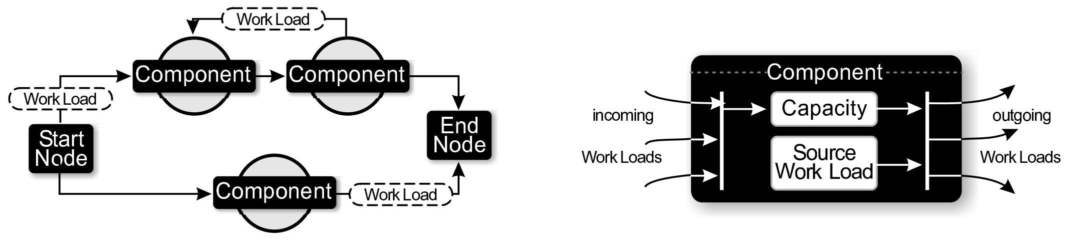


Figure 3.6: Work flow model of a technical system

3.4.2 Bottleneck Analysis for Heuristics Retrieval

Bottleneck analysis is becoming an increasingly important means for optimization of very complex technical systems (see for instance [KE96]). Bottlenecks, i.e., unbalanced structures occur if the system was configured under false behavior estimations. Both, over- and underloaded entities in the system reduce overall performance since overloaded units cause delays, while underloaded units could have been used elsewhere.

Bottleneck analysis provides an assessment method for units in technical systems: a unit is characterized on a rather high level of abstraction by only regarding its input/output behavior. The system is represented as a directed graph (left-hand side of Figure 3.6); its nodes coincide with system components, arcs represent the work flow. In general, two additional nodes are introduced: a start node and an end node, representing the beginning and the end of the work flow.

Arcs are labeled with *work load* values. Such a value represents the pass-through of load per time unit on this connection. Each component is labeled with a *capacity* value, a *source work load* and a *transition function* (right-hand side of Figure 3.6). The capacity is a measure of pass-through on this working unit, limiting the processing of incoming work loads. It has to be regarded as an upper threshold of work load to be processed without leading to delays. A unit's source work load constitutes work load that originates from that particular unit, i.e., that did not emerge from other factors. The effect of work load (incoming as well as new source load) on a component results in a reduction of capacity and of new work load leaving the unit, determined by the transition function.

The transition function has to compute the contribution of the various incoming loads to capacity consumption, the contribution of the new produced work load to capacity consumption, the total capacity consumption and the throttling of incoming work load (i.e., the overload), should the situation arise, and the outgoing work load depending on the above quantities.

In a traditional approach, a bottleneck is regarded as a location of overload within a system structure. *Real bottlenecks* are usually distinguished from *hidden* *ones*: Real bottlenecks occur if the capacity of a unit is smaller than the work load it has to cope with. A hidden bottleneck is an imbalance that is currently not effecting the system's overall performance, but will do so if other, real bottlenecks are removed. Goal of bottleneck analysis is the detection and removal of bottlenecks: a mathematical model of the system is put up by characterizing the work flow in a fashion described above. Network flow algorithms developed in Operations Research can then be applied to that model in order to detect bottlenecks (see [AMO93] for an overview).

Such an approach can be adapted to detect sub-optimalities within a society of artificial agents. However, it has some severe drawbacks:

- Only overloaded units can be found. Underloaded components cannot be detected; their unused work power is lost.
- Work load and capacities have to be estimated for the mathematical model of the system. If these estimations are incorrect, no mechanism can find real bottlenecks.
- In traditional bottleneck analysis, work load and capacity are not considered variable over time. In such cases of variations, average values are taken or the procedure has to be rerun for each variation.
- Similarly, the analysis has to be repeated after each system modification due to detection of real bottlenecks in order to find hidden ones.

An Alternative Approach

We propose a different approach to bottleneck analysis in societies of artificial agents: in a mathematical model of an agent society, each agent would be represented by a node and each communication channel by an arc. As an alternative, we do not estimate work load and capacities for building up a mathematical model, but make direct use of the architecture of the multi-agent society which provides us with information about agents' work load. The following conclusions derived from action and communication behavior can be drawn in order to derive heuristics in a local search. This information can be derived for instance, by inspecting goal queues and communication queues of the agents (if existing), or by the number of actions and communication acts per time unit.

- The size of a member agent's goal queue and the number of tasks per time unit performed by the agent indicate their work load: A very large goal queue may be a sign that more agents are needed that are specialized on achieving that type of goals. On the other hand, if only very few tasks have been performed by an agent, this agent might be better removed from the system and the tasks of this agent may be done more efficiently by another agent of the same type, considering the fact that an idle agent still consumes time and space resources. This may therefore be an indication to delete that particular agent.

- The size of agent communication queues (if they exist) and the number of performed communication acts per time unit can be used to validate the usefulness of the chosen type of communication protocol: a too large queue reveals that the currently applied communication protocols are too complex. An almost empty communication queue or a extremely small number of communication acts per time unit is a sign for an underload: a more complex communication protocol that might lead to better results could be used probably without reducing overall performance.

- Furthermore, an analysis of agent communication behavior in terms of the distance of communication partners in a network and of communication complexity can be used to decide when to employ agent migration: If complex protocols are often pursued over a long distance, it may become more efficient to send the agent over the net and let it perform communication locally.

- Depending on the communication language used among agents, the type of performed communication acts gives also hints about the optimal structure: well-defined social structures of a group may speed up cooperation as cooperation roles are already pre-defined. For instance, if an agent emits many directive communication acts such as *request* and often refuses requests from other agents, a more hierarchical structure may be introduced, giving this agent a relatively high position.

All these observations can be used as heuristics in a higher staged RAL, i.e., the RAL performed by a monitor agent to approximate optimal structure and communication patterns in an agent group. In Chapter 6, we will instantiate the rather general technique to the environment of the MoTiV-PTA domain.

3.4.3 Machine Learning for Heuristics Retrieval

The presented Greedy-RAL can be extended by integrating a machine learning technique. Assuming a dynamically changing environment, we are faced with an arbitrarily high number of possible situations. To cope with such a scenario, we apply an approach derived from memory-based reasoning [SW86], which is based on on the observation that only few of all possible situations really occur. In

Algorithm 4 `applyHeuristics (Situation sit, Action act, double perfChg)`

```
1: $newAct$ = void
2: for $i \in memoryBase$ do
3:    select $\bar{i}$ with $max_i$ match($memoryBase$, $sit$)              /* get most similar situation */
4: endfor
5: if $\bar{i} \neq$ void                                                  /* if there exists such a situation */
6:    if $perfChg(\bar{i}) > 0$
7:       $newAct$ = generateSimilarAction($memoryBase$, $\bar{i}$)          /* behave accordingly */
8:    else
9:       $newAct$ = generateOppositeAction($memoryBase$, $\bar{i}$)         /* behave differently */
10:   endif
11: endif
12:
13: if $|perfChg| > threshold3$
14:    insert($memoryBase$, [$sit, act, perfChg$])                        /* store significant situation */
15: endif
16: computeDecay($memoryBase$)                                            /* blur older memories */
17: return $newAct$
```

contrast to case-based reasoning, (see e.g., [GS98]), no generalization of previous situations is performed, but experiences of the agent are directly stored as *prototypical situations.*

If performance significantly increases or decreases during a run, we store a tuple consisting of the occurred situation, (i.e., the resource allocation), the performed action (i.e., resource shift). Such experience can be used in a new decision situation of the RAL: if the decision making agent has no other information (such as from bottleneck analysis, etc.), it can make use of previous experience in order to avoid performing a random step (Step 13 in Algorithm 3).

To derive a heuristic, the current situation is matched against all prototypical situations; whenever a similar situation is found (according to a match-making function), the effect of the action chosen in that particular situation is taken as advice for the current situation: if previously performance had increased, the resources are shifted accordingly. On the other hand, a performance decrease in a prototypical situation can be regarded as an indication to move towards the opposite direction. If no similar situation can be found (i.e., the similarity measure is beyond a certain threshold), either another heuristic is applied or a random step has to be taken.

In Algorithm 4, the *applyHeuristics* procedure shown the integration of a maschine learning technique: Initially, *memoryBase* is empty. The procedure takes as input a situation (i.e., a position in the search space), an action (i.e., a step direction and size) and the difference of the current performance from the previous one. For the current situation, the *match* procedure in Line 3 returns the most similar situation stored in the memory base, if there exists one that is similar enough. If this is the case, a new action is generated (Line 7 or 9).

Completely independent from the outcome of the search for similar situations, the current situation is inserted into the memory base (Line 14), if its performance change has been significantly good or bad.

Another effect of the dynamically changing environment is the decreasing relevance of older experiences. For this reason, we annotate each experience with a time stamp: the similarity to older experiences is decreased according to a penalty function. Experiences that are older than a certain age are completely removed from the memory base. Furthermore, the memory base can be completely erased if certain specified triggering events occur. This is done in procedure *computeDecay* (Line 15). This technique will also be used and refined in the integration of GRAIL into MoTiV-PTA (Chapter 6). For instance, the learning procedure does not only consider single allocation changes, but whole sequences of modifications as actions to be learned.

3.4.4 Other Techniques

In the previous section, we have derived Greedy-RAL and its extensions for the given setting requirements. In this section, we present alternative RALs for different settings. However, we will not use them in the case study of Chapter 6. These techniques can be employed for other applications, some of which are used in the case study of Chapter 7. After having presented these approaches, we discuss their usefulness for different settings.

Traditional Search Strategies

A whole range of search strategies have been derived, which may be incorporated as RALs if supported by the application domain. Besides the well-known *depth-first search* (where Greedy-RAL is an instance of) and *breadth-first search* strategies, more specialistic strategies have been developed: *Iterative deepening* combines the former two strategies by running a series of depth-first searches with increasing depths. *Bi-directional search* pre-assumes a well-defined start and goal states and performs concurrently breadth-first searches from start and goal states. In order to overcome the problem of local optima, *simulated annealing* introduces the idea to perform from time to time random jumps in the search space whose frequency and step size decrease over time.

Complex Decision Making

A different RAL approach is to employ *decision-theoretic planning* if the performance of agents is not purely measured in terms of a real-valued function, but can be expressed through a combination of symbolic characterizations of desired system states (*goals*) and numeric priorities between them (*utilities*).

Decision-theoretic planning can be used as a means to pro-actively assign resources. Such behavior is inevitable if a certain desired societal configuration can be specified as a goal. Positions in the search space must be represented as states, e.g., the initial and goal states; resource shifts are considered as actions which have to be composed into plans. To reach the goal, steps have to be planned from the current configuration to the goal configuration.

Each action is annotated with preconditions, i.e., applicability conditions, and effects, i.e., configuration changes arising from the execution of that action. Planning procedures (e.g., the event calculus; see [Esh98]) can now be applied to create a plan that is applicable in the current situation. Such an approach is suitable for achieving (at least in principle) a global optimum in the search space: the plan corresponds to a path from the current situation to the global optimum.

Finding such optimal solutions results in high computational complexity: the planning of a sequence of system reconfigurations consumes also computational resources. Therefore the time and space consumption of the planner's computation have to be mapped to the abstract resources available to the planner. Jung [Jun98, Jun99] shows how they can be used in modeling resource-adapting agents for the RoboCup [KAK$^+$97] domain.

Combining Complex and Simple Decision Making

Based on the allocation of abstract resources we can combine the complex decision making procedure with Greedy-RAL by developing a two-fold allocation procedure to be integrated on *every stage* in an agent society. This procedure follows Russell and Wefald's proposed meta-level reasoning architecture in every agent of the hierarchy, including the monitor agents (Figure 3.7).

In our case the object level task for a monitor agent is the resource assignment for the lower stages. At each social stage, the complex decision making procedure is directed by the simple decision making mechanism as its meta-level: all possible *object-level* actions for the *complex decision making unit* cover resource allocation for lower stages while the *simple decision making unit* on the *meta-level* reasons about the optimal adjustment of the complex decision maker.

De-centralized Allocation Strategies

A RAL can either be performed by a single device (see Greedy-RAL which is performed by each agent on its own) or it can be performed in a de-central fashion

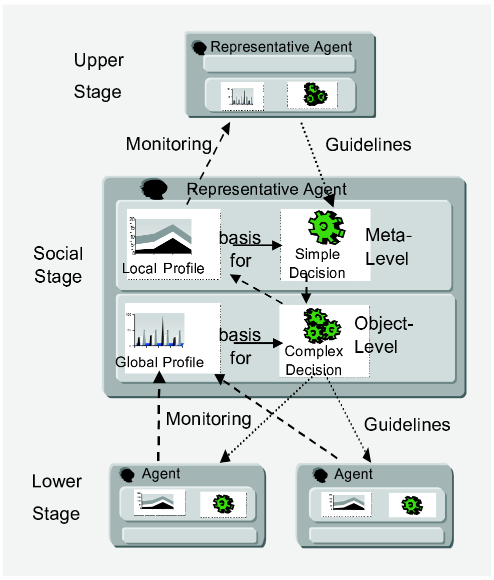


Figure 3.7: Decision procedures in a hierarchical resource control

through negotiation of the members of the hierarchy. We now discuss various options of the latter case and comment on their suitability for different settings. In particular, we distinguish between *cooperative* and *non-cooperative* settings: In a *cooperative setting*, the participants have no local utility valuation. They are eager to maximize the utility of the group, and, hence, a utility measure is needed that enables the agents to decide locally whether a trade is globally beneficial. In a *non-cooperative setting*, each agent tries to maximize its local utility. In this case, we must ensure the possibility for utility transfer via side-payments.[6]

Market mechanisms Market-based mechanisms can be used to distribute and re-distribute tasks or resources among a group of agents. The general idea of these mechanisms (see [FRV98] for an overview) is that a manager agent "advertises" a task or resource; the other agents bid for the resource or for the execution of the task which is then allocated such that the quality of the overall distribution increases.

We distinguish between coordinated and uncoordinated market mechanisms: in an uncoordinated market, agents negotiate and decide locally whether or not to agree on a deal. Sandholm [San96] proposed a trading mechanism to exchange tasks between agents; he showed that an optimal allocation is possible under certain circumstances.

[6]It might seem contradictory to consider self-interested society members since we require a common overall goal. However, such a common goal does not prevent conflicting goals of minor priority.

In a coordinated market the monitor agent moderates the trading process and provides the global utility function. *Simulated trading* [BHM92] is a randomized algorithm that realizes such a market mechanism. A central instance (the monitor agent in our case) collects the trading offers and evaluates them such that the global quality increases. The trading proceeds over several rounds, each of which consists of a number of decision cycles: in every cycle each agent submits one offer to sell or to buy a task. At the end of a round the monitor agent tries to match the sell and buy offers of the agents in oder to increase the quality of the global solution. As in simulated annealing, a relaxation value that decreases from round to round can be specified: in early rounds the central instance might accept a worsening of the global solution which is helpful to leave local maxima in the solution space. Nevertheless, those maxima are saved. If the algorithm terminates before a better solution is found, the best solution hitherto is returned, hence simulated trading is an anytime algorithm.

Game theoretic allocation mechanisms For these classes of mechanisms, the central instance, the monitor agent in our case has to mediate between agents and to allocate the resources or tasks to the group members on the basis of reported valuations. Again, we can distinguish between the cooperative case where truthful behavior is guaranteed, and the non-cooperative case where the agents may try to increase their own benefit at the expense of others. In the latter case, for the sake of global performance, it might be useful to apply *truth revealing mechanisms.* Some sort of currency is needed that allows an explicit utility transfer.

One of the classic protocols for cooperative settings is the *contract net protocol* [Smi80]. It assigns a task or resource to a single agent competing with a number of other possible contractors (the member agents, in our case). The manager (the monitor agent) announces the resource or task to be allocated to the contractors which then submit a bid and state their cost of the bid. The manager grants the item to the bidder that stated the best offer and all other bids are rejected.

For a non-cooperative setting, *auction-based protocols* are better suited for the distribution of tasks and resources. In the *sealed-bid-first-price* auction, all bidders submit a sealed bid and the bidder who has offered to pay the highest price makes the deal and pays the price he has actually bid for. In the *sealed-bid-second-price* auction (also called *Vickrey auction*) the bidder that has submitted the highest bid wins the competition but will only be charged the price the next bidder was willing to pay. The *English auction* starts with the minimal price the auctioneer would accept and the bidders successively out-bid each other until a single bidder is left. In the *Dutch auction* the auctioneer initially starts with a very high price which he lowers stepwise until one of the bidders accepts to buy the item at the current price. The Vickrey auction is an *incentive compatible* mechanism: A bidder's dominant strategy is to reveal his real valuations to the auctioneer, i.e., it is well suited for a non-cooperative setting. The Vickrey auction is logically equivalent to the English auction, assuming a small step size in price increase.

Techniques from Operations Research

Operations research is concerned with minimizing or maximizing a function of certain variables (i.e., the objective function) which may be subject to equality or inequality constraints on other functions of the same variables (constraint functions). Several important cases are distinguished: Either the solution values can take only whole number values for some or all of the variables (*discrete programming* or *integer programming*) or the solution values can take any real values (*continuous programming*).

The latter case can be divided into three sub-cases: In *non-linear programming*, the functions involved are arbitrary. If all functions are convex, more efficient *convex programming* techniques can be applied. Efficient approaches are for instance the *quasi-Newton method*, *steepest descent*, *line search*, or the *Davidson-Fletcher-Powell method*; some of them require knowledge on the first and second derivative of the target function (for an overview see [Hor87a]). If all functions involved can be modeled as linear functions, *linear programming* approaches can be used. Besides the classic *simplex method* and its derivatives, the *ellipsoid algorithm*, *interior point iterations* and others have been developed ([Sch86] provides an overview).

If a RAL can be modeled as a linear or non-linear problem, the various approaches can easily be applied to let a resource-adapting agent perform RAL steps, since we have already specified a RAL problem as a (very general) non-linear problem in the beginning of this section.

3.4.5 Discussion

All the presented techniques can be used for resource assignment. They differ with respect to the degree of central control. Market mechanisms require no central control unit, except for the representation of the group to higher instances. The other approaches presuppose such a control unit whose competence, however, varies. In a non-cooperative setting, the representative agent has only administrative competence and therefore mechanisms that enforce cooperative behavior have to be applied. In a cooperative case, decision power is split: the representative agent decides on the basis of the local calculations of the member agents. In the central Greedy-RAL approach, in traditional search approaches, OR techniques, and in the decision-theoretic approach, the monitor agent has the full resource allocation competence. Local information is provided by the members and is only used as a heuristic.

As stated above, Greedy-RAL can be seen as an instance of depth-first search, its heuristic-based extension as an instance of best-first search. These types of search approaches can cope better with changes in the environment than approaches that base on breadth-first strategies, such as iterative deepening or bi-directional search. Since there is no well-defined end of the search, simulated annealing can hardly be used.

Techniques from operations research are highly efficient and proven approaches that can be used if some vital pre-conditions, (e.g., linearity or convexity of all functions) hold. The Greedy-RAL approach roots in the steepest descent method, a convex programming technique. However, as we have shown, Greedy-RAL can also cope with some violations of assumptions required for steepest descent; it subsumes that approach.

The decision-theoretic approach is only tractable, if actions can be modeled as transitions form one explicitly represented state to another. By modeling a global optimum as a goal state, the global optimum can be targeted; however running such a strategy leads to high computational costs. Therefore we will not use this technique in any of our case studies.

More decentral approaches are better suited to cope with highly complex allocation problems, as they can often be reduced to a set of problems with less complexity (divide-and-conquer). On the other hand, the use of these methods may lead to sub-optimal results. Hence, the choice of the appropriate mechanism depends on the nature of the application, inducing a trade-off between optimality and complexity. For the MoTiV-PTA case study (Chapter 6) we will employ Greedy-RAL with its presented extensions, while in the TeleTruck and IFMS case studies (Chapter 7) we will use de-centralized RALs.

3.5 A Resource Implementation Algorithm (RIL)

In the previous section we have presented different techniques to allocate abstract resources. In this section, we present a hierarchical framework RIL that allows us to implement the resource allocation in an agent society. First, we elaborate how to build up a hierarchical agent society, then we present a scheme to distribute abstract resources in this hierarchy. We show the correctness of the approach and analyze its complexity.

3.5.1 Construction of a Hierarchy in the Agent Society

In Section 3.2, we have characterized abstract resources as environmental entities that impose interdependencies between agents. In GRAIL, these interdependencies have to be explicitly specified by a system designer, just as he has to specify the interactions among agents in a regular MAS. The resulting constraints can then be used to model *automatically* the desired hierarchy of the agent society. Finally, it is again the designer's job to correctly model the constraints between the various resource allocations. This task cannot be solved in an automated fashion since the designer has a wide range of possibilities how to model the resources and their meanings (see Section 3.2.3).

Basic Principle

After the specification of the abstract resources, we can describe and model the agent society by a *directed graph*, where agents are represented by *nodes* and resource dependencies by *arcs*: an agent α_j is a successor of α_i, if the resource allocation of α_i can influence *directly* the behavior of agent α_j, (for instance, α_i has permanent control over a fuel supply that α_j needs for further work.) Since communication among agents is a crucial property of a multi-agent system, often an agent's behavior depends on results of communications with other agents (e.g., in auctions). In such cases, *information* has to be modeled as an abstract resource.

For our purposes, we need one or more hierarchies: We will apply a top-down resource allocation which would end up in an infinite loop if applied to a cyclic graph. We do not require a tree-like structure: there may be more than one agent whose behavior does not depend directly or indirectly on the resource allocation of another agent (node with in-degree 0). This may lead to disjoint hierarchies, in which case the overall system performance depends only on the cumulated quality of the performances but not on direct interaction between agents of different hierarchies. Also, agents may depend on several other agents, possibly in different hierarchies (nodes with in-degree greater than 1).

Since resources are modeled by arcs, our focus on this graph is arc-centered, not node-centered; therefore, as a special property, we allow more than one arc pointing from one node to another one, representing dependencies imposed by different abstract resources. We call such a graph a *directed acyclic multi-graph (DAMG)*, since the collection of arcs is actually a multi-set. As a side-effect, when replacing cycles, we cannot consider nodes in a cycle as members of an equivalence class since they might have different numbers of incoming and outgoing arcs.

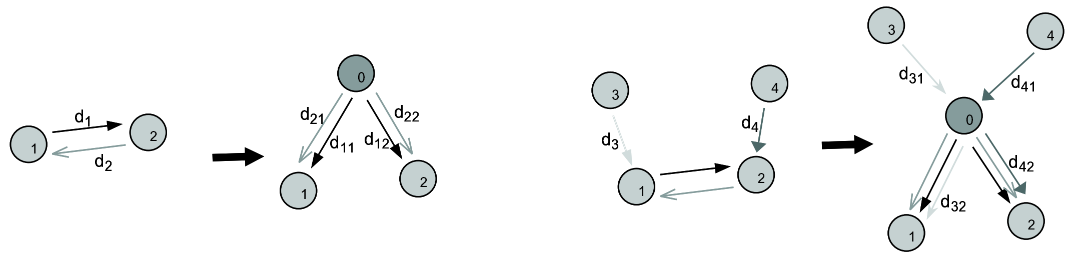


Figure 3.8: Replacing dependency cycles

Replacing Dependency Cycles

The following procedure transforms a directed graph into a DAMG which is suitable for RIL. The key idea is that for an optimal cooperative society with benevolent agents, resource allocation power can be given to a higher-ranked instance (the monitor agent in our case) without loosing system functionality. Hence, a dependency between two agents can be also be represented as dependencies of agents from the higher instance. Therefore, cyclic resource dependencies d among agents $\alpha_1, \ldots, \alpha_n$ can be managed by introducing a new monitor agent α_0 and replacing each dependency arc from α_i to α_j with new dependency arcs from α_0 to α_i and from α_0 to α_j where the monitor agent α_0 has to control boundaries of the new dependencies.

For instance, consider two agents α_1 and α_2 in the left-hand side of Figure 3.8. While α_1 controls a supply of fuel, which is needed by both (displayed by dependency arc d_1), α_2 controls the resource water, also essential for both agents (displayed by dependency arc d_2). For the sake of clarity we introduce different gray scales or shapes of arcs to express different dependencies. Assuming a cooperative agent society, we can introduce α_0 that now controls both resources in a way that the amount allocated to α_1 plus that the amount allocated to α_2 may not exceed the overall supply of each resource: The dependency d_1 is converted to two dependencies d_{11} and d_{12} of α_1 and α_2 from α_0 which represent the need of α_1 and α_2 of the fuel resource which is now controlled by α_0. Similarly, d_2 is converted to d_{21} and d_{22}.

In order to assign efficiently resources, the new monitor agent needs information on further dependencies of its sub-ordinated agents from other agents. Therfore, we re-direct each of these dependencies to the monitor agent and introduce new dependencies from the monitor agents to “its” member agents to further pass control: All additional dependencies of the agents α_i from some other agents $\alpha_m \notin \{\alpha_1, \ldots, \alpha_n\}$ are removed and replaced by appropriate dependencies of the monitor agent α_0 from α_m and by additional dependencies of α_i from the monitor agent α_0.

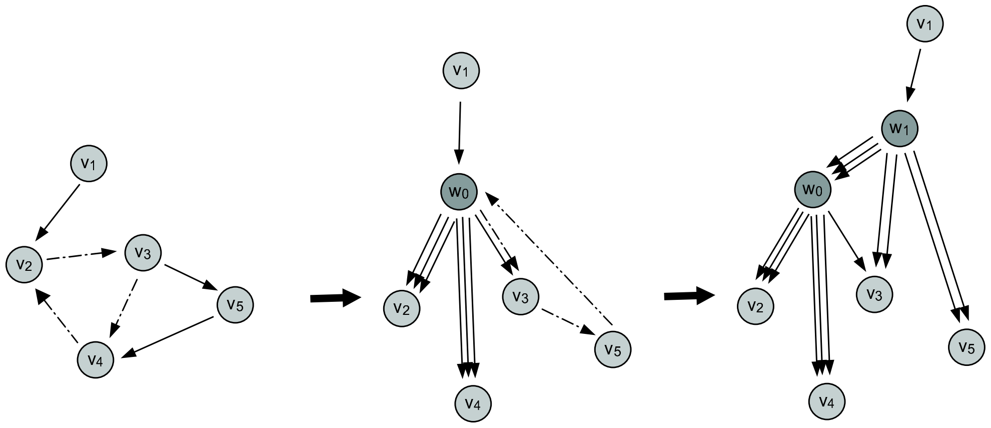


Figure 3.9: Recursively replacing dependency cycles

The right-hand side of Figure 3.8 illustrates such a case: given the previous example, further suppose that α_1 depends on some agent α_3 in terms of electricity (dependency d_3) and α_2 depends on agent α_4 in terms of coal (d_4). For an appropriate resource allocation to α_1 and α_2, the new agent α_0 must consider constraints on electricity and coal allocation imposed by α_3 and α_4. So we convert electricity dependency to d_{31} of α_0 from α_3 and coal dependency to d_{41} of α_0 from α_4. Resource guidelines from α_3 and α_4 must be delegated to α_1 and α_2. Hence we introduce electricity and coal dependencies d_{32} and d_{42} of α_1 and α_2 from α_0.

Recursively Replacing Cycles

In order to transform the whole graph, this replacement strategy is performed in a top-down and depth-first manner. Algorithm 5 describes the exact approach while Figure 3.9 visualizes the strategy: regular agents are represented by nodes v_i while monitor agents are symbolized by nodes w_j. Dashed arcs of a cycle denote that this cycle is going to be replaced in the next step. Figure 3.9 shows the major steps of the development of the graph G_{new} in Algorithm 5, where the symbol $\bigoplus$ expresses the joining of an item to a multi-set. The symbol $\setminus$ expresses the removal of only one occurrence of an item from the multi-set. In the initial graph (on the left-hand side of Figure 3.9), we have two cycles to be replaced, namely (v_2, v_3, v_4) and (v_2, v_3, v_5, v_4). We decide arbitrarily to replace the former cycle first by w_0. (middle graph of the figure). Therefore the second cycle transforms to (w_0, v_3, v_5) which is replaced by w_1 (right-hand side of Figure 3.9).

The right multi-graph appears to be rather condense. Note however, that the original graph contains a relatively high number or cycles. Furthermore, this approach is more a guideline for the system designer how to generate a DAMG;

Algorithm 5 `transform-graph (Graph G)`

```
1: Graph $G_{rest} = (V_{rest}, E_{rest}) := G$
2: Graph $G_{new} := G$
3: for $v \in V_{rest}$ do computeIndegree($v$) endfor
4: while $V_{rest} \neq \emptyset$
5:    if exists $v \in V_{rest}$ with indegree($v$) $\neq \emptyset$
6:       $V_{rest} = V_{rest} \setminus \{v\}$
7:       for $w \in V_{rest}$ with $(v, w) \in E_{rest}$ do
8:          $E_{rest} = E_{rest} \setminus \{(v, w)\}$
9:       endfor
10:   else                                                    /* there is a cycle*/
11:      Graph $G_{cyc}$ =find-cycle($G_{rest}$)                        /*traditional BFS*/
12:      $V_{rest} = V_{rest} \cup \{v_0\}$                              /*for a new node $v_0$*/
13:      $V_{new} = V_{new} \cup \{v_0\}$
14:      for $(v, w) \in G_{cyc}$ do      /*replace arcs of cycle with new arcs of same grey scale*/
15:         $E_{new} = E_{new} \setminus \{(v, w)\}$
16:         $E_{rest} = E_{rest} \setminus \{(v, w)\}$
17:         $E_{new} = E_{new} \bigoplus \{(v_0, v)\}$
18:         $E_{rest} = E_{rest} \bigoplus \{(v_0, v)\}$
19:         for $u \in V_{new} \setminus V_{cyc}$ with $(u, v) \in G$ do /*process arcs directing towards the cycle*/
20:            $E_{new} = E_{new} \setminus \{(u, v)\}$
21:            $E_{rest} = E_{rest} \setminus \{(u, v)\}$
22:            $E_{new} = E_{new} \bigoplus \{(u, v_0)\}$
23:            $E_{rest} = E_{rest} \bigoplus \{(u, v_0)\}$
24:         endfor
25:      endfor
26:   endif
27: endwhile
```

this guideline cannot consider the semantics of resource dependencies which may lead to a reduced complexity of the graph, if modeled manually. The presented strategy in facts leads to a directed acyclic multi-graph, as the following corollary shows which is proven in Appendix A.

Corollary 3.5.1 *The proposed cycle replacement technique of Algorithm 5 terminates and the resulting multi-graph does not contain any cycles.*

Furthermore, the next corollary (which is also proven in Appendix A) shows that the resulting graph displays the interdependencies correctly: If a new agent α_0 is introduced to model resource dependencies among a group of other agents $\alpha_1, \ldots, \alpha_k$, new arcs from α_0 to $\alpha_1, \ldots, \alpha_k$ are introduced. The creation of a new agent into the society can be done recursively, hence we show that if there has been an arc between v_i and v_j in the original graph G, then there will be an *undirected path* of the same gray scale[7] from v_i to v_j over only new nodes

[7]We assume that new arcs are introduced in the same grey scale as the arc currently to removed from the cycle.

(for the arc (v_1, v_2) the path $\{v_1, w_1, w_0, v_2\}$ in the example of Figure 3.9); the arc (v_3, v_5) is transformed to the path $\{v_3, w_1, v_5\}$). This property states that resource allocation constraints are taken into accent in the new structure.

Corollary 3.5.2 *If an original dependency graph contains an arc (v, w), then the resulting multi-graph after an application of Algorithm 5 contains an undirected path $(v, v_0, \ldots, v_n, w)$ of the same grey scale where the v_i are new nodes.*

As Figure 3.9 shows, an arc in the original graph can be converted to a chain of arcs with different numbers of intermediate nodes (that stand for monitor agents). Corollary 3.5.2 proves that all arcs in this chain are of the same color, i.e., they represent the same dependency as the original arc. Intermediate monitor agents delegate the dependency correctly; hence the *varying number* of intermediate monitor agents is of no concern.

3.5.2 A Resource Implementation Scheme for a Hierarchical Agent Society

In the previous section we have demonstrated how to use the concept of abstract resources in order to derive a hierarchy in a benevolent agent society. We now use this hierarchy to install a flexible resource adaptation scheme.

We can use the resulting acyclic structure to build a hierarchical resource control:[8] This DAMG induces the order of the sub-computations. We employ a *depth-first allocation scheme* The top-most agents α_{1i} perform one step to optimize their resource allocation. Then the children of α_{1i} re-allocate their resource allocation in their subspaces recursively until optimization has been performed on the lowest stage.

Once agents of the lowest stage have found a locally optimal resource allocation, *backtracking* is performed: Agents on the stage above perform another optimization step which induces a re-organization of the bottom-most agents. This backtracking procedure is propagated up to the highest societal stage. The recursive call of lower-ranked RILs can be performed in parallel since we can assume the independence between the searches in the sub-spaces.

The procedure is described in pseudo-code in Algorithm 6. For this recursive depth-first search algorithm, we assume a hierarchy of benevolent agents and an operational characterization of the abstract resources as dimensions in a search space. Furthermore, we require a resource allocation algorithm RAL that takes

[8]This is an idealized procedure, which has to be slightly modified for practical use. We will discuss these modifications later.

Algorithm 6 `RIL (Agent` a`, SearchSpace` ss`)`

```
1: Boolean $solutionFound$ := false
2: SubSearchSpaceVector $sssv$
3: while (not $solutionFound$) do
4:    ($sssv$, $solutionFound$) := RAL(SS)        /* perform one step in RAL */
5:    for each successor agents $a_i$ do          /* for all dependent agents */
6:       RIL($a_i$, $sssv_i$)                      /* perform RIL recursively */
7:    endfor
8: endwhile
```

a (sub-) search space as an input parameter and returns a flag indicating the success of the search and a new search space for each agent that is sub-located to the current agent. We require that all data necessary for the RAL in use is stored locally there (e.g., the number of sub-ordinated agents, the current position in the search space, a performance history if needed, etc).

Initially, the RIL is started for any top-level monitor agent and then propagated through the hierarchy. The algorithm terminates if a (local) optimum has been found for the whole agent society.[9] It does not produce any output since modifications of the agent society occur as side-effects. The variables ss and $sssv$ are local parameters or instance variables that express the search space and the vector of sub-search spaces for each agent in the hierarchy. These sub-search spaces are computed in the RAL (see for instance Line 15 of Algorithm 1 in Section 3.4).

In the case of backtracking, the new sub-branch in the search space often shows similarities to the previous one. We can use this fact as a heuristic for the definition of the starting point in the new search space by mapping the current position in the old sub-search space to a position in the new one.

3.5.3 Complexity Analysis of RIL

In this section, we analyze the complexity of the hierarchical RIL scheme. Since a variety of different RALs can be plugged into this control scheme, we treat the complexity of the RAL as a function d_{ral} of the number of resources and the (discretized) number of possible values of a resource. We define the following quantities:

- Let r be the total number of abstract resources modeled in a multi-agent society, n_i be the number of possible values of the i-th resource and $n = max_i(n_i)$ be the largest number of possible values.

[9]In Chapter 4 we will modify the approach to run ad infinitum to cope with changes in the environment.

- Let G be a DAMG, where a node stands for an agent α_i and an arc denotes an abstract resource R_{ij} describing a resource dependency of α_j from α_i. Let $\tau_i = \text{outdeg}(\alpha_i)$ the number of agents (or *children*) depending on α_i and $\tau = min_i(\tau_i)$ the smallest number of such dependencies. $l = \log_\tau(r)$ denotes an upper threshold of the depth of G.
- The function $d_{\text{ral}}(n, r)$ denotes the number of search steps that have to be taken in order to find the optimal resource configuration for a certain sub-search space of r resources and at most n possible values per resource.

In this definition we make some assumptions, which however do not lead to a loss of generality but to a gain of tractability and readability: In principle, depending on the type of a resource (see Section 3.2), there may by infinitely many possible values of a continuous resource. By choosing n_i appropriately high, but finite, we still can assume tractability. In the following worst case analysis, we assume all resources to have n values. Also, we now assume all agents to have only depending agents in no more than τ relations, which is again the worst case.

Obviously, the complexity of a centralistic approach to find the optimal resource allocation depends on the chosen RAL, the number resources and the number of possible values of the resources, as the following corollary shows (a proof is given in Appendix A):

Corollary 3.5.3 *The worst case run-time complexity of a centralized resource integration approach is* $O(d_{ral}(n, r)))$.

The following corollary (also shown in Appendix A) denotes the complexity of a decentralized and recursive approach. The key factor is that the run time is no longer primarily dependent on the number of resources r (and the number n of possible resource values), but on the branching factor τ of the hierarchy (and n).

Corollary 3.5.4 *The worst case complexity of a recursive decentralized resource integration approach of Algorithm 6 is* $O(d_{ral}(n, \tau)^{log_\tau r})$.

The corollary shows that, if τ can be restricted, the exponent of the complexity function is reduced from linear to logarithmic in r by switching from a centralized to the decentralized approach. Hence, the extra effort for performing backtracking (and discarding already found efficient configurations on lower stages) is more than compensated by the parallel configuration searches on independent sub-search spaces by lower-ranked agents.

Still, depending to the performance of the employed RAL, the overall performance can be beyond tractability for large-scale systems with many agents and therefore many inter-dependencies among them. However, we have only observed the extreme worst case: in average, not all resources have n values, not all agents have children in τ dependency relations. Furthermore, efficient positions in a sub-search space can be mapped to promising starting positions in a new sub-search space in case of backtracking and search space modification.

3.5.4 Discussion

We are not necessarily interested in the globally optimal resource configuration for the whole society; a relatively high local optimum will often do as well. Hence we might not need to solve the search space completely. In the case study, the various search spaces are not traversed completely; instead heuristics are employed to determine promising resource allocations. These issues are presented in detail in the section on RALs (Section 3.4).

This hierarchical approach has major advantages over a centralistic one:

- We can split the search space of resource allocations in order to derive partition that *do not depend* on each other. So we can break down complexity.
- We can employ agents to do the resource allocation of sub partitions *in parallel.* This can really be a performance benefit, if for instance, the agents run on different machines which are idle.
- If the environment changes, usually only a restricted set of agents are effected by this change, and hence have to re-optimize their resources. The proposed RIL supports this local re-configuration. In a centralized procedure however, it would be very difficult to estimate which part of the search space can be pruned.

The presented RIL approach shows some similarities to *branch and bound* algorithms, used in operations research for e.g., *integer linear programming* (An overview is given in [NW88]). The key idea of such a technique is to cut the search space into partitions and to exclude some partitions from being traversed by solving selected sub-problems (possibly in parallel) with some standard problem solver (e.g., a simplex algorithm) and by drawing conclusions on the position of the global optimum. However, the main difference lies in the dynamics of the system: In RIL, if there has been a change in the environment, a RAL can be reset in order to cope with changes. A branch-and bound approach cannot offer such a flexibility; in the case of an environmental change, the drawn conclusions on which part of the search space be neglected have be undone and the search has to be restarted.

Bottom Line

Our approach to achieve a *scalable society* of artificial agents is based on an *efficient distribution of resources* within the agent society. We do not treat resource distribution as a first class action but as *meta-level* action which interferes indirectly with the object-level behavior of the agent society.

We have extended resource concepts to the notion of *abstract resources* and have derived the term *bounded-optimal agent society* from the concept of the single bounded-optimal agent.

Our multi-folded resource adaptation scheme can be mapped on the structure of the agent society: a hierarchical *resource implementation scheme (RIL)* embeds on every stage in the hierarchy a *resource allocation algorithm (RAL)*. Due to this generic design, different RALs may be applied. We have discussed and classified a wide range of RALs and have argued for *Greedy-RAL* to be the algorithm on focus.

Chapter 4

A Framework for Agent-Based Simulation and Resource Adaptation

In this chapter, we present the generic agent framework SIF for agent-based simulation and its extension SIFIRA for the construction of resource-adapting agent societies. As a complete JAVA^{TM} package, the SIF development kit can be downloaded from `http://www.dfki.de/~sif` under the GNU license agreement.

4.1 The Social Interaction Framework (SIF)

4.1.1 Multi-Agent Systems for Simulation Purposes

There are many different approaches to realize a computer-driven simulation, ranging from using an off-the-shelf simulation tool to implementing a simulation from scratch. Recently, decentralized approaches are becoming more popular to simulate complex phenomena.

According to Troitsch [DGMT95], simulation tools are used mainly for two rather conflicting purposes: On the one hand, for the better understanding of observed real-world phenomena and on the other hand, for advice and decision support. In order to satisfy these diverging demands, a whole variety of simulation approaches has been developed since the early 1960's.

Pre-built products such as the classic flow diagram system *GPSS (General Purpose Simulation System)* [Gor69] or more recent graphic-oriented *visual interactive modeling systems (VIMS)* provide built-in functionalities leading to little modeling effort. These systems require only little programming skills. However, limitations of such tools are the rather low execution speed, the restricted expressivity which often lead to insufficient scalability as only few entities can be modeled. On the other extreme, the most flexible approach is do-it-yourself-coding, since the developer has all freedom to model entities. But such an approach requires detailed programming skills and rather time consuming implementation and debugging.

Specialized simulation programming languages and pre-written simulation libraries for general-purpose languages are located somewhere between the extremes. The simulation designer still has a high degree of flexibility and can make use of features pre-specified in the system. Such systems allow easy access to simulation modeling for novice users as well as more complex simulation experiments for experts.

Since Shoham introduced the agent-oriented programming (AOP) paradigm, agent-oriented simulation tools offer new alternatives for simulation: Simulations based on multi-agent systems can be used to model a broad variety of domains because multi-agent systems support simulations of any granularity:

- **Micro-level simulations** use agents to represent individuals. The decision making procedure of an agent can be defined in a primitive way, depending only on a certain state of the agent and on states of neighbor agents. In such a case, an agent-based simulation tool basically works as a cellular automaton [Axe84]. However, more complex cognitive processes, such as logic-based inference capabilities or even "illogical" emotions can be modeled leading to a much more realistic simulation of human behavior.
- **Macro-level simulations** let agents represent whole individual groups or societies. Hence, macro-level properties can be modeled effectively. Through combination of micro- and macro-agents the behavior of individuals in societies can be simulated easily leading to a natural bridge of the micro-macro-gap.

A very recent trend is derived from the field of *Virtual Reality (VR)*: In *co-habited virtual worlds (CHVW)*, autonomous agents and avatar agents (i.e., agents that represent a human user and that may be controlled by the user) interact in a distributed network. This movement puts requirements the virtual world simulation engine, and the agent architecture:

- **Transparent networking:** Open multi-agent platforms have to support the transparent migration of agents and other system components within a distributed environment.

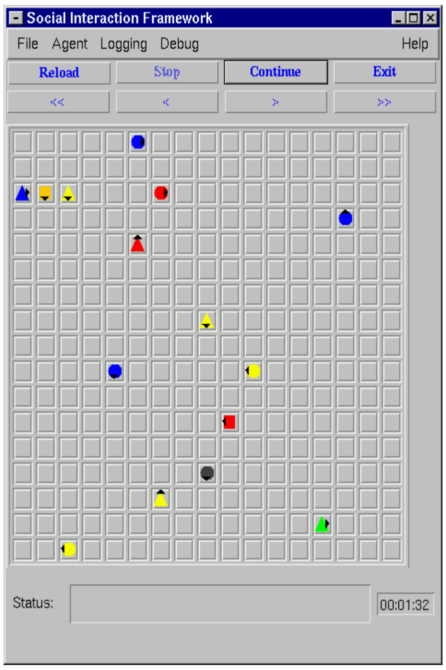


Figure 4.1: SIF screen shot

- **Fine-grained action and perception facilities:** Quasi-continuous acting and perceiving is a vital prerequisite for a realistic simulation environment. Agent perceptions have to be generated for a local perspective of the agent. Furthermore, the duration and the outcome of actions must not be deterministic in advance, but have to be modeled by considering the current environment.

- **Broad platform:** CHVW requires common, open platforms, such as those based on JAVA^{TM}, VRML'97, and CORBA in order to reach as many users as possible.

- **Multiple Users:** Co-habited virtual worlds require a sophisticated visualization and user interfaces which should reflect the perceiving and acting of avatar agents.

These new trends in simulation have influenced DAI as new demands have come up. The multi-agent framework SIF has been designed to meet these new demands. SIF is a JAVA^{TM} library which enables agent-oriented programming for quick and platform-independent simulation modeling (Figure 4.1 shows a screen shot of a simple scenario). SIF provides a multi-threaded framework where basic agent functionalities (such as sensing, performing actions, communication, etc.)

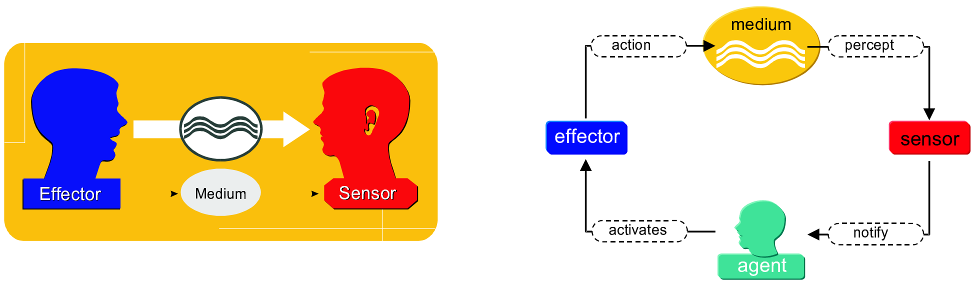


Figure 4.2: The EMS model and its realization

are already provided. Due to the generic layout, any type of agent can be connected to the framework. Next, we present the SIF system in detail; in Section 4.2 we extend the framework by a toolbox for implementing resource-adapting agent societies.

4.1.2 The Effector-Medium-Sensor Model of Interaction (EMS)

SIF's underlying basic mechanism is the *Effector-Medium-Sensor (EMS)* model based on Russell and Norwig's definition of an agent in [RN95]:

> An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.

EMS provides an appropriate abstraction of an agent acting and interacting with its environment and other agents. The central component of the architecture is the *medium*. *Effectors* emit *actions* to the medium, which in turn sends the effect of the actions as *percepts* to effected *sensors*. According to the EMS paradigm, an *agent* is equipped with a number of effectors through which it can act in its virtual environment, and with a number of sensors through which it can perceive the state of the environment (see Figure 4.2).

Formalization of EMS

We define agents and multi-agent environments in a uniform state based mathematical model. For this framework, we assume a discrete time scale and define a time step as the transition from one point of the time scale to the next. During each step, each agent receives its local perception through its sensors. The agent's

An *agent* α_i is a tuple (S_i, P_i, A_i, ϕ_i) of the set of possible states S_i, the sets of perceptions P_i and actions A_i, and its agent function $\phi_i : S_i \times P_i \to S_i \times A_i$.

A *multi-agent environment* is a tuple $(\mathcal{A}, \mathcal{E}, \Pi, \Delta)$ of the set of all agents $\mathcal{A} = \{\alpha_1, \ldots, \alpha_n\}$, the set of environmental states $\mathcal{E}$, a perception function $\Pi : \mathcal{E} \to (P_1 \times \cdots \times P_n)$ and an environment function $\Delta : \mathcal{E} \times (A_1 \times \cdots \times A_n) \to \mathcal{E}$.

Figure 4.3: Specification of an agent and a multi-agent environment

action and its new state are determined from its current state and from this perception; the action is performed by the agent's effectors (as specified in the first part of Figure 4.3).

The surrounding of an agent is explicitly represented in the *environment.* In SIF, the *world server* manages the environment and is responsible for the computation of the environmental transition function (as specified in the second part of Figure 4.3). Of course, we could have combined the perception function Π with the environment function Δ in analogy to the definition of ϕ. However, we often refer to Π and Δ separately, as this choice supports readability.

According to the specification, the state of the world changes with the actions of the agents:

$$e' = \Delta(e, a_1, \ldots, a_n)$$

describes the successor environmental state, therefore

$$s_i' = \phi_i^1(s_i, \Pi^i(e))$$

describes the successor states[1] of the agents for all i in dependence of the current state and the new perception of the agent. The state transition function $\bar{\Delta} : \mathcal{E} \times S_1 \times \cdots \times S_n \to \mathcal{E} \times S_1 \times \cdots \times S_n$ which is defined as $\bar{\Delta}(e, s_1, \ldots, s_n) = (e', s_1', \ldots, s_n')$ unifies the states of the agents and the states of the environment into *universe states.* Hence, the perception function, the agent function, and the environment function are parts of the universe transition function $\bar{\Delta}$. We will extend this formalization framework in Section 4.2 to a characterization of abstract resources and the RIL scheme.

4.1.3 The SIF Architecture

We now describe the SIF components in greater detail and characterize the kernel components; Section 4.1.4 illustrates the extension of SIF for virtual world application.

[1]An upper index n denotes the n^{th} projection of the given function.

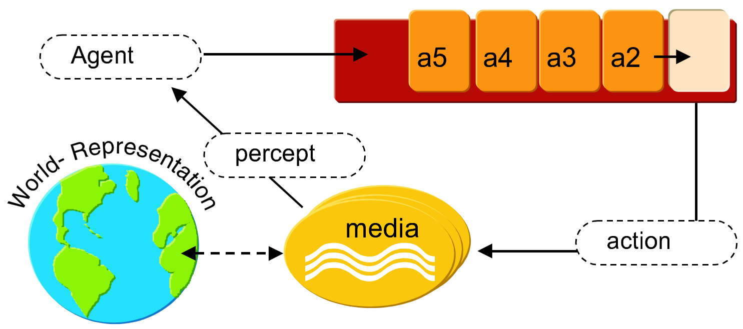


Figure 4.4: Control and information flow in SIF

The World Server updates the world representation and organizes the information flow during the simulation. It computes the effects of actions and the new perspectives of perceiving agents, and sends these perspectives as percepts to the sensors of the agents. The world server does not return any feedback on the success of some action to the emitting agent. For example, the activation of a "move" effector in general will not lead to a feedback on the success of the action, but on a new percept on which the performing agent (and any other agent perceiving this scene) has to calculate the possible success of the action. As the activation of a "move" effector in front of a wall will not lead to a position change, the agent perceives the same perception as prior to the activation of the action.

Furthermore, the world server starts and stops the simulation and provides debugging and data collection facilities. Besides pre-defined methods, the world server also allows the scenario designer to specify the agent perception of certain actions in the sub-component *sensor manager*, and to specify the change of the environmental state to these actions in the sub-component *action manager*. Furthermore, *perception ranges* can be installed: a percept is only received by a sensor if the emitter is located in the perception range of that sensor. In addition, the sensor manager can be adjusted to blur certain agent perceptions.

The world server also defines the *world representation*, a data structure which carries information on position, appearance and capabilities of objects in the environment, the specification of the environment itself and the services needed or connected to these objects. The interaction between these parts is displayed in Figure 4.4.

Agents are equipped with a number of sensors and effectors; they perceive other agents' actions filtered by the world server. Due to the generic design, any type of agent can be connected to the world server ranging from simple objects to sophisticated agents, such as for instance, InteRRaP agents.

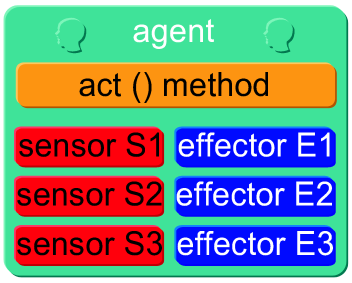


Figure 4.5: An agent in SIF

Figure 4.5 shows an example agent with effectors and sensors. The effectors of an agent could be for instance, moving, communicating, turning a robotic arm, etc. Sensors could be for example virtual vision and communication.

The Graphical User Interface (GUI) is linked to the core of SIF just like a regular agent, i.e., it communicates with the core via sensors and effectors. The received percepts tell the GUI to update certain parts of the visualization or to display agent information, etc. The GUI offers a control pad which is capable of forcing agents to perform certain actions. It also provides information windows for a convenient display of the internal states of agents.

Control and Information Flow in SIF

The agents, the GUI and the SIF kernel run independently from each other in their own JAVA^{TM} threads. To explain the control and information flow we assume that the simulation has already been started. We further assume that it is the turn of some agent to decide on its next action. As soon as the agent has committed to perform some action, it activates the corresponding effector.

The world server stores all incoming actions in an event queue that usually works on a first-in-first-out basis. Exceptions are made for user-driven action, agent communication and for the case that measurement of action duration is introduced. These exceptions will be described later.

In each step of the simulation, the *action manager* takes the next action from the queue and accesses the *world representation* to check the applicability of an intended action. If applicability is given, the action manager updates the world representation. Furthermore, a percept is sent to the GUI, specifying which part of the world has to be updated in the graphical representation before control is passed to the sensor manager. If the action is not applicable in a particular situation, the action manager will directly pass control to the sensor manager.

In either case, the *sensor manager* computes the input for all sensors that are affected by the change of the world. The input is sent as a number of percepts (representing the local perspective of the agents) to the sensors in question. As a side effect of the asynchronous implementation of SIF, the agent does not have to wait for this feedback from the world. The agent can emit new actions of any kind; however it has to take into account that these actions might not be applicable by the time they are being executed and therefore dumped by the action manager.

A special kind of action is triggered if the experimenter wants to start or stop the experiment or if he uses a control pad (which is part of the GUI) to force an agent to execute a particular action. In the former case a directive action is sent to the action manager to start or stop the control flow in the simulation. In the latter case, a directive action containing the command is posted to the queue driving the agent to execute the respective action. Here, the agent works as an avatar, controlled by the user.

Assigning Duration Values to Actions

SIF is equipped with a mechanism that simulates the duration of actions: To every action, a certain amount of simulated time can be assigned that the agent needs in order to complete the action.

If an action is emitted by an effector, the world server inserts the action to the event queue. For a correct ordering, the estimated completion time of the action is taken as the key for the insertion. For this reason, the event queue is implemented as a *priority queue*. Certain actions (for instance control actions to the GUI) have higher priority and are therefore inserted in the beginning of the event queue.

As stated above, the world server always processes the first action in the queue. So it executes the action with the earliest completion time first. However, this does not mean that an agent can be blocked from performing long-term actions, because eventually the current time will be greater than the completion time of the long-term actions and the action will be executed. Hence, a *fair* processing strategy is guaranteed.

For the realization of this concept, an internal "clock agent" is employed that sends a synchronizing action to the world server after a well-defined time interval. Modeling the clock agent as part of the simulation guarantees that the processing speed of a sequence of actions will always be constant in terms of simulated time: if the system runs faster because the JAVA^{TM} Virtual Machine (JVM) can allocate more computational resources to the simulation engine, the timer agent will also get more resources which leads to an independence from hardware or virtual machine circumstances.

Whenever the world server has processed an action, it controls the feasibility of the remaining actions in the queue. Only if all *feasibility preconditions* for an action hold for the whole time the action was in the queue, it will be executed. For instance, suppose that agent α_1 intends to jump over agent α_2 which is located close to α_1. Further suppose that a jump action takes five time units while moving lasts only three time units. α_1 can only jump over the other agent, if α_2 remains close to α_1. If however, α_2 moves away while the *jump* action of α_1 is in the event queue, this action gets removed from the queue. In a sense, the world server works as an interpreter of the event calculus as it computes preconditions and effects of events. This technique is related to the scheduling process of the umpire of the *MyWorld* system [Woo94], which however, does not only compute the effects of agent actions, but is also responsible for deriving new beliefs and intentions of the agents — a feature we do not address on purpose since we are interested in a framework for agents of arbitrary architectures.

Communication as a Special Case of Action

For modeling communication and action, different principles can be applied:

- *Action and communication are not treated in a common framework.* Often, agent architectures that focus on a single agent or that are concerned with interaction among only very few agents (such as the InteRRaP architecture) use such a model.
- *Action is regarded as a special case of communication.* Many agent architectures do not have the integrated view of agents embedded in the environment. Often, the environment is simply modeled by some "environment agent" which simulates the surrounding. In such an architecture, an action is performed by communicating to the environment agent.
- *Communication is regarded as a special case of action.* Systems that explicitly model the environment and inter*actions* between agents and the environment can easier support the treatment of communication like any other action.

For SIF, we have chosen the latter option, since it allows the most canonical mapping of natural environments to artificial agent societies: all agents communicate only via media, not directly with each other. Therefore, we can view communication as a special case of action, transmitted according to the EMS paradigm: For instance, consider the agent α_1 sending a message (a communication action) to agent α_2. To do so, α_1 activates its communication effector, which sends this action over the medium. The the world server evaluates this action, identifies the receiver and sends the message as a communication percept to the communication sensor of α_2.

The recipient has to check explicitly the reception of an effector activation. This feature prevents α_2 from actions it does not intend to perform. It still can decide what to do with the perceived information. This feature establishes agent autonomy. For performance reasons, communication percepts are treated with higher priority. Hence, they are inserted at the beginning of the queue by assigning a duration of 0 to each communication act.

Blocking of Effectors

After an agent has emitted an action via one of its effectors there may may be a time delay until the action is executed by the action manager (in particular, if a duration has been specified for the action). There are different options how to define the state of the agent and of the used effector during this delay time:

- The whole agent is not supposed to perform anything.
- The agent is blocked from emitting new actions, but can perform internal reasoning.
- Only certain effectors are blocked (possibly including the used effector).
- There is no lock at all: the agent can trigger any action.

We prefer the third option, which is used in SIF by default; all other possibilities can be selected manually. This option does not only help to keep the number of actions in the event queue small, but also leads to a more natural simulation of the behavior of the agents: For instance, an agent cannot pick up an item while already picking up something else; it needs to wait until the first action is complete. Furthermore, by setting a flag, the user can inhibit agents to use some combination of different effectors concurrently, for instance turning while moving forward.

Scripts

For easier testing and varying of experiments in the simulated world, SIF provides a simple script language to specify experiments. Scripts can be loaded, modified and reloaded without necessarily shutting down the SIF system. Any text editor can be used to modify scripts.

4.1.4 Extension of SIF to Virtual Worlds (SIF-VW)

In this section, we extend SIF to virtual world applications. Therefore we first extend the EMS model and then elaborate on the SIF architecture accordingly. As this extension is not of relevance for the case study in Chapter 6, we sketch only briefly the additional features of SIF-VW for completeness reasons.

New developments in *Virtual Reality* (VR) have led to a new dimension in human-computer interaction, in the form of co-habited virtual worlds. In such worlds, human users communicate with other users or artificial agents in a global network setting. Applications of CHVW are, for example, virtual conferences where lifelikeness and interaction of avatars are key issues, but also virtual marketplaces where electronic salesmen agents and customer avatars trade over goods.

Extension of the EMS Model

In order to extend SIF to the requirements of CHVW, we apply three modifications to the underlying EMS model which are also implemented in the SIF architecture.

In general, a natural environment offers several independent media corresponding to different types of sensors and effectors: For instance, agents in nature have *mouth* effectors and *ear* sensors for verbal communication which is transmitted through the *air* which is done independently from, e.g., emitting and detecting *infra-red light* and outpouring *odors* that stimulate a *smelling* sense. For a realistic virtual world simulation, an agent must be equipped with a particular set of sensors and effectors in order to interface to multiple media.

In this case, natural effectors and sensors do not process pure information, but *interpret* the data: they comprise the *view* of an agent. For instance, aural percepts by which an agent is notified normally become more and more noisy with increasing distance from sender to receiver. The degree of noisiness depends on characteristics, e.g., the quality of the agent's ear. For this purpose, we split EMS sensors and effectors into *virtual* and a *logical* parts. The virtual part of a sensor or effector interfaces the engine and serves as a preprocessing function. The logical part is tied to the agent and establishes a connection to its virtual complement.

EMS also matches very well the special role of human users in CHVW. A user interface can be seen as the view of a human in which a 2D/3D-browser serves as a virtual sensor (the display) and at the same time as an effector, e.g., by translating a mouse-click into a command for the user's avatar in the virtual world.

Extension of the SIF Architecture

The additional functionality of EMS requires to extend the SIF architecture. The world server has to process different media concurrently, which is realized by the introduction of several priority queues.

Agent and avatar views are facilities to create and discard particular virtual effectors and sensors on request of the agent. Virtual sensors can be regarded as interpretation functions which convert method calls by the sensor manager into method calls for the agent's logical sensors. Complementary, virtual effectors inter-operate with logical effectors and the medium's action manager.

The world server receives data from the different media and delivers them to avatar user view for visualization and interaction purposes. Actions of the human user, on the other hand, are not object-level actions, but guidelines for their avatars or even commands for controlling the overall simulation. The world server processes these commands and transmits behavior guidelines to the avatars through *meta-sensors.* Agents and avatars also own a *meta-effector* to transmit their appearances, i.e., visualization data, to the meta-medium at startup.

SIF is equipped with two optional user views in 2D and 3D. Both interface the meta-medium via RMI. The 2D version is JAVA$^{\mathrm{TM}}$*Abstract Window Toolkit*-based and is suitable for grid-like worlds. The 3D version runs in standard WWW-browsers using VRML'97 technology.

4.2 An Extension of SIF for Integrated Resource Adaptation (SIFIRA)

In this section, we present the SIFIRA framework, a tool for resource-oriented programming which extends the standard SIF system by built-in functionalities for the construction of resource-adapting agent societies. These features implement the RIL and some RAL schemes of Chapter 3.

SIFIRA is a JAVA$^{\mathrm{TM}}$ library that can be added to the original SIF system. It contains abstract classes for the inheritance of resource allocation, monitor agent and member agent functionalities, pre-defined monitoring and guidelining effectors, sensors, actions and percepts. The package also contains extensions of the action manger and sensor manager to cope with these new features, an expanded script parser that supports the definition of group member and representative relations, and a library of RALs which can be integrated into the functionality of a resource-adapting agent.

A *utility function* is a mapping $u : \mathcal{E} \times S \rightarrow \mathbb{R}$ where $u(e, s)$ denotes the value of the environmental situation e for an agent in state s.

Let $e = (e_1, \ldots, e_i, \ldots, e_n, s) \in \mathcal{E}_1 \times \cdots \times \mathcal{E}_n \times S$ be a universe state, and u a utility function for the agent in question. A dimension $\mathcal{E}_i$ is an *abstract resource* for the agent with respect to u if

- there is a $U \in \mathbb{R}$ with $U > u(e)$,
- there is sequence of actions $a_1, \ldots, a_n \in A^*$ that transforms e into a state g with $u(g) \geq U$, and
- for some other $e'_i \in \mathcal{E}_i$, the state $e' = (e_1, \ldots, e'_i, \ldots, e_n, s)$ cannot be transformed into a state g' with $u(g') \geq U$ by the same sequence of actions.

Figure 4.6: Specification of utility and abstract resource

Formal Characterization of the Behavior of a Resource-adapting Agent

We now extend the formal agent specification of Section 4.1.2 by a resource adaptation functionality. As stated above, we assign utility values to the states of the abstracted, discretized environment. This utility value expresses the *local* utility estimation of the individual agent. For the sake of simplicity, we assume a single-agent (multi-dimensional) environment $\mathcal{E} = (\mathcal{E}_1 \times \cdots \times \mathcal{E}_n)$ for the agent $\alpha = (S, P, A, \phi)$.

Taking abstract resources to be part of the environment, we identify certain sub-sets of sub-states as resources. In accordance with our definition of Section 3.2.5, we model abstract resources as that part of the environment that enables the agent to reach a state of the world that has a higher utility than the current one.

In particular, we are interested in sequences of actions that serve to leave the current universe state e that has a utility lower than a certain threshold $U \in \mathbb{R}$ and to reach a state g with a utility greater than U. We define those dimensions of the current environment $\mathcal{E}_i$ as abstract resources that are essential for the success of these action sequences, i.e, there is an instance e_i of $\mathcal{E}_i$ that is available to the agent and after the action sequence state g will be reached while otherwise (e_i is not available) only a state g' will be reached which has a utility less then U. Figure 4.6 formalizes this scenario which is displayed in Figure 4.7.

As an example, consider an environment in which the agent needs fuel to get from its current geographical position a to some other position. Let $\mathcal{E}_5$ denote the dimension of the environment that describes the fuel supply assigned to the agent. Currently the agent has $e_5 = 5$ fuel units available. The agent considers the current universe state to have a utility of 50. States in which the agent is located at some other geographical position b are assigned to utility 70 and states

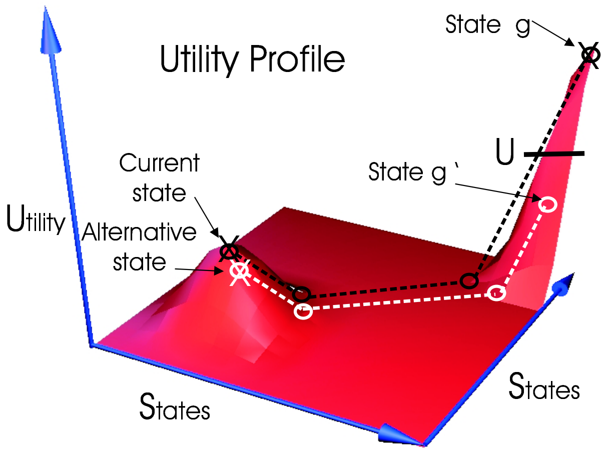


Figure 4.7: Utilities of different states

with some location c are valued to 100. To reach b takes 3 fuel units while to reach c takes 5 units.

According to the previous definition, $\mathcal{E}_5$ is a resource, since the current state can be transformed to a state with a utility greater than a threshold U of, say, 90, while a similar initial state that differs to the current state only by having 4 units of fuel can only lead to a state of utility 70 which is less than U. This of course implies that fuel is sometime a resource for an agent and sometimes it is not, which fits our definition of an abstract resource as a subjectively perceived entity.

4.2.1 Functionalities of Additional Agent Classes

For the design of agent hierarchies that implement the RIL scheme, we need different types of agents: those that perform RALs, those that send directives and those that receive and obey these directives. In a resource-adapting hierarchy, all agents need the first functionality, the top-most agents need the second one and the bottom-most agents need the third one, while intermediate agents need all functionalities.

In the original SIF system, the scenario designer can define new agents by inheritance from the abstract class `Agent`. For SIFIRA, we offer four additional abstract classes `ResourceAdaptingAgent`, `MonitorAgent`, `MemberAgent` and `MonitorMemberAgent`[2], all of which extend the class `Agent` and have additional methods at their disposal. By extending one of these classes, the scenario

[2]If JAVA^{TM} allowed multiple inheritance, we would abstain from this abstract class in which both, monitor agent and member agent functionalities are provided.

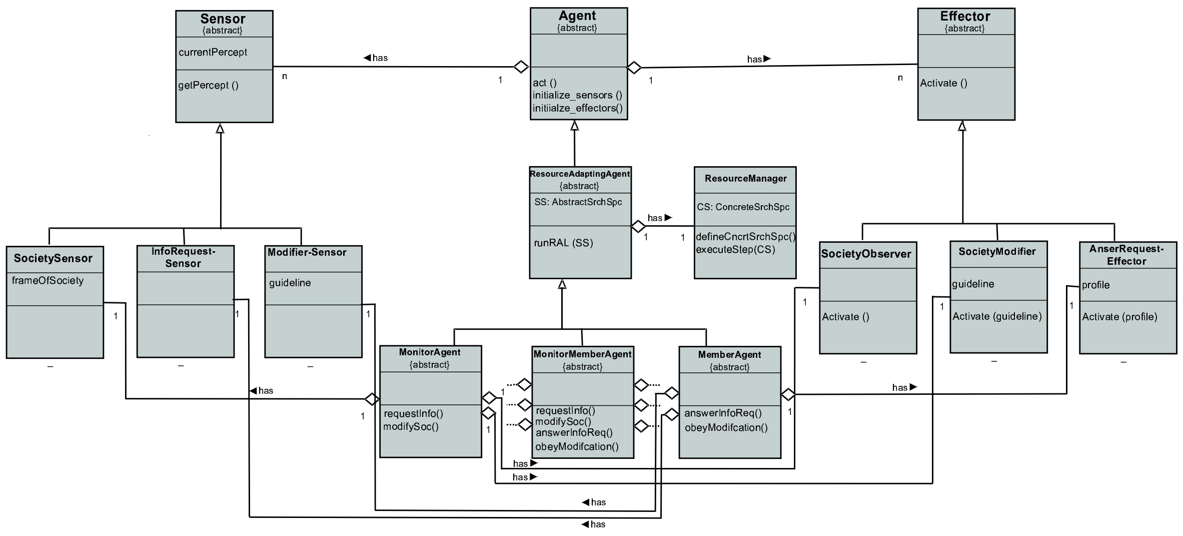


Figure 4.8: Agent hierarchy in SIFIRA

designer can create agents that inherit meta-level resource-allocation functionalities, but that can also perform object-level tasks that have to be defined by the scenario designer. Each time, an agent thread that extends one of these classes receives computational time from the JAVA^{TM} Virtual Machine, first the functionalities of the base class are executed, and then the designer-defined individual functionalities of the agent can be performed. In the following, we simply call an agent that extends the `MonitorAgent` class or `MonitorMemberAgent` class a *monitor agent* and in analogy we use the terms *member agent* and *resource-adapting agent*.

Just like any other agent in the SIF framework, these agents interact with their environment (including other agents) via sensors and effectors: their sensors receive percepts from the world server; their effectors emit actions whose effects on the environment are computed by the world server. In addition to the sensors and effectors that are defined by the scenario designer, these agents have predefined sensors and effectors.

A resource-adapting agent inherits the functionality to run a RAL which is explained in the section on RAL implementation below. The classes `MonitorAgent`, `MemberAgent` and `MonitorMemberAgent` extend the class `ResourceAdaptingAgent` as displayed in Figure 4.8 according to the UML standard [BRJ97]. (For the sake of clarity we have not displayed the aggregations of the class `MonitorMemberAgent` to all subclasses of `Sensor` and `Effector`.)

A monitor agent provides additional functionalities: A *monitoring scheme* of this abstract class contains a time management routine, which triggers the agent to monitor the structure of the agent group it is responsible for (method *requestInfo()*) after a user-defined time interval is passed. This time management routine also contains a time-out mechanism that prevents the monitor agent from running in a deadlock if some of the member agents do no longer reply to one of its requests. Furthermore, *guidelining functionality* enables such an agent to send directives to member agents (method *modifySociety()*).

A member agent inherits the functionality to receive and answer requests on the current status (method *answerInfoReq()*), and also to receive and obey guidelines (method *obeyModification()*).

As in standard SIF, besides the definition of new agents, the system designer has to specify the effect of actions on other agents in the action manger and sensor manager. In SIFIRA, both classes already contain consequences for actions of the monitoring and guidelining scheme. However, the scenario designer can add further features. Similarly, the scenario script parser contains already functionalities to cluster agents to groups and to assign monitor agent roles. This parser can still be extended to additional scenario features. In Chapter 6 we will exemplify this in detail.

4.2.2 Implementation of RALs

The RAL library currently contains the standard Greedy-RAL algorithm and its extensions for the retrieval of heuristics from bottleneck analysis and machine learning.

The class `ResourceAdaptingAgent` allows to incorporate any RAL from the library. Therefore, this class has an instance variable of the abstract type `AbstractSrchSpc` at its disposal, that represents the search space. Furthermore, the method *runRAL(SS)* is provided in order to take one step in the abstract search space.

The class `ResourceManager` is aggregated to `ResourceAdaptingAgent`. This class has an instance variable of the abstract type `ConcreteSrchSpc` at its disposal, where abstract resources, their interpretation, boundaries, step size etc. have to be specified by the scenario designer using the method *defineCncrtSrchSpc()*. The resource manager translates abstract RAL steps of the resource-adapting agent into actions, i.e., the concrete society modifying actions are derived and executed using the method *executeStep(CS)*.

The separation into resource-adapting agent and resource manager follows the information hiding paradigm, also applied to the partition of the world server into domain-independent and domain-dependant parts (action manager and sensor manager): the resource-adapting agent performs a RAL on some abstract domain-independent search space while information on the scenario-dependent resources are stored only in the resource manager, which interprets an abstract move into concrete steps.

4.2.3 Implementation of RIL

Modification of the RIL Scheme

For the persistent use in multi-agent societies, we have to modify the abstract RIL scheme presented in Chapter 3. In a hierarchical approach to find an efficient position in a multi-dimensional search space of configurations we did not specify how to measure the performance of a certain configuration.

We shall now address this issue. An agent of any stage in the society that performs a RAL needs to measure "its" performance (which is the joint performance of a sub-group in the case of a representative agent) A performance metric must be a priori provided by the scenario designer: The performance of a (sub-) system can be measured in many quantitative ways (for instance the number of answered requests to the agent society per time units, see the case study of Chapter 6), as well as qualitative ways, such as for instance the quality of a tour (i.e., its length) in a traveling salesman problem (see the case study of Chapter 7). Due to the unpredictability of the environment, the quality of a solution may vary, even if the agent society has not been modified. For more valid performance values SIFIRA offers facilities to collect a *performance profile* by averaging the performance over a period of time.

In the following, we recall the terms *global profile* and *local profile* from Chapter 3 to denote the profiles of monitor agents or member agents, respectively. Often, a global performance profile is based not only on the overall performance value of a group of agents, but also on parts of the local profiles (for instance, the workload of the member agents). Therefore, SIFIRA does not only offer a *guidelining scheme* for spreading directives from monitor agent to member agents, but also a pre-defined *monitoring scheme* for retrieving information of local profiles of the group members. The use of this scheme is in particular important as the monitor agent cannot be sure that its directives are obeyed completely by the members. Applying the monitoring scheme helps to update the global profile and hence to determine the actual current state in the search space.

The abstract RIL of Chapter 3 performs backtracking in the sense that a new step in a search space of a certain stage is only performed when all agents of the stages below have completed their RALs and have found a locally optimal solution. Employing this scheme for realistic applications is not tractable since it may not always be clear when a "solution" has been found; furthermore due to external environment changes, the search space might have changed or the execution of a RAL of one of the sub-agents simply takes too long. Therfore SIFIRA incorporates a *time management system* which triggers in well-defined time intervals agents to perform a new RAL step, no matter if all lower-ranked agents have succeeded in their optimization. Obviously, higher-ranked agents perform RAL steps much less frequently than lower-ranked agents since their sub-ordinated agents need time to re-configure.

Determination of the next time point to perform a RAL step Hansen and Zilberstein [HZ96] propose the use of *anytime algorithms* for finding an efficient trade-off between solution quality and computation time for certain applications. We can make use of their techniques in the following way: During the run of GRAIL, we are interested in finding for a given stage the optimal time point for collecting a new performance profile and performing a modification in the resource allocation. Until this point is reached, the lower-ranked agents can continously perform their RALs.

To do so, we need an estimation of the solution quality after a certain period of time: $P(q'|q, \Delta t)$ denotes the probability that from a current state in the resource allocation space with quality q a new position of quality q' is found after Δt steps.

However, the usefulness of a state is not only determined by its quality, but also by the duration of the search procedure to achieve this quality. We therefore define $U(q, t)$ as the *utility* of the approach at time t and quality q.

The *marginal utility* of continuing for one single time step is the difference between the expected utility (EU) a time $t + \Delta t$ and the utility at the current time t:

$$EU(t, q) = \sum_{q'} P(q'|q, \Delta t)U(q', t + \Delta t) - U(q, t)$$

This value can be used to decide whether to immediately perform a RAL step ($EU(t, q) < 0$) or to wait another time period Δt and then decide again ($EU(t, q) > 0$). According to [HZ96], this strategy is optimal, if the following holds:

$$\forall t, q, \Delta t, \Delta q > 0 : EU(t, q) < 0 \Rightarrow EU(t + \Delta t, q + \Delta q) < 0$$

A decision rule can hence be found by optimizing the following value function, where d represents the decision to wait another time period Δt and then collect a new performance profile or to perform a step in the own RAL now.

$$V(q, t) = \max_d \begin{cases} U(q, t) & \text{if } d = \text{perform RAL,} \\ \sum_{q'} P(q'|q, \Delta t)V(q', t + \Delta t), & \text{if } d = \text{wait} \end{cases}$$

The resulting strategy π can hence be derived as

$$\pi(q, t) = \arg\max_d \begin{cases} U(q, t) & \text{if } d = \text{perform RAL,} \\ \sum_{q'} P(q'|q, \Delta t)V(q', t + \Delta t), & \text{if } d = \text{wait} \end{cases}$$

The computation of this recursive strategy is computationally expensive, but can be performed *prior* to the run of the system, if the probabilities $P(q'|q, \Delta t)$ can be estimated with significant accuracy. For such a case SIFIRA allows the scenario designer to specify individual delay times in the time management system.

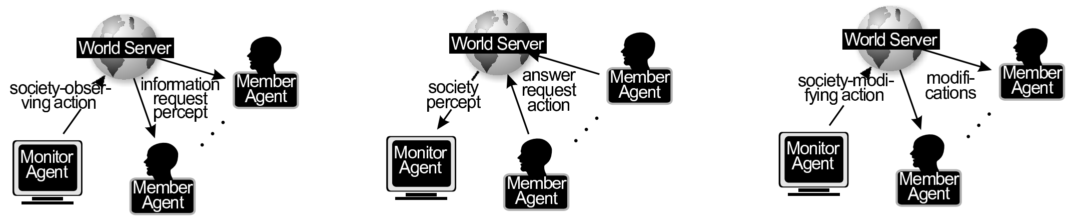


Figure 4.9: Control flow in SIFIRA

Implementation of the RIL Control Scheme

During the run of GRAIL, the time management routine of a monitor agent determines a time point for the next society observation. If this time point is reached, the monitor agent automatically emits a *society-observing* action through its *societyObserver* effector, triggering the extended action manger of the world server to determine the current society status (i.e., number of agents, their structure etc.) and the average group performance since the last RAL step (i.e., system re-configuration). After having emitted this action, the monitor agent performs the actions defined by the scenario designer.

All member agents of the monitor agent's group receive via their *infoRequest sensors* some *information-request percepts* that ask for information on their local profiles. By defining the perception range of these sensors appropriately, it is guaranteed that only agents receive that request which are members of the group the monitor agent represents.

Once a member agent has perceived such an information request, an answering action is performed by its *AnswerRequest effector*, prior to all actions specified by the scenario designer. A member agent emits the requested pieces of information through performing an *AnswerRequest action.* Once all answers have been collected (or timed out) by the extended action manager, the extended sensor manager sends that data as a *Society percept* to the *Society sensor* of the monitor agent.

The monitor agent in turn, automatically uses the collected data to perform the next RAL step (i.e., the modifications in the society configuration). Once the new position in the search space has been determined, the *SocietyModifier* effector of the monitor agent emits a *society-modifying action* (containing instructions how to modify the group) to the world server which commits the modifications (or computes impacts on the society respectively, as it cannot be guaranteed that all actions are committed successfully).

A monitor agent can never assume that all of its directives have been followed to all extend by the group members. In order to get correct information on the

An *agent* α_i is a tuple (S_i, P_i, A_i, ϕ_i) of the set of possible states $S_i = S_i^r \times S_i^{mo} \times S_i^{me}$, the sets of perceptions $P_i = P_i^r \times P_i^{mo} \times P_i^{me}$ and actions $A_i = A_i^r \times A_i^{mo} \times A_i^{me}$, and its agent function $\phi_i : S_i \times P_i \rightarrow S_i \times A_i$.

Let $\mathcal{E} = \mathcal{E}^r \times \mathcal{E}^m$. We define $\Pi^r : \mathcal{E}^r \rightarrow (P_1^r \times \cdots \times P_n^r)$ and $\Pi^m : \mathcal{E}^m \rightarrow (P_1^{mo} \times P_1^{me} \times \cdots \times P_n^{mo} \times P_n^{me})$ as perception functions and $\Delta^r : \mathcal{E}^r \times (A_1^r \times \cdots \times A_n^r) \rightarrow \mathcal{E}^r$ and $\Delta^m : \mathcal{E}^m \times (A_1^{mo} \times A_1^{me} \times \cdots \times A_n^{mo} \times A_n^{me}) \rightarrow \mathcal{E}^m$ as environment functions.

A *monitor-member-relation* between an abstract resource, a monitor agent α_i and a member agent α_j is a relation $MMR \subseteq AR \times \mathcal{A} \times \mathcal{A}$. A *local profile agent function* of member agent α_j is a function $\phi_j^{LP} : (S^r, F) \rightarrow S^{LP}$.

Figure 4.10: Splitting of agent and environment; defining relations between agents

society situation, the monitor agent needs to perform a new *society-observing action* the next time that group has to be controlled.

This modified RIL scheme is repeated until the application is terminated. Figure 4.9 visualizes this control flow. The scheme shows similarities to the contract net protocol. In both cases, a central unit (here the monitor agent) has to solve a problem and announces it to a set of agents (here the members of the group). However, instead of collecting the members' evaluations of the problem based on their local profiles (i.e., their bids for organizing the society according to their local preferences), the monitor agent collects these profiles in order to get a more global profile which enables it to find a better solution.

Formal Specification of the Scheme

Interaction among agents can be defined best by an automaton-oriented description of agents, since the internal states of an agent are hidden and only its actions and perceptions are examined. We use this technique to specify the control flow in SIFIRA.

For our purposes, we distinguish three agent functionalities: the object-level ***r**egular functionalities*, and meta-level ***mo**nitor agent functionalities*, and ***me**mber agent functionalities*. For these different functionalities, we extend the agent formalization of Section 4.1.2 by introducing three-fold partitions for agent states, percepts, actions and transitions which are marked by upper indices r, mo, and me for regular functionalities, monitor agent functionalities and member agent functionalities (first part of Figure 4.10).

We also distinguish between regular object-level facts in the environment description ($\mathcal{E}^r$) and those that are concerned with meta-information ($\mathcal{E}^r$). Here,

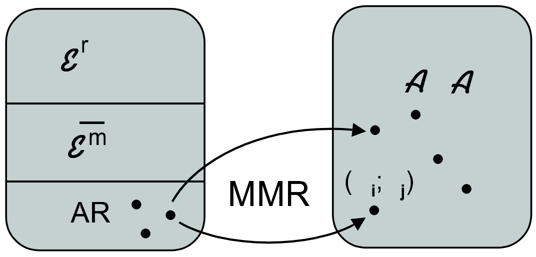


Figure 4.11: Resources specifying relations among agents

we do not distinguish between monitor and member agent issues (second part of Figure 4.10).

The definition of abstract resources induces structural relations of the society, i.e., relations between monitor agents and the members of the groups (see Section 3.2). Therefore, we define a relation *MMR* between the set of abstract resources and the cross product (monitor agent; member agent) of the set of agents (third part of Figure 4.10 and Figure 4.11). As specified above, we consider the set of abstract resources AR as a part of the environment, in particular, part of the meta-level partition $\mathcal{E}^m = \bar{\mathcal{E}^m} \times AR$.[3] For instance, if a monitor agent α_0 controls the resource *fuel* needed by a member of the society α_1, while α_2 controls fuel of α_3, then (fuel, $(\alpha_0, \alpha_1)) \in$ MMR and (fuel, $(\alpha_2, \alpha_3)) \in$ MMR.

A *local profile* of a member agent α_j is represented as a sub-state LP_j of $S_j^{me} = S_j^{\bar{me}} \times LP_j$ of the agent. A local profile is generated by the agent function applied on the regular part of the state of α_j and some external feedback F_j which is part of the regular perception of an agent $P_j^r = \bar{P_j^r} \times F_j$.

The RIL scheme employs a monitoring sequence prior to the actual (re)-assignment of resources in order to build a *global profile.* In our scheme, such a monitoring action induces automatic local profile reporting actions of all member agents in question. These reporting actions are derived from the perception of the request and from the local profiles of the member agents. We enable only those agents to send profile requests that stand in an appropriate monitor-member-relation to the affected agents. A global profile of a monitor agent α_i is represented as a sub-state GP_i of $S_i^{mo} = S_i^{\bar{mo}} \times GP_i$ of the agent. In order to describe a temporal development, in Figure 4.12 we annotate future values of variables with ticks (').

We assume that an agent has to perform a set of tasks. We do not care how such tasks are represented in the concrete agent architecture. For instance, they could be represented as goals in a BDI-oriented architecture or they could be

[3]We denote the complement set of a set by a bar.

An agent α_i *requests a local profile* lp_j from agent α_j if there exist a *society-observing-action* a_i, a feedback f_j, local profile lp_j and a *answer-request-action* a'_j, with

- $(ar, \alpha_i; \alpha_j) \in MMR$ for an appropriate abstract resource ar,
- $e' = \Delta(e, a_1, \ldots a_i, \ldots a_n)$, *(the next state of the environment)*
- $(\bar{p_j^r}', f'_j) = \Pi_j^r(e')$, *(feedback as part of the perception)*
- $lp'_j = \phi_j^{LP}(s_j^{r\prime}, f'_j)$ *(local profile derived from state and feedback)*
- $a'_j = \phi_j^2(s_j^{r\prime}, f'_j)$ *(answer request action derived from state and feedback)*
- $e'' = \Delta(e', a'_1, \ldots a'_j, \ldots a'_n)$, *(the next state of the environment)*
- $(s_i^{r\prime\prime}, s_i^{\bar{m}o\prime\prime}, gp''_i) = \phi_i^1(s'_i, \Pi^i(e''))$ *(the next state of the monitor agent)*

The *resource assignment* of an agent α_i is a function $ra_i : AR \rightarrow T_i$ from the set of abstract resources to the set of tasks of α_i.

Figure 4.12: Specification of information requests and resource assignment

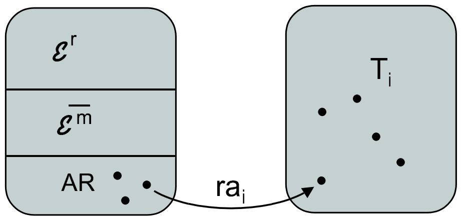


Figure 4.13: A resource assigned to a task

directly implemented into the agent source code. In our framework, tasks T_i are represented in the regular part of the agent state $S_i^r = \bar{S}_i^r \times T_i$ of the agent α_i.

Similar to Jameson [Jam97] (see Section 3.2.3), we define in the second part of Figure 4.12 a *resource assignment function* ra_i ranging from the regular state of an agent α_i to a task that agent has to perform, as a part of the regular part of the state of an agent.[4] Figure 4.13 visualizes the assignment. The previous definition reflects the local perspective of agent α_i on the assignment of a resource to one of its own tasks. Now we treat the case where one agent restricts or enables the access of other agents to a resource. We call this case resource *allocation.*

An agent allocates an abstract resource ar from agent α_j to agent α_k, if prior to that allocation there was an assignment ra_j from ar to one of α_j's tasks and after that allocation there is an assignment ra'_k from ar to one of α_k's tasks. We enable only those agents to shift resources that stand in an appropriate monitor-member-relation to the affected agents. A monitor agent α_i can allocate a resources ar to/from itself, if $(ar, (\alpha_i; \alpha_i)) \in MMR$ (see Figure 4.14).

[4]This of course induces that every agent is aware of its resources.

An agent α_i allocates the resource *ar* from agent α_j to α_k if there exists an *allocation-action* a_i, with

- $(ar, \alpha_i; \alpha_j) \in \mathit{MMR}$ and $(ar, \alpha_i; \alpha_k) \in \mathit{MMR}$
- $e' = \Delta(e, a_1, \ldots a_i, \ldots a_n)$ *(the next state of the environment)*
- $s'_j = \phi^1_j(s_j, \Pi^j(e'))$ *(the next state of member agent α_j)*
- $s'_k = \phi^1_k(s_k, \Pi^k(e'))$ *(the next state of member agent α_k)*
- $ra_j(ar) = t_j$ for some $t_j \in T_j$ *(α_j holds currently resource)*
- $\not\exists \alpha_l : ra_l(ar) = t_l$ for some task t_l of α_l *(no other agent cur. holds res.)*
- $ra'_k(ar) = t'_k$ for some $t'_k \in T'_k$ *(α_k then holds resource)*
- $\not\exists \alpha_l : ra'_l(ar) = t'_l$ for some task t'_l of α_l *(no other agent then holds res.)*

Figure 4.14: Specification of the resource allocation scheme

Bottom Line

In this chapter we have presented the SIF system, a toolkit for agent-based simulation, and its extension SIFIRA that supports conveniently the setup of resource-adapting agent societies.

This general framework is the basis for all resource allocation mechanisms implemented and presented in this work. This framework does neither specify the actual resource to be modified, nor the amount to be shifted, nor the donator and recipient of resources, nor the time intervals between monitoring and allocation actions. These items depend on the strategy selected and on the application domain. We will instantiate this framework to each of the case studies in Chapters 6 and 7.

Chapter 5

Structural Adaptation in Holonic Multi-Agent Systems

In Chapter 3 we have introduced a generic hierarchical self-adaptation scheme which bases on the distribution of abstract resources. We have identified critical properties of the application, such as openness, dynamics, unpredictability, complexity and benevolence. We have also derived GRAIL, an optimization approach that is suitable for domains of such properties. Now we further restrict application characteristics, mainly in terms of *autonomy* of the agent, *inherent structures* and *decomposability* of the domain. In such *holonic* cases, we can refine the resource-adaptation scheme.

In this chapter, we identify traits of domains that are suitable for the holonic paradigm and derive an agent definition and use it as a basis for a definition of holonic agents; furthermore, we propose a general implementation framework for holonic agent systems and apply GRAIL in the central issue of resource management for building and maintaining holonic systems. We also present four prototypical holonic domains and corresponding holonic solutions.

5.1 Background

Many distributed problems exhibit a recursive structure: an agent that works on a macro-level of a decomposable problem may have a similar structure as the agents for the micro-level sub-problems, thus the agent society should be structured recursively. More generally, an agent that appears as a single entity to the outside world may in fact be composed of many sub-agents and conversely, many sub-agents may join into the coherent structure of a super-agent and thus act as single entity. We call agents consisting of sub-agents with the same inherent structure *holonic agents.*

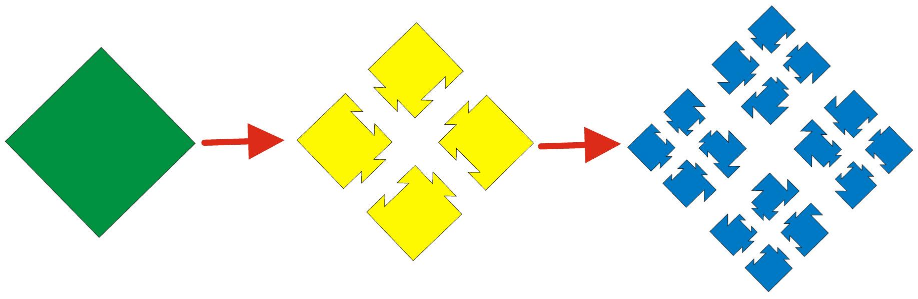


Figure 5.1: Decomposition of a holon

5.1.1 The Holonic Principle

The term *holon*, a combination of the Greek *holos* (whole) and the suffix *-on* (part), was originally introduced in 1967 by the Hungarian philosopher Arthur Koestler [Koe67] in order to name recursive and *self-similar* structures in biological and sociological entities. According to Koestler a holon is a biological or sociological structure that consists of further holons that function similarly. No natural structure is either "whole" or "part" in an absolute sense, instead every holon is a composition of subordinate parts as well as part of a larger entity (see Figure 5.1). For example, a human individual is on the one hand a composition of organs consisting of cells that can be further decomposed, and on the other hand the person may be part of a group which in turn is part of the human society.

Koestler's concept has been applied in the area of flexible manufacturing systems (see for instance [Dee94]), where the advantages of Koestler's holonic societies (or *holarchies*), namely stability, adaptability, flexibility and efficiency have motivated a similar design for sufficiently redundant manufacturing processes.[1]

Holons in *holonic manufacturing systems (HMS)* are characterized by *holonic attributes*, namely autonomy and cooperativeness. In this context, autonomy means the ability to create and control the execution of plans and strategies. Cooperativeness allows for joint planning and coordination of joint plan execution. In a holonic manufacturing system, holons consist of an information processing part and sometimes a physical part, which is responsible for transforming, transporting, storing and validating information as well as physical objects. Manufacturing holons can be build recursively from of other holons.

For the characterization of the term *holonic multi-agent system* we use the terms *holon* and *holonic agent* synonymously. By *super-holon* we denote a composition of subordinate agents, which we shall call *sub-holons* or *sub-agents*. As these sub-holons may be further decomposable into sub-sub-holons we shall use the term *immediate sub-holons* to distinguish it from its transitive closure.

[1]For details see `http://hms.ifw.uni-hannover.de/public/hms_tech.html`.

5.1.2 Models of Autonomy

Autonomy is not only a vital property of traditional agent systems, but also a critical issue for holonic systems where agents joining a super-agent have to surrender (some of) their autonomy. In this section, we present a collection of existing definitions for autonomy starting with popular definitions and then looking at more research based technical definitions and implementations. We use these definitions as a starting point for our characterization of agent autonomy.

Basic Definitions

Autonomy literally means “self-governing”, and is derived from the Greek words *autos* (self) and *nomos* (law, rule). Generally, it has implications of independence and seperatedness. The American Heritage Dictionary of the English Language describes the term *autonomous* as *1. not being controlled by others or by outside forces; independent: 2. independent in mind or judgment; self-directed.*

Usually, autonomy has become is associated with the self-regulation of smaller systems, such as agents. Russell and Norwig [RN95] define an agent's behavior *autonomous to the extent that its behavior is determined by its own experience.*

According to Castelfranchi [Cas95], agent autonomy means that agents control their actions and internal states to enable them to operate *without the direct intervention of humans or others.* He defines autonomy as *the amount of separation between external influences and an agent's goals.*

Falcone and Castelfranchi [FC99] treat autonomy equal to *self-sufficiency*, not being dependent on others for private goals: the less dependent agent α_1 is on agent α_2 regarding the resources for that task, the more autonomous α_1 is of α_2 regarding that task.

Luck and d'Inverno [Ld95] define an autonomous agent to *anything that has motivations*, where motivation is defined to be any desire or preference that can lead to the generation and adoption of goals.

Operationalizations

Barber and Martin [BM99] interpret autonomy with respect to *an agent's degree of freedom from intervention of others.* They distinguish between three discrete categories: A *command-driven* agent does not plan and must obey orders given by another (master) agent. A *consensus* agent works as a team member, sharing planning decisions equally with other agents while a *locally autonomous/master* agent plans alone and may or may not give orders to other agents.

Cesta et al. [CDC99] examine agent autonomy in terms of human interaction. They distinguish between *low-interactive agents* that perform complex tasks that require very few interaction with human users, and *highly-interactive agents* that carry out expert tasks along with the users. For Cesta et al., agents of the former type are more autonomous than agents of the latter type. A degree of autonomy could then be measured by counting the number of user interactions.

Hexmoor et al. [HLT99] see autonomy as *self-governance over output*. They classify three types of autonomous agents: the *fully autonomous agent* makes decisions and reacts to the environment without much input from an outside source. A *boss* is an agent that has control over other agents as well as over itself. Finally, *cooperative* agents communicate with others and as a whole, they come up with a course of actions.

Huhns and Singh [HS98] define a collection of different autonomy concepts: they distinguish between *absolute autonomy* (an agent completely ignores other agents and does whatever it wants to do), *design autonomy* (describing the freedom an agent designer has for the setup of an agent), and *execution autonomy* (capturing how much freedom an agent has in making decisions while executing a plan).

Huber [Hub99] considers autonomy with regard to BDI theory by speaking of *belief, goal and intension autonomy* where an agent is autonomous in terms of its beliefs (goals, intentions, respectively) if it is free to modify its beliefs (goals, intentions) without the inference of others.

Dorais et al. [DBK$^+$98] characterize autonomy of a system to depend on the *complexity of commands* to control the system, the *number of autonomously controlled subsystems*, the *circumstances that allow the system to override user commands*, and the *duration of the autonomous operation*.

Our Perspective

For the purpose of partially restricting autonomy in holonic agent societies, the basic definitions provided in [Cas95] and [FC99] reflect also our perspective of autonomy. We do not rely on a certain agent architecture, and hence we do not base on a special underlying theory, such as the BDI theory. For our purpose, we decompose autonomy into three aspects:

- **State autonomy:** An agent's state is determined only by its previous states and its perception, as supported by e.g., the SIF framework (see Section 4.1).
- **Action autonomy:** Like in Castelfranchi's definition, the action of an agent is determined solely by its current state.

- **Computational autonomy:** The agent either has computational means of its own or is supplied with computation time (and space) in a fair manner.

5.2 Holonic Domains

In this section we examine how domains can be characterized and delimited from those that are better suited for traditional multi-agent system. Obviously it is not possible to give an absolute classification: the boundaries between domains that are suitable for holonic agents and those that are not, are blurred. So we will present a collection of criteria as a *guide* for the classification.

Actions of different scopes: Holonic systems are well suited for domains with actions of different granularity. Macro-level actions are carried out by the holon's super-agent and decomposed onto the sub-holons. This could be realized in a traditional MAS too; however, the relationship between the individual agents and the group would have to be represented explicitly; a holonic system provides a priori the relevant features.

Hierarchical structure: An application domain that exhibits a hierarchical structure is usually an excellent candidate for a holonic system, since hierarchies of sub-holons can be modeled canonically. The structure of the domain induces abstraction levels, which can be modeled naturally in a holonic system.

Decomposability: One of the main pre-requisites for a traditional agent-based system is a decentralized or decomposable problem setting, where each agent is assigned to one of the sub-problems. Pro-activeness and autonomy of the agents are the major features.

Often however, problems are neither completely decomposable nor completely non-decomposable; in many *hybrid* cases, some aspects of the problem can be decomposed, while others cannot. Holonic agents are structured hierarchically, they can easily realize actions of different granularity, hence holonic agent systems can naturally deal with problems of that type. Still, completely decomposable as well as completely non-decomposable problems can be attacked with holonic agents leading to overheads imposed by components not needed for these special cases.

Communication: If the overall problem is decomposed into sub-problems that are not partitions of the original one, but some overlapping exists in the sense that logical interdependencies occur, communication between the problem solvers is needed. A domain often induces an unsymmetrical communication behavior between problem solvers in the sense that each unit does not communicate to all other units equally often, i.e., patterns in the communication behavior can be observed. These patterns indicate possible structures for holonic agents: Holons provide facilities for efficient communication inside the holon (*intra-holonic* communication), which occurs more frequently than communication among different holons (*inter-holonic* communication).

Cooperativeness: In Section 3.4 we have already distinguished between cooperative and non-cooperative settings. A cooperative setting does not constrain the use of holons in any way. However, in non-cooperative settings (e.g., virtual market places), things are different. If there is *no* cooperation among agents in the domain, the use of holonic agents is not very reasonable. If there are cooperative elements in the domain, holonic agents can be used to model the cooperative sub-domain.

Situatedness and real time requirement: For many applications real-time behavior is a vital issue: The problem solver has to find a solution within limited computation time. As for some traditional agent architectures, bounded rationality for all members of sub-holons is a way to cope with these challenges by explicitly reason about time and other resources in order to find the best possible action within a given resource allocation.

Summary

The most important requirements for a holonic domain are structure and cooperation: Similarly to structured agent groups as discussed in Chapter 3, centralistic aspects of a domain can also be modeled. However, in contrast to domains of benevolent agent societies described in Chapter 3, the domain should have a holonic structure, i.e., it should be recursively decomposable. The holonic system can then be mapped canonically onto that structure. Furthermore, there must be sufficient cooperative elements between the distinguished problem solvers.

5.3 Holonic Agents

In this section, we first determine relevant properties for regular agents which we then extend for a definition of a holonic agent with respect to the characteristics of holonic domains of the previous section. Finally, we give a formal characterization of a holonic agent and its perspective on the environment.

5.3.1 Characteristics of Traditional Agents

In the DAI literature exists a wide range of agent definitions. In Section 2.1 we have already compiled definitions that are relevant for this context. For our purpose, we focus on the following agent characteristics:

Autonomy: Autonomy is one of the vital requirements for agents, while an appropriate restriction is one of the crucial traits of holonic agents. Therefore we have discussed this issue already in Section 5.1.2.

Goal-directed behavior: An agent has explicitly or implicitly represented goals and desires, where desires are defined as in BDI theory.

Action: Due to Wooldridge and Jennings [WJ95], an agent is able to behave *reactively* to changes in the environment and/or it can behave *pro-actively* in order to reach some goal. For the scope of this work, we do not have to go into this distinction but we do require the general capability to act.

Belief: Agents have implicit and/or explicit representations of their environment.

Bounded rationality: In analogy to Russell and Wefald [RW91], we require a rational agent to behave optimally with respect to its limited resources and its goals.

Communication: Similar to Wooldridge and Jennings' requirement of social ability in [WJ95], the agents have to share a communication language.

5.3.2 Characteristics of Holonic Agents

We now extend these requirements to a definition of holonic agents. According to Koestler's original framework, arbitrary structures can be viewed as holons; the sub-structures do not necessarily have to be of the same kind. In contrast to his picture we restrict all entities to agents as defined above, and furthermore, we require that sub-holons always have the same structure as the super-holon.

Autonomy: Several agents that form a holon, act as a single entity. The holon interacts with the environment as an autonomous agent in the sense of the criteria presented in Section 5.1.2 (state autonomy, action autonomy and computational autonomy). By joining a holon, agents accept some restrictions to their autonomy: they commit to the goals of the holon and they accept restrictions of their abilities to act and to communicate. Still, they may keep their autonomy to some extend; in particular, they are free to leave the holon or enter a second holon.

Common goal-directed behavior: Sub-agents of a holon still pursue their private goals and by doing so, they have to pursue at least one common goal of the super-holon which may be represented explicitly or implicitly.[2] Hence, the super-holon's overall goals are consistent with the goals of the sub-agents. We do not require that the super-holon's goals are also the goals of its sub-agents, but the goals of the super-holon and those of its sub-agents must not contradict. Consequently, an agent can only be a member of several holons with conflicting goals if the agent is indifferent to these goals. This requirement corresponds well to the cooperation feature of an HMS holon.

Increased group capabilities: An agent's capabilities to act are extended in the holonic case at the group stage to *Macro-level* actions which are composed of the actions of the sub-holons. Hence, a super-holon may have actions at its disposal that none of its sub-agents can perform on its own.

Belief: The requirements for an agent's belief remain unchanged: holons have some representation of their environment, i.e., they hold beliefs about their surroundings. This belief might be represented explicitly within the super-holon, or it may be distributed and implicitly provided by the local belief of the individual sub-holons. Inconsistency between the holon's beliefs and the beliefs of some of its members is of course an issue.

Bounded rationality: A holon has to control its resources in order to exhibit a bounded rational behavior. Resource management of the sub-holons is monitored by the super-holon, which distributes guidelines to its sub-holons for local resource management. This is an essential issue of our holon definition and will be discussed further in Section 5.4.2.

[2]For example, BDI architectures provide an explicit representation of goals. Implicit goals can be ascribed to any agent that exhibits some kind of pro-activeness.

Communication: The ability to communicate is an essential part of an agent's autonomy. We distinguish between communication inside the super-holon and communication between super-holons: In our framework, we regard that the right to communicate with other agents is an exclusive macro-level abstract resource of the holon and not of its sub-holons. This right corresponds to a communication channel between two equally ranked holons. Such channels are managed solely by the super-holon.

Internal communication among the sub-holons is of course allowed, but is controlled by the super-holon. Since problem solving inside a holon is cooperative, internal communication load can be high, hence, an efficient data structure for internal communication should be provided.

5.3.3 The Perspective of Holonic Agents on their Environment

We show that from a theoretical perspective, an agent's view on an environment containing multiple agents can be isomorphically mapped onto an environment of that agent in which only this agent is represented explicitly, while the others are perceived to be integrated into the environment. Furthermore, we show how to construct for a multi-agent environment that contains a holonic agent an another one where (from the holon's view) the holon is split into a group of agents (*holonic decomposition*). Vice versa, we speak of a *holonic composition* if a holon is composed from a group of holonic agents.

This theoretical perspective covers only an agent's *local perspective* of the environment and society. It does not consider *internal* configuration mechanisms inside a holon such as resource allocation, etc. As an extension of the object-oriented paradigm, the holonic paradigm hides these mechanisms to the outside world. Therefore the following scheme is not intended to express the merge procedure in a *globally* adequate way.

For the following definition, recall the agent and multi-agent environment definitions of Section 4.1 where an *agent* α_i has been defined as a tuple (S_i, P_i, A_i, ϕ_i) of the set of possible states S_i, the sets of perceptions P_i and actions A_i, and its agent function $\phi_i : S_i \times P_i \to S_i \times A_i$. A *multi-agent environment* has been defined as a tuple $(\mathcal{A}, \mathcal{E}, \Pi, \Delta)$ of the set of all agents $\mathcal{A}$, the set of environmental states $\mathcal{E}$, a percept function Π and an environment function Δ.

Integrating the state of the environment and agents to universe states enables an agent to observe the same multi-agent setting from different perspectives: It can represent any other entity in the universe explicitly as an agent or implicitly as a part of the environment. The isomorphy specification of Figure 5.2 allows to perceive a multi-agent environment containing several agents to the single agent

Two multi-agent environments $(\mathcal{A}, \mathcal{E}, \Pi, \Delta)$ and $(\mathcal{A}', \mathcal{E}', \Pi', \Delta')$ are ***isomorphic*** if there exists a bijection function $\Psi : \mathcal{E} \times S_1 \times \cdots \times S_n \to \mathcal{E}' \times S'_1 \times \cdots \times S'_m$ such that for all $(e, s_1, \ldots, s_n) \in \mathcal{E} \times S_1 \times \cdots \times S_n$ holds: $\bar{\Delta}'(\Psi(e, s_1, \ldots, s_n)) = \Psi(\bar{\Delta}(e, s_1, \ldots, s_n))$

Figure 5.2: Specification of an isomorphy relation

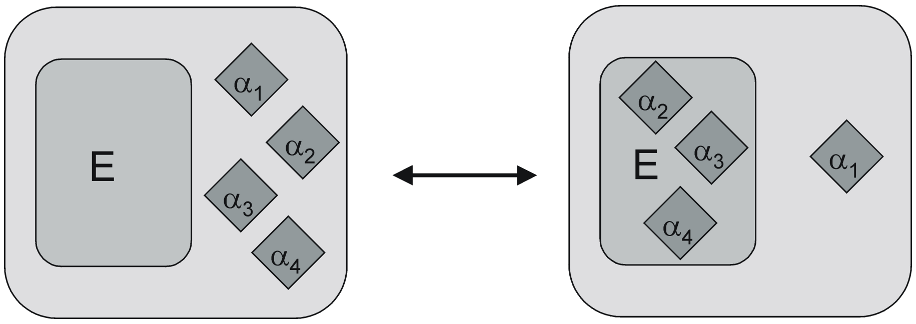


Figure 5.3: (De-)composition of a multi-agent environment to a single-agent environment

case by representing all agents but one as entities of the environment. Figure 5.3 shows the intuition behind this construction, which is explicitly stated in the following corollary. (A proof can be found in Appendix A.)

Corollary 5.3.1 *If* $E = (\{\alpha_1, \ldots, \alpha_n\}, \mathcal{E}, \Pi, \Delta)$ *is a multi-agent environment then for each* $i \leq n$ *there exists an isomorphic multi-agent environment* $(\{\alpha_i\}, \mathcal{E}', \Pi', \Delta')$.

Notice that for any multi-agent environment there exist two special cases: first, an environment without any agents where all state transitions are encoded into the environment function Δ and second, an environment containing only one constant environmental state and one agent where all state transition is encoded into the agent function ϕ. We now introduce the notions of a holonic merge (Figure 5.4) and decomposition and show how to merge a set of agents into a holon, as shown in Figure 5.5.

Corollary 5.3.2 *For every multi-agent environment* $(\{\alpha_1, \ldots, \alpha_n\}, \mathcal{E}, \Pi, \Delta)$ *and* $k \leq n$ *we can construct an isomorphic multi-agent environment* $(\{\alpha', \alpha_{k+1}, \ldots, \alpha_n\}, \mathcal{E}, \Pi', \Delta')$, *where* α' *is a holonic merge of* $\alpha_1, \ldots, \alpha_k$ *in the sense of Figure 5.4.*

Corollary 5.3.1 and Corollary 5.3.2, (which is also proven in Appendix A) enable a view of a collection of agents as one super-holon and to reduce the state transition of the super-holon to the single-agent case.

Consider the two isomorphic multi-agent environment $(\{\alpha_1, \ldots, \alpha_n\}, \mathcal{E}, \Pi, \Delta)$ and $(\{\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i,1}, \ldots, \alpha_{i,m}, \alpha_{i+1}, \ldots, \alpha_n\}, \mathcal{E}, \Pi', \Delta')$. We call α_i the *holonic merge* of $(\alpha_{i,1}, \ldots, \alpha_{i,m})$.

Figure 5.4: Specification of a holonic merge

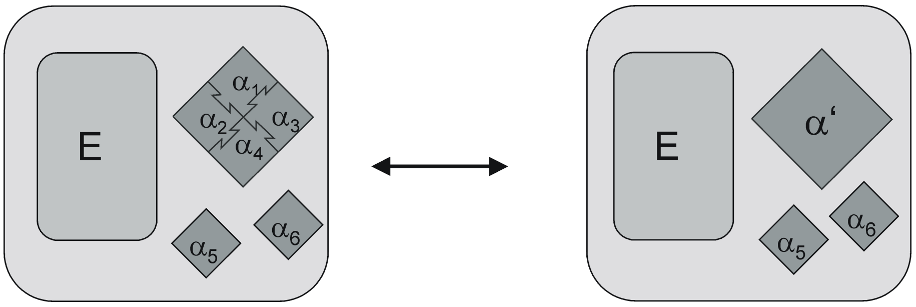


Figure 5.5: (De-)composition of several agents to a holon

5.4 A Scheme for Holon-Oriented Programming

Now that we have distinguished between regular and holonic agents, we provide a basis for an actual implementation of a holonic multi-agent system. A holonic agent of a well-defined software architecture shall be able to join several other holonic agents to form a super-holon; this group of agents then can act as if it were a single holonic agent with the same software architecture.

The nature of the merge of several separate entities into one entity is the subject of this section, where we first present and discuss several approaches how to realize holonic structures; second, we show how GRAIL can be used to implement the dynamic holon (re-)formation procedure. This framework can used for a scheme for *holon-oriented programming* which can be seen as an extension of Shoham's agent-oriented programming paradigm.

5.4.1 Implementation of Holonic Structures

We now present a generic framework for the implementation of holonic agent systems, where the holonic structure of the agent society is explicitly represented. First we examine general possibilities for modeling holonic structures and evaluate whether they are suitable for the design of a holonic system. Major emphasis lies on the aspect of autonomy. The following approaches differ in the degree of autonomy the sub-holons have and cover the spectrum from full sub-holon autonomy to a complete lack of autonomy.

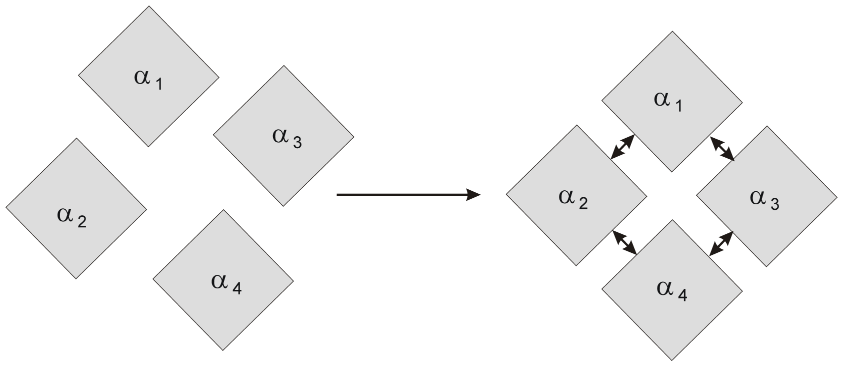


Figure 5.6: A holon as a federation of agents

A holon as a federation of autonomous agents: At one end of the spectrum is a model which assumes that the sub-holons are fully autonomous agents with their predefined architectures and the super-holon is just a new conceptual instantiation of the same generic agent architecture. Figure 5.6 displays this constellation.

In this case, no agent has to give up its autonomy, and the super-holon is realized exclusively through cooperation among the sub-holons. The most transparent way of cooperation for this way is an *explicit coordination by commitment* via communication, i.e., agents negotiate over joint plans, task distribution or resource allocation. If commitments cannot be established through communication, *implicit coordination* can be achieved in two ways: either, the holons are designed such that a goal directed common behavior emerges from the behavior of the sub-agents, or some sub-holons are able to represent goals and intentions of others and to reason about them; thus, they coordinate their actions without or at least with only little communication.

The representation of a holon as a group of autonomous agents is in a sense just another way of looking at a traditional multi-agent system. The holon entity itself is not represented explicitly. In this case, holonic structures are only a design aid for structured agent-oriented programming.

Several agents merge into one: The other extreme of the design spectrum requires to terminate the participating sub-agents and to create a new agent as the union of the sub-agents with capabilities that subsume the functionalities of the sub-agents (Figure 5.7). In this case the merging agents completely give up their autonomy and their existence but they may be re-invoked into an existence of their own when the super-holon is terminated.

The realization of this approach assumes procedures for splitting and merging holons that lead to the creation of new agents. For agents of the same kind with

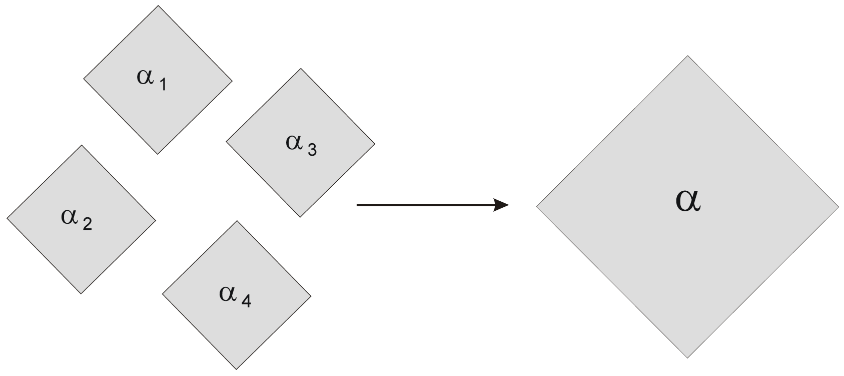


Figure 5.7: Several agents merge into one

an explicit representation of goals and beliefs (e.g., BDI agents) merging may be achieved by creating an agent with the union of the sub-agents' beliefs and goals provided a guaranteed consistency. Especially for a heterogeneous group of agents merging can be intractable and in either case may not be very desirable. A universally applicable merging procedure is hence not available. According to this model, agents cannot participate in more than one holon, unless they are copied.

A holon as a moderated group: The two solutions above may be useful only under very specific circumstances. We propose a hybrid approach that subsumes the extreme architectures above: agents give up only part of their autonomy to the super-holon which could be achieved by designation of one agent as a representative agent or *head* of the holon. Figure 5.8 visualizes this approach.

This head represents the super-holon to the outside world, i.e., it provides the interface to the rest of the agent society. Its competence may range from purely administrative tasks to the authority to emit directives to the other sub-holons. Furthermore, the head has the authority to plan and negotiate for the holon on the basis of its sub-agents' plans and goals, and even to remove some sub-holons or to incorporate new sub-holons. Additionally, it also serves as the interface and contact of the holon to the rest of the agent society.

There are at least two methods to determine the head. Either, a new agent is created for the lifetime of the holon, or one of the members of the holon takes the role of the head and gains the additional functionality. In the second case either one member of the holon is a priori pre-destinated for the leadership or an election procedure is needed to promote one of the agents to leadership. In this model, the autonomy of the member agents is partially traded for control by the head. Depending on the application domain, the competence of the representative agent may vary: the resulting structure can range from a loosely moderated group

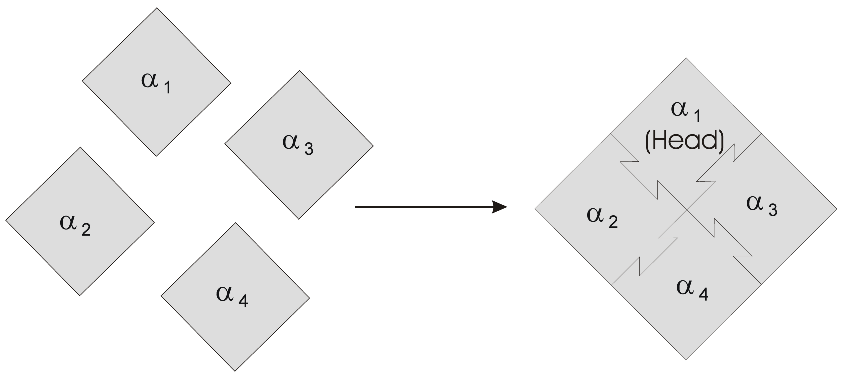


Figure 5.8: A holon as a moderated group

to a hierarchical structure. However, the members of the super-holon are always represented as agents, and, hence, we do not loose the capability to solve problems in a distributed fashion.

Conclusion

Considering the strengths and limitations of these three approaches, we prefer the hybrid one: It allows for an explicit modeling of holons, a flexible formation of holonic groups, and a scalable degree of autonomy of the participating agents. The most challenging problem raising from this definition is the control of the individual and overall computation of the holonic multi-agent system. We propose that control is established by resource bounded computation, i.e., computational resources are allocated within the holonic structures. This will be addressed in the following section.

5.4.2 Resource Allocation in Holonic Systems

The holonic system is given a certain amount of computational resources and the computation within that system is determined by the way these resources are distributed onto the sub-holons. Hence, the whole problem of distributed computation boils down to the problem of how the resources of the super-holon are distributed to its immediate sub-holons. Since a holonic agent system is a special case of a resource-adapting agent society, we can directly employ the approach for building up a hierarchical resource-adapting agent society, and the GRAIL approach of Chapter 3. The RIL can be mapped canonically onto the holon's internal structure, while super- and sub-holons run RALs for their resource management.

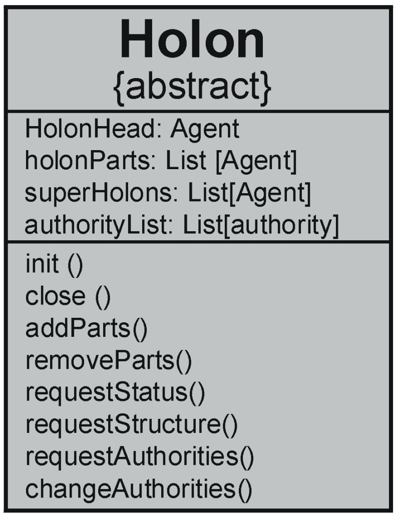


Figure 5.9: The holon object

All RAL mechanisms presented in Section 3.4 are based on the distribution of abstract resources (i.e., parts of the environment that can enhance, or by its absence, constrain the agent's action). From the point of view of the super-holon, the capabilities of its sub-holons can be treated as abstract resources as well. Resources allocated to a super-holon are re-distributed at a finer granularity to its sub-holons, and this allocation can be viewed as a *guideline* of the computation for the lower ones.

5.4.3 INTERRAP as a Basis for Holon-Oriented Programming

In our framework, the INTERRAP architecture (see Section 2.1) is the basis for every member agent of a holon: every holon and sub-holon consists of a private cooperative planning layer, local planning layer, behavior-based layer, world interface, and knowledge base.

Holon Management

INTERRAP is well suited as a basic architecture, since the cooperative planning layer already provides facilities that are extended for holon management.

Each holon is represented by one *holon object* (see Figure 5.9) that is maintained by the cooperative planning layer of the holon's head. The functionality of the holon object is to store the structure of the holon in *reference-lists* with links to the holon's parts and head. Furthermore, the head administers a list of *authorities* which maintains the access rights to the methods of the holon object, and, optionally the rights to use communication channels. The holon object also stores *incompatibility lists* that maintain information about the holon's parts and other agents, for example, the ability of two agents to merge or not to merge to the same holon. The holon object additionally provides a number of methods for the incorporation, removal or modification of sub-holons. These methods are exclusively accessible to the holon's head.

The sub-holons may request information about the holon's structure, and about their own status and authorities from the holon object. Every agent maintains the references to the holon objects of the holons in which it participates and has methods for its incorporation into a holon and its removal from a holon. The incorporation has to be acknowledged by the agent that is to be integrated while the removal needs not to be acknowledged.

Extensions of the Cooperative Planning Layer

The holon object is maintained in the extended cooperative planning layer of the holon's head. Furthermore, cooperative planning layers of all holon members are extended: The holonic structure of the agent society is represented as a directed graph of pointers that is maintained in a distributed fashion by the cooperative planning layer of the member agents.

The extended cooperative planning layer provides functionalities for communication, negotiation and the administration of holonic structures. We distinguish between *inter-holonic* and *intra-holonic* communication: The former type is treated by INTERRAP just like regular inter-agent communication: Communication between agents that are not part of the same holon is organized via communication protocols according to *KQML (Knowledge Query and Manipulation Language)* [FF94].

Efficient intra-holonic communication is realized through method invoking. Additionally, shared logic variables between the sub-holons are introduced. Conceptually, this amounts to overlapping internal states of the agents, which is an intended violation of the autonomy requirement: The use of a shared memory and method invocation can be seen as a partial and reversible merge of the agents involved. If the overlapping is not total, which is usually the case, agents can participate and communicate in more than one holon.

5.5 Four Sample Holonic Domains and Holonic Solutions

We now discuss the suitability of the holonic scheme for a variety of applications. The following four example domains serve for an evaluation of the postulated criteria for a holonic approach. For each of these examples we shall first discuss the domain and then present a holonic solution which bases on the extended INTERRAP architecture described in the previous section.

5.5.1 Transportation Scheduling as a Multi-Agent Domain

The Domain

The *fleet scheduling problem* is a two-staged planning problem: (1) Transportation tasks that express customer requirements have to be assigned to the vehicles of a shipping company. (2) Vehicle tour plans for the assigned tasks have to be generated. Both sub-problems are known to be NP-hard and even constrain each other leading to a further increase in complexity.

The haulage company has a limited number of *transportation units* like drivers, trucks, trailers or tractors at its disposal that must be combined to appropriate means of transportation, i.e., vehicles. Transportation tasks are planned and executed with a limited amount of transportation resources.

The transportation units are not uniform but differ in many ways: The working time of a driver has legal constraints and also the type of cargo he is allowed to transport depends on his legal status. The trucks can be classified into trucks with or without loading space and in truck tractors. The type of the loading space constrains the type of cargo that can be transported, etc. This domain meets the characteristics of a holonic domain as follows:

- **Actions of different scopes:** Object-level actions to be performed in this domain are the execution of transportation tasks. Actions can however be defined at different levels of abstraction: On the most general level it is specified which transportation task a vehicle has to perform. These actions are recursively decomposed into actions of lesser abstraction, such as loading, driving or vehicle maintenance, which again consist of sub-actions such as docking to a terminal, using traffic information systems, refueling, etc.

- **Hierarchical structure:** Transportation units must be combined to form vehicles.

- **Decomposability:** The fleet scheduling problem can be naturally divided into the subproblems of assigning a set of tasks to the vehicles and secondly of route planning for the vehicle fleet.
- **Communication:** Coordination among units that form a common vehicle requires a high amount of communication; cooperation among units of the same company that do not participate in the same vehicle needs less; units belonging to different companies do not communicate in this scenario.
- **Cooperativeness:** The setting is cooperative within a company and competitive between companies.
- **Situatedness and real time requirement:** Although in general there is plenty of time for tour planning (since the planning of a tour is done much faster than its execution), some situations require a fast and real-time answer, e.g., in case of re-planning during execution time or when urgent orders are coming in and have to be scheduled immediately.

The Solution

A canonical way to implement an agent-based solution for this domain is to model vehicles as autonomous agents that compute local plans from which the global solution is derived (see for instance [FMP96, Fal95]). The holon-based system TeleTruck [BFV98a, BFV98b] refines this scheme: In its holonic modeling the basic transportation units (trucks, trailers, drivers, chassis, and containers) are modeled as *component agents* of the InteRRaP architecture.

In order to apply GRAIL to this system, we identify four abstract resources concerning the basic transportation units, all of which are necessary to actually execute a transportation task. In our framework, abstract resources implement dimensions of the search space of a RAL; therefore we present for each resource the range of the dimension and its type (i.e., whether the dimension is continuous or discrete, nominal or cardinal).

- The daily *driving time* of the driver, ranging on a cardinal and quasi-continuous scale from 0 to 9h and arbitrarily small step size,
- The *loading space* (in load meters[3]) of trucks and containers on a cardinal and discrete scale ranging from 0 to 700 cm (for a regular container) or 1400 cm (for a double-sized container) on a step size of 1 cm,

[3]The reduction from a two- or three-dimensional representation to a one-dimensional representation is often used in logistics and helps in this case to cut down complexity.

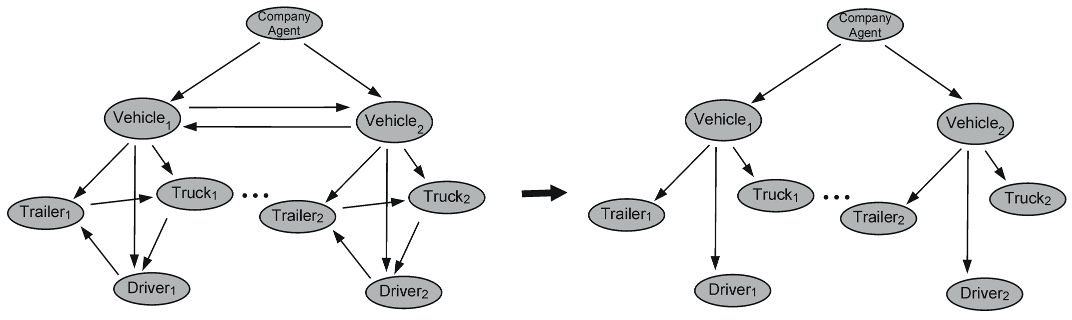


Figure 5.10: Establishing holonic structures in TELETRUCK

- The *chassis* that is supplied by components that can carry containers or swap bodies on a discrete nominal scale containing four elements [*one large chassis; one small chassis; two small chassis; no chassis*], and
- The *motor* resource on a discrete nominal scale with two elements [*motor; no motor*].

Since the allocation of any of these resources constrains the behavior of any component of a complete vehicle, cyclic dependencies among the component agents occur. Furthermore, the vehicle as a unit imposes additional constraints on the behavior of the component agents during the planning process by announcing the customer task it may finally be assigned to.

The route plans of various vehicles have to be set up by regarding the constraint structure of the plans. Hence, there is a cyclic dependency among the vehicles. Finally, the company as the highest entity in this domain constrains the behavior of the vehicle entities by granting or rejecting tasks to vehicles.

Applying the topology construction scheme of Section 3.5, the representation of the company and complete vehicles as agents is recommended, leading to an initial dependency graph as shown in the left side of Figure 5.10. (For the sake of simplicity we display only three different component agents and two vehicles.) Applying the graph transformation algorithm (Algorithm 5 in Section 3.5) and pruning redundant middle agents and arcs lead to an agent hierarchy as shown in the right part of Figure 5.10.

Although originally not modeled according to this structuring scheme, the TELETRUCK realization implements the above topology: In TELETRUCK, component agents are part of a holon that represents the vehicle for the transportation task. The vehicle holons are headed by a *Planning 'n' Execution Unit (PnEU)*, a special agent that is equipped with planning capabilities. The vehicle holons and the agents representing currently idle transportation units form a super-holon

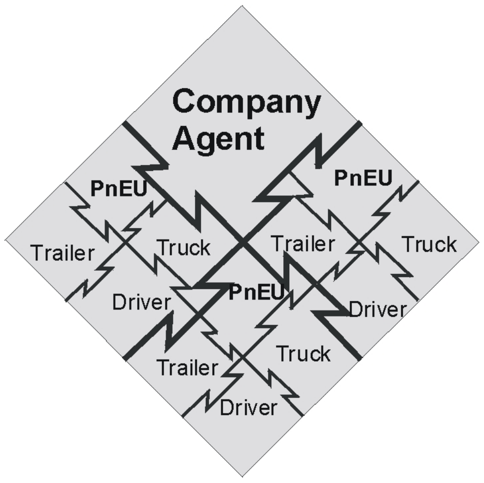


Figure 5.11: The holonic solution in TELETRUCK

that stands for the whole transportation company. The head of the company holon, called the *company agent* coordinates the interaction with the user and communicates with other companies that employ the TELETRUCK system. This modeling is in accordance with the methodology of a *holon as a moderated group* as discussed in Section 5.4.1. Figure 5.11 displays the hierarchy.

The TELETRUCK system allocates transportation tasks to the available transportation units such that the resource consumption is minimized. In this system, the *contract net protocol* and the *simulated trading procedure* (see Section 3.4.4) are used for resource and task allocation. All agents in the TELETRUCK realization follow our requirements for holonic agents of Section 5.3.2:

- **Autonomy:** Agents representing transportation units are autonomous in their decision to participate in a vehicle holon. Participating in the holon however restricts the autonomy of the sub-holons for the time span they are members, since they have to execute the sub-tasks allocated to them.
- **Common goal-directed behavior:** The agents forming a vehicle holon cooperate in order to pursue the goal of executing a set of transportation tasks. Different vehicle holons may also cooperate for a more complex task.
- **Increased group capabilities:** A vehicle holon is able to transport the cargo, which none of its components could do on its own.
- **Belief:** Agents in TELETRUCK have an explicit representation of the environment and the agent society.
- **Bounded rationality:** Because of the dynamics and the real-time requirements in this domain, an anytime algorithm is used for task allocation: the

run of the simulated trading procedure can be interrupted at any time and the current solution can be taken as a tour plan. Hence, the overall performance increases monotonically over time (if the environment does not change meanwhile).

- **Communication:** In TeleTruck, communication is structured in a hierarchical fashion. The company agent communicates with those agents that represent other companies, but also with the PnEUs, which in turn interact with the basic agents representing the components. Furthermore, in order to optimize the task and resource allocation, PnEUs communicate with each other to exchange tasks. Communication among companies is exclusively performed by company agents.

We shall present further details of the domain and the solution in a case study where we shall extend the original TeleTruck architecture (Section 7.1).

5.5.2 Flexible Manufacturing Systems

The Domain

In a manufacturing process, job-shop scheduling of work in a production plant must be optimized. A vital issue is the *dynamical scheduling* of a production plan, as it cannot be guaranteed in general that a schedule will be fulfilled: work stations may fail, supply parts may be out of stock, etc. Applying the criteria for holonic domains leads to the following results:

- **Actions of different scopes:** Obviously, there are plenty of actions at different levels of abstraction: At the very bottom level, actions are e.g., screwing and welding, higher level actions are the integration of modules to the work piece or at the highest level of abstraction, assembling a product.
- **Hierarchical structure:** This scenario exhibits basically two types of entities: work stations (with human workers and automated cells) and work pieces (consisting of smaller modules), each of which imposing a two-staged hierarchy.
- **Decomposability:** The overall problem of controlling a manufacturing plant can be canonically decomposed into subproblems: If a work station fails, only the schedules of the affected work pieces have to be modified.
- **Communication:** The domain imposes no restrictions on communication among entities. Communication between work station representatives and work piece representatives will be necessary if the work piece must be detoured from a broken work station to one in function.

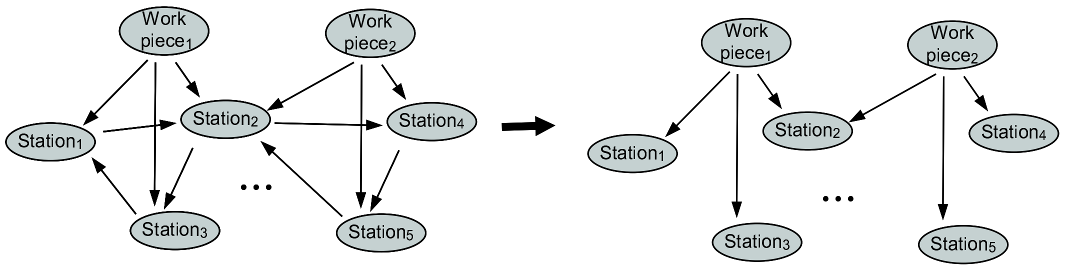


Figure 5.12: Establishing holonic structures in IFMS

- **Cooperativeness:** Clearly, this domain is strictly cooperative.
- **Situatedness and real time requirement:** A centralized re-planning of the whole schedule is often impossible, however, local re-scheduling may be feasible since this is a question of seconds while the system runs in terms of minutes and hours. Hence, bounded rationality is not really an issue if the replanning procedure is decentralized.

In summary, most criteria are fulfilled in the setting, in particular the two most important ones, namely hierarchical structure and cooperation. Depending on the degree of detail of the model, it may be reasonable not to model all entities in that domain by (holonic) agents as we shall discuss next.

The Solution

The *Intelligent Flexible Manufacturing System* (IFMS) has been developed in cooperation with experts from Daimler-Chrysler. The key idea of IFMS is to represent each work station and each work piece by an InteRRaP agent that plans and monitors the local schedule of its station or piece. If for some reason the current schedule cannot be executed anymore, agents re-plan the schedule in a decentralized manner.

In IFMS, work station agents control two types of abstract resource: *station functionalities* (on a discrete scale where each position on that scale is a combination of possible functionalities) and *idle time slots* (on a discrete scale). These resources induce interdependencies between work stations, in particular of those that are involved in the production of the very same work piece.

For a work piece agent, *production tasks* are viewed as abstract resources to be allocated to the holon members, inducing a dependency on the schedule of the involved stations. Figure 5.12 shows the initial dependency graph and the resulting holonic structure where for simplicity reasons only two work pieces with only three associated work stations each are shown.

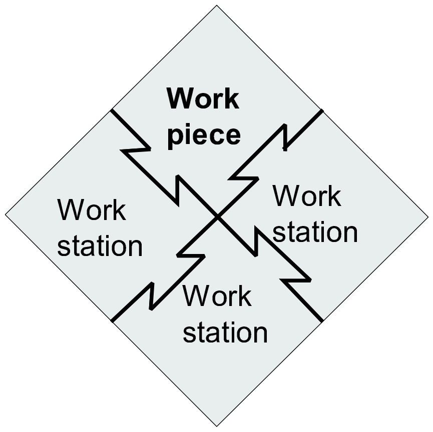


Figure 5.13: The holonic solution in IFMS

We consider a work piece as a holon that exists as long as the processing of that work piece lasts. After an initial schedule has been computed (prior to the actual assembly), a work piece is assigned to the set of work stations, which are involved in this particular manufacturing process. The holon consists of the work piece agent as its *head*, which controls the assembly of the product, while the work station agents are the *sub-holons*. A more detailed representation (e.g., the agentification of the automated parts or the human workers) is not necessary in this case, leading to a rather flat hierarchy (Figure 5.13). Note, that a work station agent can be a member in quite a number of holons at a time.

If during the assembly process one of the work stations fails, its corresponding agent has to leave the holon (since it can no longer provide the required resource), and the head has to find a substitute by announcing its need to all work station agents. These agents evaluate their extra effort on a local basis and send a proposal to the head which selects the best offer and invites that work station agent to join the holon. We now discuss the criteria for a holonic solution:

- **Autonomy:** All agents are autonomous, except that work station agents may not join a second holon, if that would clash with the schedules of the holon it is currently member of.

- **Common goal-directed behavior:** All agents have the common goal to finish the assembly of the work piece the head is assigned to.

- **Increased group capabilities:** In general, the construction of a work piece cannot be completed by a single work station as every work station assembles only those parts it is specialized for.

- **Belief:** The internal knowledge of the member agents is left untouched by the head. The reasoning of the head is at a higher level of abstraction, namely on facilities of a set of work stations.
- **Bounded rationality:** This domain does not really require bounded rationality for the agents involved.
- **Communication:** Member and candidate member agents communicate only with the head, not with other work station agents.

To summarize, as most of the requirements are fulfilled, we can model this flexible manufacturing system as a holonic agent system in a holonic domain. Similar to the first example, we will present a detailed case study on IFMS in Section 7.3, where the re-scheduling mechanism is described in detail, and furthermore, the issue of topology optimization is targeted.

5.5.3 The Coordination of Business Processes in a Virtual Enterprise

The Domain

Arnold et al. [AFHS95] define *virtual enterprise* as a *temporary federation of otherwise legally independent companies.* Usually companies form such a construct if they identify a short-term market opportunity that none of the partners could exploit alone. The above transportation scheduling and flexible manufacturing domains can be seen as special instances of business process management in general virtual enterprises.

A virtual enterprise is not institutionalized, it has neither employees nor offices of its own. Nevertheless, the partner companies of a virtual enterprise act as one single corporation when seen externally. The partners of a virtual enterprise contribute their core competences to the common business processes, hence the enterprise is usually able to provide services or products of high quality within a rather short respond time.

The coordination of the business processes within a virtual enterprise, especially the coordination of manufacturing processes or the supply chain management, is a more challenging task than the management of a classical firm. Since there is no hierarchy in a virtual enterprise, decision competence is often unresolved among the partners with clashing economic interests. This domain is naturally holonic:

- **Actions of different scopes:** Any business process can be modeled as an agent action, ranging from very elementary tasks to complex procedures.

- **Hierarchical structure:** Virtual enterprises have no overall institutional hierarchy. However, the member companies a virtual enterprise consists of usually do so, which leads to the effect that this domain shows structures only on its lower levels.
- **Decomposability:** The tasks of the virtual enterprise are the planning, distribution, and execution of business processes. These processes can be decomposed in elementary actions.
- **Communication:** There is a high degree of interaction required in order to coordinate the work of organizational units in companies and virtual enterprises.
- **Cooperativeness:** The setting has cooperative elements as well as competitive ones. In principle, the partners within a virtual enterprise aim at a common goal which has been the original reason for their partnership. Nevertheless, the allocation of tasks and the distribution of profit is competitive. Even in a single company, there are competitive situations when sub-units have to compete for limited resources.
- **Situatedness and real time requirement:** By their definition and purpose, virtual enterprises have to react instantly to dynamic changes in the market.

Outline of a Solution

A software solution represents the knowledge of the virtual enterprise at several levels of abstraction, it has to plan and supervise business processes among several companies, and the precise allocation of tasks and resources has to be administered.

A virtual enterprise can be modeled holonically as follows: the organizational structure of the virtual enterprise is modeled as a holon, whose sub-holons are the individual companies that in turn are decomposable into sub-sub-holons that represent the different departments or subsidiaries. Companies without a hierarchy can be represented as holons in which the head has only a rather limited, moderating competence.

The exhaustive modeling of all processes in a virtual enterprise with a holonic multi-agent system is a large and visionary task. But this task becomes more tractable if a larger structure can be formed from different holons that represent sub-companies at different positions in the business process. For example, a multi-agent manufacturing system and a system for supply chain management implemented in the same holonic framework could be linked at a higher stage into a system that is able to coordinate the supply with the manufacturing process. First steps to derive a software solution are described in [RV98].

5.5.4 RoboCup

The Domain

The RoboCup initiative [KAK+97] defines and coordinates the "official Soccer world championships for physical robots and software agents".[4] In the *simulation league* for software agents, each player is represented by a separate program that is connected via TCP/IP to a central simulation server.

Every 100 milli-seconds, a player program can perform an action (dashing, turning, kicking, catching, communication with other players) by sending an appropriate string to the server which in turn computes the effect of that action. Every 150 milli-seconds such effects are transformed into the local perception of the agents and sent to them as percepts. Perception is more blurred the farther away entities are from the receiver. The server treats communication just like any other action; hence, long-distance communication is disturbed or not possible at all. Although the RoboCup domain has not been developed for the design of holonic agent systems, its design supports holonic architectures for the following reasons:

- **Actions of different scopes:** In the RoboCup simulation engine, atomic actions can be performed by any player program. However, there are also complex, strategic actions which involve several players (e.g., a double-pass). The simulation engine also contains a "coach" program for each team. This program has a more global perspective of the scene and it can communicate with all agents of its team. Such a strategic communication action is more abstract than the other actions.

- **Hierarchical structure:** A priori, all players have equal status. However, it turns out to be of advantage that some players are designated to manage the offense block, the defense block, or the mid-field. At the next hierarchical stage, the coach agent gives guidelines to these regional leaders.

- **Decomposability:** The overall task to win a game can be decomposed into offense goals (to score many goals) and defense goals (to avoid scores of the opponent team).

- **Communication:** Players can communicate with only those players that are not too far away (in general, players of the same region). The coach can communicate with all members of the team.

[4]For details see `http://www.robocup.v.kinotrope.co.jp/02.html`

- **Cooperativeness:** Within one team, the setting is cooperative, between teams it is of course competitive.
- **Situatedness and real time requirement:** This scenario poses strong real-time requirements, where a player program has to cope with rapidly changing percepts within 150 milli-seconds in order to determine the next action. Hence bounded rationality is an issue.

The Solution

Each player of the *CosmOz* team [Jun98] system is implemented as an InteRRaP agent and is assigned to a region (offense, mid-field or defense). Each player controls a set of individual resources; we present some selected examples (a much more detailed description can be found in [Jun99]):

- **Stamina** (on a integer-valued scale from 0 to 2500) simulates the "physical condition" of a player and this is controlled by the central simulation server of the RoboCup initiative. Every action of a player results in a decrease of stamina; a low stamina reduces the speed of an agent on the grid, i.e., the player gets more and more exhausted. Performing no action for a while increases the stamina (the player recovers).

  Several threads inside a CosmOz agent (representing e.g. the actions *move*, *kick* and *turn*) apply for the stamina resource and based on expected utilities of these actions, a fast and greedy mechanism allocates this resource.
- **Aim** (on a nominal and discrete scale [*aim; no aim*]): Again, several threads inside an agent (for example *aimSelf*, *aimPlayer* and *aimGoal*) apply for this resource. It is assigned on the basis of local utility measures and the winning thread performs the envisaged action, for instance to pass to a colleague or to shoot at the goal.

These micro-level resources do not lead to an interdependency between different player agents. However, the original CosmOz architecture can be extended in order to further structurize the agent society by introducing *roles of members in a block*, *tactic*, or *line-up* as macro-level abstract resources (all on discrete and nominal scales, where each configuration denotes one option) These resources can be controlled by the coach agent and the regional leaders. For instance, if the team is only closely leading and the game is almost over, all player agents should be assigned to the defense.

An initial resource dependency is shown on the left of Figure 5.14; the final holon structure after applying Algorithm 5 and removing redundancies is shown

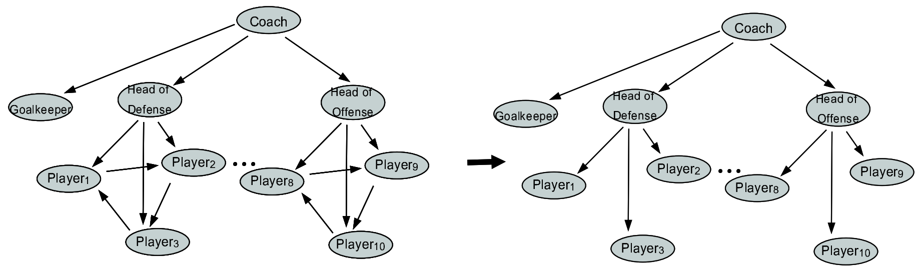


Figure 5.14: Establishing holonic structures in the CosmOz team

on the right. Only for clarity reasons we have left out the mid-fielders. The resulting agent society is viewed as a holon in the paradigm *holon as a moderated group* where the coach takes the role of the head. Each player joins into the super-holon of the *regional players* whose head is the *regional leader*. This super-holon in turn is a sub-holon of the holon *team*, whose head is the coach (See Figure 5.15).

Since for resource allocation some heuristics are applied, it implements the *coordination approach based on heuristics* of Section 3.4. We now check the requirements for a holonic solution:

- **Autonomy:** The coach and all players are independent programs that communicate over TCP/IP, hence agent autonomy is predefined. However, as players are commanded by the coach and the regional leaders, their autonomy is restricted in the sense of our holon definition.
- **Common goal-directed behavior:** Obviously, the team members have a goal they all strive for: to win the next game (and to become world champion eventually).
- **Increased group capabilities:** At the highest stage, the team stage, the execution of strategies (e.g., playing in an offensive or defensive style) is realized by the whole team. Cooperative actions such as double-pass are defined at the next lower stage, but still no single agent could execute them alone.
- **Belief:** Every player agent has to maintain its own world representation (for instance, its own position and stamina, the relative position of the ball). At the group stage, the coach agent represents the state of the whole team and of the opponent team (e.g., score, tactic, time to play, average stamina).
- **Bounded rationality:** CosmOz agents show bounded rationality since the abstract resources define limits for the individual computation.

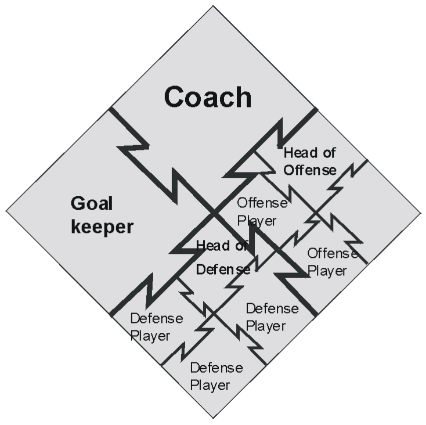


Figure 5.15: The holonic solution in CosmOz

- **Communication:** In the RoboCup setting, communication is restricted to agents within a certain geographical distance. However, communication to the head of the holon, i.e., the coach, is always possible. Communication to agents outside of the holon (the players of the other team) is not provided.

In summary, all requirements for a holonic multi-agent approach are fulfilled and this domain supports a holonic modeling. In particular dynamic strategies to form a group (such as double-pass, offside trap) can be realized easily by a dynamic configuration and reconfiguration of holons.

Bottom Line

Inspired by biological systems in the sense of Koestler and Minsky, we have defined the concept of holonic agent systems as an extension of resource-adapting multi-agent societies. The main advantage of the holonic approach is the chance to recursively map an application domain directly and naturally onto a holonic multi-agent system where the agents are again composed of agents.

We have explored the whole spectrum of this new paradigm, ranging from definitorial issues over classification of possible application domains to implementational aspects: Based on general criteria for the distinction between holonic and non-holonic domains, we have examined domains suitable for holonic agents. The criteria have also been employed as a design aid for modeling and implementing holonic agents in several settings. We have employed the GRAIL approach of Chapter 3 for holon formation and on-line re-configuration and we have shown how to extend the InteRRaP agent architecture to work as a basis for holon-oriented programming which has been used in three sample applications.

Chapter 6

Case Study: Applying GRAIL to the MoTiV-PTA Agent System

The theoretical foundations for self-adaptation in agent societies have been presented in Chapter 3 and a tool set that supports a canonical realization of societies has been proposed in Chapter 4, so we shall now turn to our first case study, the MoTiV-PTA system. First, we give a brief overview of the system and then we present a SIF-based simulation engine that allow to run sufficiently realistic simulations. Finally, we use SIFIRA to integrate a resource adaptation scheme into the scenario and show the results of a number of test runs to validate our approach.

6.1 The Domain

Individual mobility has increased significantly over the last years: a growing number of people travel over more and expansive distances, so the demand for transportation increases enormously. In part, this demand is met by the construction of new roads and the expansion of railway and airway systems. However, such extensions cannot catch up with the increase in demand. Hence, new approaches must be introduced to enable more efficient use of existing transportation means.

A second phenomenon of the 1990's is the tremendous increase in electronically available information, mainly because of the increased performance of computer hardware, the massive use of wireless communication facilities and the broad acceptance of the Internet. These new media allow the use of integrated information systems in telematics applications of dimensions not conceivable a couple of years ago.

The MoTiV-PTA[1] project has been initiated by the German Ministry for Education, Science, Research and Technology (BMBF) to develop a unified distributed telematics system for planning and supporting individual inter-modal travel. Industrial partners are BMW, Bosch, Daimler-Chrysler, IBM, Opel, Siemens, VDO, and Volkswagen. During a trip, the MoTiV-PTA system does not only assist a user for car navigation, it also accompanies him while using basically any transportation system. The user of such a system shall be able to get on-line information on different means of transportation, and accommodation, obtain a detailed proposal of a complete itinerary with the best expected user satisfaction (including time schedule and route plan, hotel suggestion etc), and conveniently book the chosen means of transportation and accommodation.

6.2 System Architecture of the MoTiV-PTA Prototype

6.2.1 Overview of the MoTiV-PTA Functionalities

There are three types of user interface devices: first, a notebook-like tool, called *mini-Personal Travel Manager (PTM)* or *Personal Intelligent Communicator* (PIC) where the user accesses the system via Windows^{TM}CE-based devices running a dedicated graphical user interface, second, a system to be integrated in the user's car dashboard and third, a piece of software which is run on the user's PC. The system running on a PC is intended to have all necessary capabilities for off-line performing the travel managing tasks. The capacities of PICs and dashboard tools, on the other hand, are much too small to perform complex planning and negotiation processes. These tools are designed for mobile on-line usage: they communicate with more powerful mirror modules over GSM or ISDN.

The integrated information providing services are distributed on different servers, accessible via Internet. The key idea here is to provide an interface to already existing information sites in order to allow for automated information retrieval. The MoTiV-PTA specification contains the following travel services:

- inter-modal route planning combining different transportation modalities such as railroad, automotive and airways in a route from a given starting point to a specified destination,
- combination of automobile route planning with car park scheduling, for instance to obtain an optimal car park for the destination (based on availability of parking spaces) and to calculate the route to the car park,

[1]German acronym for *Mobility and Transport in Inter-modal Traffic - Personal Travel Assistance*

- traffic monitoring on e.g., traffic jams or car park information,
- hotel information and reservation, and
- tourism information.

6.2.2 Realization

An agent-based approach has been selected for developing this system, since the following requirements can be dealt naturally by a multi-agent system:

- **Distribution:** Timely and geographical distribution of users and services have to be taken into account.
- **Heterogeneity:** User support, services, devices and networks are provided by different sources.
- **Co-ordination and communication:** Optimal user satisfaction can only be achieved if co-ordination and communication between users and service providers is provided.
- **Mobility:** Users and user devices may change their physical and logical position over time.

The MoTiV-PTA system is implemented in JAVA^{TM} using agents of the MECCA architecture. The use of JAVA^{TM} leads to platform independence while the employment of the MECCA agent architecture is advantageous since its supports the use of the FIPA agent communication standard. A MoTiV-PTA prototype has been realized in compliance to part 4 (Personal Travel Assistance) of the FIPA 98 specification which defines an *Agent Communication Language (ACL)* and specifies agent services such as the *Directory Facilitator (DF)* and the *Agent Management System (AMS)*, and interfaces to pre-existing information providers.

The user accesses the system via a *user device agent (UDA)* running on his local device. The UDA communicates via ISDN or GSM with a MoTiV-PTA server. Here a *dispatch agent* distributes the incoming messages to the corresponding *user agent (UA)* of the UDA. The UA manages the user preferences and sends requests taking the preferences into account to a *broker agent* to plan a complete journey.

The broker agent decomposes the request into requests for individual *service agents*, such as a *car park manager* for finding a parking lot or an *inter-modal route planner (IMRP)* for calculating the route. For achieving up-to-date information, this agent contacts information providers, for instance agentified railway

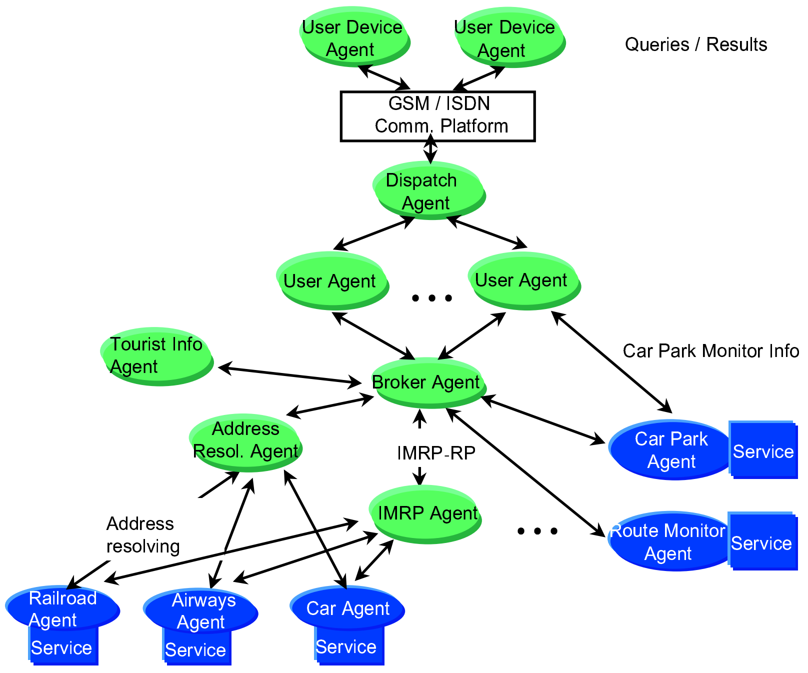


Figure 6.1: The agent society in MoTiV-PTA

or airline information systems. The broker agent combines the results and passes the alternative itineraries to the user agent. This agent sends the results to the user device agent which presents them to the user.

The user chooses among the prepared alternatives, the chosen itinerary can then be monitored for unforeseen events, such as traffic jams on the scheduled route or over-booking of the chosen parking lot. If the current itinerary is no longer feasible, re-planning is automatically initiated, and the user is informed of the consequences. Figure 6.1 visualizes the MoTiV-PTA agent society where DF and AMS are left out for simplicity reasons.

Before a system such as MoTiV-PTA can be introduced into the market place, a series of test runs and simulations have to be performed. We use the generic simulation framework SIF proposed in Chapter 4 to build a simulation engine to achieve a successful application.

The extension SIFIRA is employed in *off-line simulations* to determine optimal system configuration for a given environment profile. SIFIRA can also be used in *on-line simulations* (and later, in realistic runs) to adapt the system to dynamically changing environments and to achieve scalability. The overall optimization task is thus to minimize the communication effort and run-times of the various services such as information retrieval, bookings, etc., in the MoTiV-PTA system. Since the types of services may vary and the setting of the network may also diverge greatly, we focus on optimizing the average duration of services during the daily run of the system. In Section 6.4, we present the optimization procedure in detail. Section 6.5 shows empirical results that underline the applicability of the approach.

6.3 A SIF-Simulation Engine for MoTiV-PTA

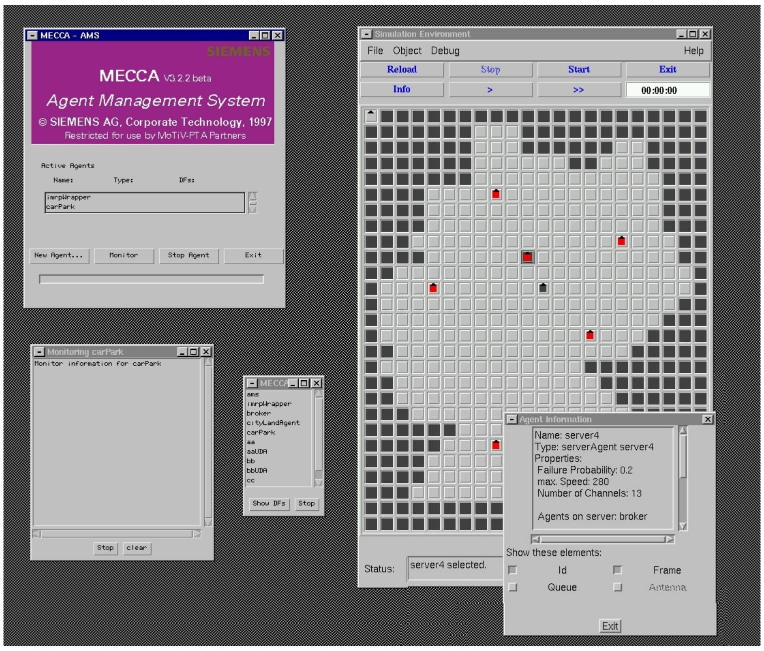
In the following we describe all extensions of the MoTiV-PTA system to a sufficiently realistic simulation testbed. First, we give an overview of the implemented simulation engine; in Section 6.3.1 we show implementation details. The key idea for the simulation is to introduce SIF agents for an explicit representation of hardware servers. In Section 6.4 we use SIFIRA agents for this task in order to additionally optimize the overall performance of the system.

6.3.1 Architecture of the Simulation Engine

For a realistic simulation we represent explicitly the hardware servers and communication duration. Additionally, we simulate the services provided by the MoTiV-PTA prototype that runs on these servers, because the overall performance of a MoTiV-PTA system does not only depend on the efficiency of the agent architecture but is also heavily influenced by the following factors:

- The user behavior may vary over location and time: Certain types of requests may be initialized more often from one part of the domain than others. Additionally, there may be a work load distribution on a temporal basis.
- Servers might be overloaded or might break down; usually they vary in terms of equipment and are geographically dispersed.
- Depending on the type of the communication medium, e.g., Internet, GSM-bearer, or ISDN, the medium throughput is different.
- The duration of services may also differ. This depends on the used server, but also on the work load of the service providers induced from other non-PTA based sources (e.g., a railway information provider may also be accessed by a vast number of travel agencies).

The above factors lead to different service run times dependent on the work load and the configuration of the servers the MoTiV-PTA agents are logged on. In order to integrate these factors into a simulation engine, we apply modifications



imrpWrapper
carPark

Monitor information for carPark

ams
imrpWrapper
broker
cityLandAgent
carPark
aa
aaUDA
bb
bbUDA
cc

00:00:00

server4 selected.

Name: server4
Type: serverAgent server4
Properties:
Failure Probability: 0.2
max. Speed: 280
Number of Channels: 13

Agents on server: broker

Figure 6.2: Screen shot of the system

to the MoTiV-PTA prototype. However we want to interfere with the original MoTiV-PTA system as little as possible in order to avoid side effects between the simulation engine and the system to be evaluated.

Figure 6.2 shows a screen shot of the integrated simulation system: On the left, the original MECCA agents are shown while the right-hand side displays the console simulation engine (indicating the locations of the servers on a rough map of Germany) and an information window that indicates the state of a server. The following extensions of the original MoTiV-PTA architecture are made:

Explicit representation of servers: In a first step we extend the original MoTiV-PTA architecture by aggregating each MoTiV-PTA agent with a SIF agent representing a hardware server that implements the communication platform of a MoTiV-PTA agent (see Figure 6.3). We do not use simple JAVA^{TM}objects, but autonomous agents for representing servers in the simulation in order to model more accurately server failure behavior: an agent can autonomously determine and change its state according to the simulation parameters. The use of MECCA agents would spoil the accuracy of the simulation since such agents have to register at DF and AMS that are part of the system to be simulated. We use the SIF framework for this purpose since SIF is especially designed for simulation purposes and also written in JAVA^{TM} allowing an easy integration.

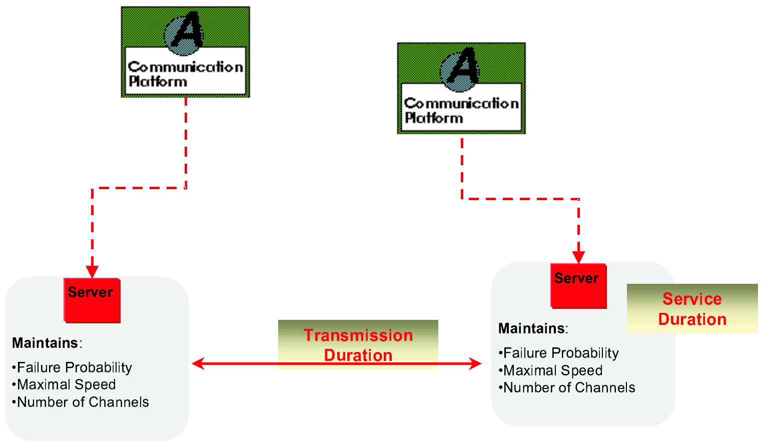


Figure 6.3: Representation of hardware servers

In MoTiV-PTA, all agents are derived from the abstract base class *MECCA-Agent*. In order to generically model the link from a MoTiV-PTA agent to a SIF server agent, we introduce a new abstract class *MECCASimAgent* that extends *MECCAAgent* and provides a reference to a SIF server agent. All MoTiV-PTA agents now extend *MECCASimAgent*. Figure 6.4 shows the correlations, where original MECCA components are marked darker, while original SIF components are lightened.

Integrated long-term simulation: Since there are many factors that influence run times in MoTiV-PTA, we provide an integrated long-term simulation environment, where user requests are posed according to a frequency distribution in terms of geographical distribution of the user, temporal distribution of the number of requests, and a distribution of the type of requests the user can submit. The process duration for these requests can be measured and used for statistical evaluations. SIF offers constructs for simulating these distributions: The SIF *world server* can easily be extended to control the timely emission of appropriate requests.

Simulation of delay of communication among agents: Each SIF server agent is assigned to a certain simulated position in the network. Whenever two MoTiV-PTA agents communicate, a logical distance between their two servers in the simulation is determined which needs not to correspond exactly to the geographical distance. Due to the structure of the Internet, transmission between very close sites is not necessary faster than between far sites. Communication duration has to be simulated depending on the logical distance of the server. We employ the SIF world server to control the durations of communication acts.

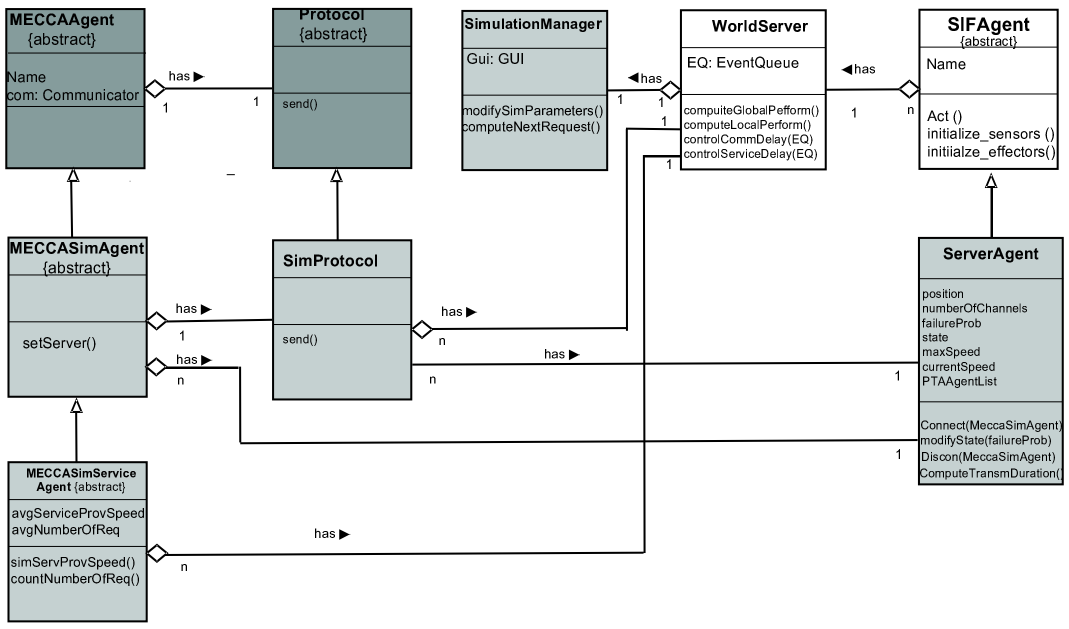


Figure 6.4: Architecture of the simulation engine

Migration: We extend the scenario by enabling user agents to *migrate*: In a traditional approach, communication between geographically distributed agents is performed over the network. This can be rather time consuming in case the network is heavily loaded and the communication process consists of some complex negotiation procedures. In the *migration* approach, an agent is transmitted over the net in order to communicate with its partner on the local server. A user agent, for example, might travel from home to a ticketing place to obtain theater tickets. Later the agent might travel back home to describe to its user the tickets it has obtained.

This approach is fast if the receiving server has an accurate model of the traveling agent. In this case, only data describing the agent's state (i.e., the instance variables) have to be transmitted; a copy of the agent can be generated locally. In this simulation migration can be realized even more simply: An agent has to modify only its server agent link.

It may make sense to let only those agents migrate that are expected to do a lot of long-distance communication. Such a setting is given if, e.g., the user (and his UDA) have moved temporarily far from the location of the UA. For the communication between UDA and UA, distance is irrelevant, since it is done over GSM or ISDN; however since many of the tasks the system performs are used for information retrieval (i.e., communication with service agents) of the new logical neighborhood of the user, it is reasonable to allow the UAs to migrate to a server closer to the service agents and to integrate this feature into the simulation.

Registration of UDAs: The logging processes of UDAs to servers are also simulated: whenever the user emits a new request, the corresponding UDA which is located on the PIC or any other local system has to connect to the server of the user's UA which may be its home server or some other server if the user agent has migrated previously.

If for some reason logging to that server is not possible, the UDA must log onto another server in the simulation; the choice is made on the distances and on the expected current speeds of the new candidate servers.

Simulation of services: All services in the simulation scenario are replaced by dummy procedures that do not actually provide the real service, but do determine a simulated service duration time on which the answer is delayed.

6.3.2 Simulation Constructs in Detail

The following details of the simulation have been established in cooperation with experts from Siemens. The resulting values for service and transmission duration do in principle represent real-world circumstances. The achieved accuracy suffices for the realized prototype which has been developed to demonstrate the applicability of the approach.

A *SIF server agent* maintains the following instance variables, as shown in Figure 6.4.

- $position = [xPos; yPos]$, $xPos, yPos \in [0; 20]$, the location of a server,
- *failureProbability* $\in [0; 1]$, the reliability of a server,
- *state* $\in \{$down, up$\}$, indicating whether the server is down at the moment (depending on the failure probability),
- *PTAAgentList*, a list of all MoTiV-PTA agents running on that server at a time,
- *numberOfChannels* $\in [0; 50]$, determining how many channels can be opened at most for connecting user device agents,
- *UDAList*, a list of all user device agents that are currently connected (up to the maximum number of channels),
- *maxSpeed* $\in [0; 400]$, the maximum speed of the machine in some abstract measure,

- *currentSpeed* $\in [0; 400]$, the current processing speed depending on the number of agents running on that server and maximal speed of the server:

$$\mathit{currentSpeed} = \mathit{maxSpeed} * \min(1, \frac{5}{\mathit{numberOfChannels}})$$

For simulation of *communication delays*, a new communication protocol *SIM-Protocol* is introduced to MoTiV-PTA. This protocol bases on the method invoking MECCA protocol *localProtocol*, but enables the simulation of communication duration: For each performed communication act in MECCA, a new instance of a protocol is activated (see Section 2.1). If a *SIMProtocol* is used, the duration (in milliseconds) of a transmission from server s_1 to server s_2 is computed according the formula

$$\sqrt{(\Delta xPos)^2 + (\Delta yPos)^2} * \frac{1000}{6} * (2 - \frac{\mathit{currentSpeed}(s_1)}{400})(2 - \frac{\mathit{currentSpeed}(s_2)}{400})$$

In case of communication between UDA and UA, a constant time of 2000 msec is used for GSM communication which includes the establishment of the connection and the transmission of the message. In the simulation, the SIF world server controls the simulation time; only if the computed transmission time has passed, the communication act is sent to the receiver's communicator.

We enable those UAs to *migrate* whose current distance to the corresponding service agent is greater than a certain minimum distance. A user agent can only migrate if the server in question is currently not down (which is determined by the corresponding SIF server agent). If the server is down, another server is selected for the attempt to migrate. The one server is selected that maximizes a tradeoff between speed and distance to the broken server:

$$\frac{\mathit{currentSpeed}^{\;2}}{\sqrt{(\Delta xPos)^2 + (\Delta yPos)^2}}$$

In the simulation, a MoTiV-PTA agent migrates from one server to another by simply exchanging the aggregated server agent and by modifying the *PTA-AgentList* vectors of both server agents. The duration of a migration attempt is computed according to the following formula:

$$\sqrt{(\Delta xPos)^2 + (\Delta yPos)^2} * 500 + \frac{10000}{\mathit{currentSpeed\ (target)}}\ [msec]$$

Logging of an UDA on a server is simulated similarly by changing the aggregated server agent from PIC to the selected server and by adjusting *UDAList*s accordingly, if that server is active and has unused communication channels at its disposal. Otherwise, another server has to be found, just like in the migration case. The duration of that procedure is computed similarly to migration duration:

$$\sqrt{(\Delta xPos)^2 + (\Delta yPos)^2} * 2000 + \frac{10000}{\mathit{currentSpeed\ (target)}}\ [msec]$$

UDAs are modified to directly aggregate to the SIF system: Each UDA holds a reference to the world server which forces the UDA to emit a request from time to time. Furthermore, the UDA has to report on the reception of an answer to that request.

The answer of a *dummy service* is delayed according to an abstract *serviceProviderSpeed* $\in [0;500]$ and the current *numberOfRequests* $\in [1;\infty[$ which indicates the number of requests a service source is currently processing. These requests need not necessarily all be imposed by the MoTiV-PTA system, but can also be posed through other media, such as call centers. To realize the delay in a generic fashion, we introduce yet another abstract MECCA agent *MECCASimServiceAgent* which extends *MECCASimAgent* by these two variables and by a reference to the world server. All MECCA dummy service agents in the simulation extend *MECCASimServiceAgent* (See Figure 6.4).

The world server can modify the number of requests during the simulation run. The world server simulates the process delay for a service according to the formula

$$\frac{500}{serviceProviderSpeed} * log_{10}(\mathit{numberOfRequests}) * 1000\ [msec]$$

6.3.3 Functionally of the Extended Scenario Script Parser

The MoTiV-PTA system has been enriched with the SIF script parser in order to allow the specification of scenario parameters that do not vary during a run of the simulation. This way we need not always modify and re-compile the main source code for scenario changes. We now briefly describe the elements of the simple script language. Figure 6.5 shows an example script.

Relevant Entities for the SIF Simulation Environment

- `Country` <***country***> A script line beginning with the key word `Country` determines the general type of the scenario. The user can specify a country, an abstracted map of that country will be then be loaded and displayed in the main simulation window (see Figure 6.2 which displays a map of Germany).

```
Country Germany
Migration 22
ServerAgent server1 14 24 0.1 100 25
ServerAgent server2 11 10 0.2 80 15
ServerAgent server3 11 22 0.15 150 20
ServerAgent server4 17 9 0.05 200 10
Target server1

DF df server1
AMSAgent ams df server1
IMRPAgentWrapper imrpWrapper ams df server3 300 200
BrokerAgent broker ams df server9
CarParkAgent carPark ams df server1 250 150
UserDeviceAgent johnUDA ams df
UserAgent john johnUDA ams df server1
UserDeviceAgent maryUDA ams df
UserAgent mary maryUDA ams df server1
```

Figure 6.5: A sample scenario script

- `ServerAgent` ***<serverName>***, ***<xPos>***, ***<yPos>***, ***<failureProb>***, ***<maxSpeed>***, ***<numberOfChannels>*** A script entry beginning with the keyword `ServerAgent` leads to the creation of a SIF server agent named *<serverName>*, located in the map at position [*<xPos>;<yPos>*] and equipped with the specified failure probability, number of channels and maximum speed.

- `Migration` ***<number>*** This parameter defines the initial value of *minimum migration distance*: Only user agents that are located farther to the target than this value are allowed to migrate.

- `Target` ***<serverName>*** For the sake of visibility and simplicity we focus on the simulation of requests stemming from UDAs located on *different* servers which ask for information concerning only *one* location in the web. This can be specified by naming a *target server*. Requests to other locations could be simulated as well; however results are rather independent from the simulation of requests aiming to other locations. Hence, we focus on only one target server.

Original MECCA Entities

- `DF` ***<DFName>***, ***<serverName>*** The user can specify a directory facilitator which is simulated to run on a server named *<serverName>*.

- `AMSAgent` <***AMSName***>, <***DFName***>, <***serverName***> Similarly, a named AMS is created which runs on the specified server and registers at the named DF.

- `UserDeviceAgent` <***UDAName***>, <***AMSName***>, <***DFName***> A user device agent is created that is registered at <*DFName*> and <*AMS-Name*>. This user agent is initially not associated to a certain server since it is simulated to run on the user's individual device (e.g., a PIC).

- `UserAgent` <***UAName***>, <***AMSName***>, <***DFName***>, <***UDA-Name***>, <***serverName***> A user agent is created which corresponds to <*UDAName*> and registers at the specified DF and AMS and runs on the given server.

- `BrokerAgent` <***BAName***>, <***AMSName***>, <***DFName***>, <***server-Name***> A broker agent is created which registers at the specified DF and AMS and runs on the given server. In analogy, agents of the type *Re-solveAddressAgent* can be specified.

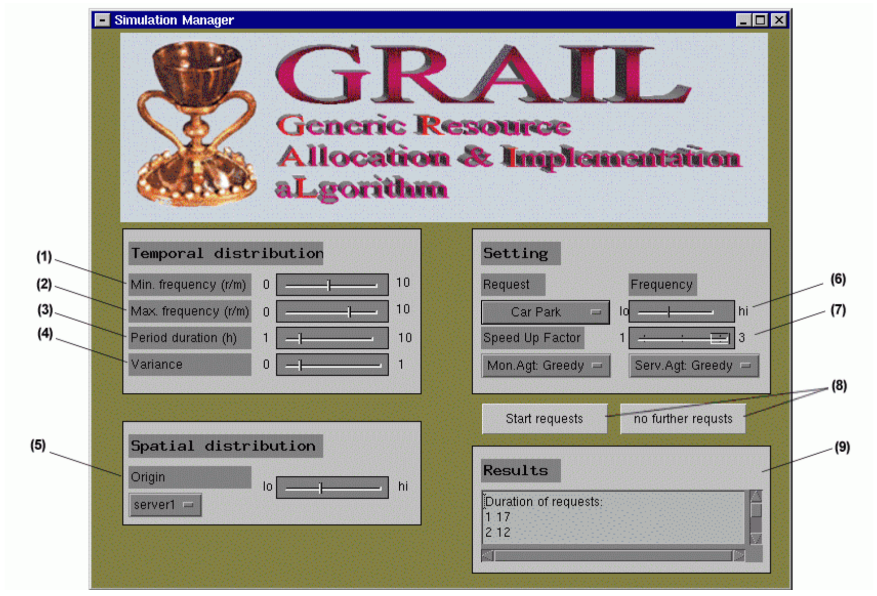
- `IMRPAgentWrapper` <***IAName***>, <***AMSName***>, <***DFName***>, <***serverName***>, <***avgServiceProvSpeed***>, <***avgNumberOfRe-quests***> A dummy agent for inter-modal route planning is created. The last two parameters express the speed and work load of the service provider. Similarly, agent wrappers of the type *CarParkAgent*, *CarAgent*, *Air-waysAgent* and *RailroadAgent* can be specified that agentify different types of information services for uni-modal travel.

6.3.4 User Interface of the Simulation Engine

Whereas scripts are used to load scenario parameters that do not change during the run of a simulation, SIF offers functionalities which can be modified to control an experiment during the actual run of the system. Additionally, a new graphical user interface, the *simulation manager* helps to modify parameters during the actual run of the system. As already displayed in Figure 6.4, this module is aggregated to the world server.

Control Features of SIF

Buttons: The SIF console provides a number of buttons for user interaction. All buttons are active if their functionality is currently available. Pressing the *start* button activates the world server and the agents while pushing the *stop* button halts the computation. Using the *reload* button deletes all agents and resets



Duration of requests:
1 17
2 12

Figure 6.6: User interface of the simulation manager

the world representation of the simulation engine. Furthermore, the previously selected script is reloaded. Pressing the *information* button opens an *object info window* (see below) which contains information on a selected agent.

Menus: The *file* menu offers general functions such as to *load* a new script or *save* the current state of the simulation as a script. By clicking on *edit*, scripts can also be modified. The *object* menu offers functionalities to *kill* previously specified agents and *display information* of the internal state of an agent.

Object Info Window: For every server agent, this window displays static information like its name, position, failure probability, number of channels, maximum speed, but also dynamic information such as current speed, currently logged MoTiV-PTA agents and used communication channels. (Figure 6.2 shows shows such a window at the bottom of the right side.)

Control Features of the Simulation Manager

The simulation manager (Figure 6.6 shows its GUI) has been created for the convenient on-line modification of simulation parameters that are specific for MoTiV-PTA. In particular, the request frequency distribution can be modified during the run of a simulation. Furthermore, the simulation speed can be changed for demonstration purposes.

Temporal distribution (1) - (4): The simulation manager allows to specify a periodic request frequency in order to simulate times of different work load. The experimenter can define a sinus curve by specifying the values of the maxima **(1)** and the minima **(2)** of the curve (in number of requests per minute) and its period **(3)** (in hours): Whenever a request has been posted by one of the UDAs, the world server computes the next time point for posing a new request according to the above specified temporal distribution by computing a value which expresses how long to suppress the emission of a new request. A normal distribution is additionally laid on top of the periodic distribution in order to blur the sinus curve: The previously computed delay time is taken as the expected value, while a user-defined factor **(4)** of that value is taken as the variance.

Spatial distribution (5): A spatial distribution of requests can be realized by defining several UDAs to run on differently located servers. The simulation manager allows to specify on-line a number between 0 and 1 for each of the created servers. This value denotes how frequently requests are launched from this server. The actual likelihood for each server is computed by dividing this number by the sum of specified numbers.

Type of requests (6): In the MoTiV-PTA demonstrator the user can choose from a variety of different services, such as inter-modal route planning, car park reservation, etc. In this field the experimenter can specify the likelihood of each type of request to occur. This is done by using the slider to assign a number between 0 and 1 to each service. The actual likelihood of this service is then computed as the ratio between the specified number and the sum of all inserted numbers.

Simulation speed-up (7): Using this slider the experimenter can speed up the simulation: When put in the middle value, communication delay is reduced to one third while service duration is reduced to 40%. Requests are emitted three times more often. When put on the right value, communication delay is even reduced to one tenth while service duration is reduced to one fifth. New requests are then emitted five times faster.

Starting and Stopping the MECCA part of the simulation (8): By pressing the bottom *Start Requests*, the UDAs are forced to emit requests according to the above frequency distribution. If the bottom *No further Requests* is pushed, the UDA will no longer pose requests; already emitted requests are still being answered by the system.

Result Box (9): This box displays the process duration of previously answered requests. This information is then passed to the world server which determines the processing time and stores it for statistical analysis.

6.4 Optimization of MoTiV-PTA with GRAIL

6.4.1 Overview

Scalability and stability of the complete MoTiV-PTA system depend on the accuracy of the server configuration. In order to meet these goals, we propose to integrate an optimization scheme for the server society for the off-line case as well as for the on-line case.

In the following, we exemplify such an optimization by using the simulation environment of the previous section. As stated before, this simulation engine bases on a demonstrator of MoTiV-PTA and hence simplifies circumstances in order to show in-principle applicability. The optimization presented in this section bases on this system; its purpose is to show in-principle feasibility of the GRAIL approach.

As specified in Section 6.3, in the simulation scenario each server can be configured according to *failure probability*, *number of channels* and *maximum speed*. Goal of the optimization scheme is to find optimal configurations for all servers, i.e., optimal trade-offs between the three quantities: In the on-line case, these optimal trade-offs have to be modified if the environment changes (for instance, the frequency distributions of the type of requests or their origin varies.)

In order to model such a trade-off, we introduce the notion of an *abstract currency*. In the off-line simulation abstract currency can be used to simulate the payment of fixed hardware costs; in the on-line case, credits of abstract currency can additionally be used to simulate costs of running the system (such as personnel or hardware replacement costs). A fast machine is more expensive than a slower one; a machine with smaller failure probability or one with more UDAs channels is more expensive than a less reliable one or one with only few channels.

Besides the optimization of each server configuration, there are macro-level issues to be regarded, such as the *number of servers*, their *geographical dispersion* or topology, the *distribution of abstract currency* credits to the various servers given a fixed overall maximum, etc.

The Approach

Clearly, for the relatively independent tasks of server optimization, a *decentralized problem solving* approach is suitable. However, there are also arguments promoting a *centralized approach*:

- The above mentioned macro-level issues favor a centralized decision making procedure.
- The system has to be optimized according to the global task to minimize the average run of a MoTiV-PTA service, as stated above. In addition, feedback is also returned in a global fashion: the actual duration of the runs.

Since both, centralized and decentralized approaches have their pros and cons, we employ a hybrid approach that unites both perspectives: GRAIL. It is instantiated on two stages: a global stage and an underlying local stage. Running GRAIL on the server agent society allows dynamic adaptation of each single server and the entire server topology to the current demands posed by the environment. This technique is related to agent-based optimization of ATM networks (see e.g., [Hay98]); the main difference is that our optimization operates on the servers (i.e., the nodes in the network) while ATM optimization considers the connections (i.e., the arcs in the net). Optimizing connections is not applicable in our case, since the MoTiV-PTA system runs on the Internet which cannot be manipulated in such a fashion.

For each server, we can run a RAL to optimize its configuration: *failure probability*, *maximal speed* and *number of channels* are now considered as abstract resources; it is the aim to find an optimal resource configuration given a fixed amount of abstract currency. Therefore, we now use SIFIRA agents with resource-adapting functionalities to represent servers and to perform local optimizations for these abstract resources. These agents control an instance variable *abstractCurrency* which stores the amount of assigned credits.

We also introduce a higher level SIFIRA *monitor agent* for the global optimization view. This agent adapts the system due to the current state on the societal level: It tries to optimize the allocation of *macro-level abstract resources*, such as the *overall number of server agents*, their (logical) positions, the *minimum distance of UA migration*, but also the distribution of *abstract currency* to the server agents.

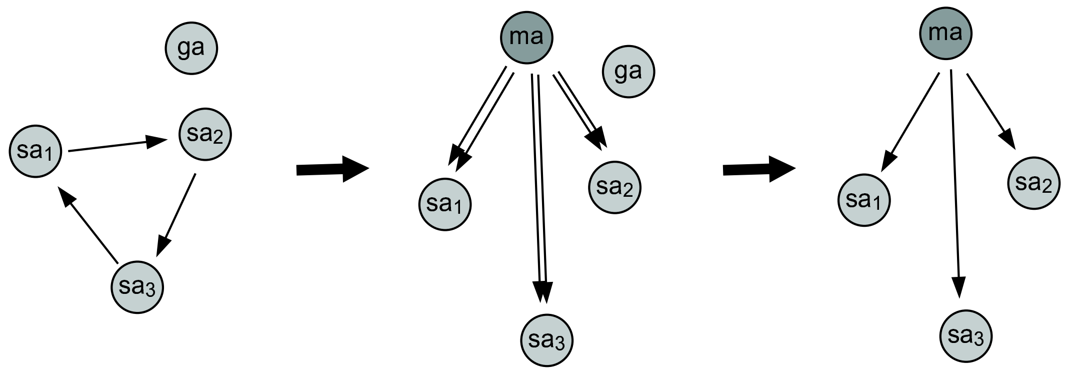


Figure 6.7: Building up a hierarchy

6.4.2 Implementing the Optimization Procedure

Construction of the Hierarchy

In the very beginning, we introduce a server agent sa_i for the local optimization of each server. Additionally, there is an agent ga for global optimization. According to the methodology of Section 3.5, the management of abstract currency for the various servers constrains the behavior of all server agents sa_i, inducing a cyclic dependency between the server agents. The left-hand side of Figure 6.7 depicts the initial situation.

According to Algorithm 5, the cycles can be replaced by the introduction of a new monitor agent and the creation of new dependencies from this agent to the server agents (middle part of Figure 6.7). In Section 3.5 we have presented this algorithm as a guideline for the design of a hierarchical agent society. The scenario designer then can use his knowledge of the domain to simplify the graph, which in our rather simple case leads to a merge of the various arcs from the monitor agent to each server agent. Furthermore, the functionality of the ga agent can be performed by the monitor agent, making ga redundant (right part of Figure 6.7).

The RIL Scheme

According to the GRAIL paradigm, each server agent receives *guidelines* from the monitor agent and reports on its current state as a *local profile*. In this scenario, guidelines are implemented through the abstract currency distributed by the monitor agent to the members. A local profile of a member agent contains the number of communication acts of MoTiV-PTA agents logged on that server, the number of connections of MoTiV-PTA agents to that server, and the number

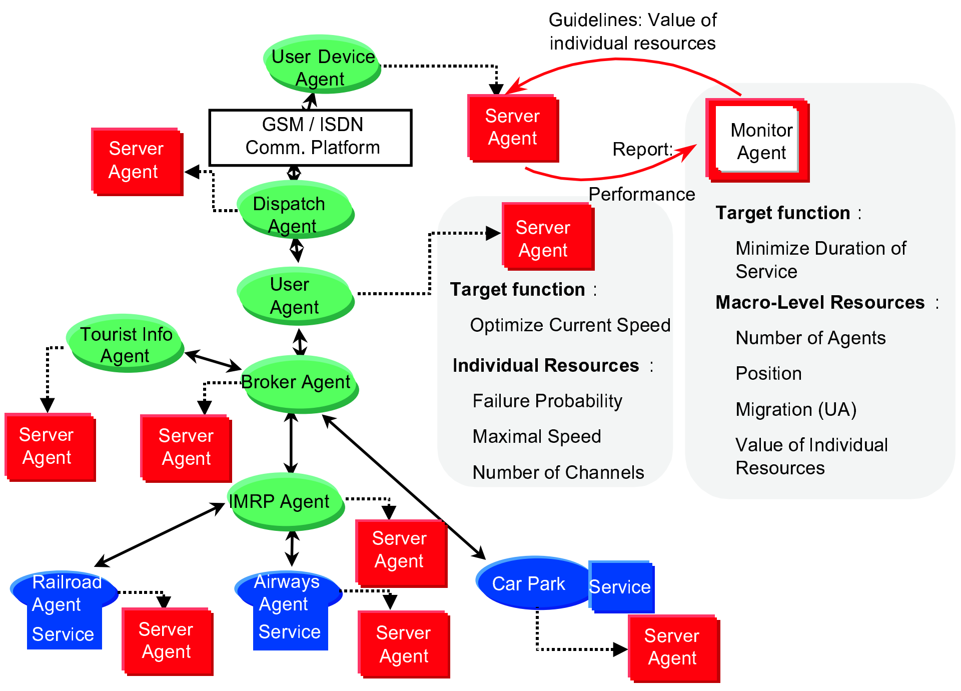


Figure 6.8: Server optimization in the MoTiV-PTA domain

of services running on that server. A server agent optimizes its configuration according to the average run time per service performed on this server. All these optimization processes run in parallel in order to provide a coherent adaptation of the whole system to changes in the user behavior.

Each SIFIRA server agent maintains the same properties as a SIF server agent. Now however, agents take actively part in the optimization process. Figure 6.8 shows the scenario which extends the one of Figure 6.1. Hence, server agents extend SIFIRA member agents by the simulation functionalities, as shown in Figure 4.8 in Section 4.2. In this scenario, the monitor agent has no additional domain oriented functionalities; hence it can be taken directly from the SIFIRA library.

The RIL scheme is performed using the SIFIRA constructs presented in Section 4.2: *societyObserverActions*, *AnswerRequestActions* and *SocietyModifierActions* are emitted and perceived.

The Micro-Level RAL

At the micro-level of each server, the abstract currency assigned by the monitor agent has to be spent for the following abstract resources:

- **Maximum speed:** We define an abstract speed measure with a range between 0 and 400 on an integer-valued and cardinal scale.
- **Number of channels** (Range between 0 and 50 on an integer-valued and cardinal scale)
- **Failure probability** (Range between 0 and 1 on a continuous and cardinal scale).

The amount of abstract currency (AC) is computed according to the following formula:

$$\text{AC} = [\frac{maxSpeed}{500} + \frac{numberOfChannels}{50} + (1 - 4 * failureProbability)] * 100$$

The standard step size for shifting AC credits form one micro-level resource to another is 5 AC units. The objective function is derived from the throughput of the server: it is desired to achieve per time unit a high number of communication acts of MoTiV-PTA agents that are located on the servers. A low number may have two reasons: either the server is too idle in the sense that there are too few MoTiV-PTA agents logged on, hence hardware resources are wasted; or the server is too overloaded in the sense that the communication activity of the MoTiV-PTA agents logged on the server is delayed. Both cases are disadvantageous. Therfore we measure a server's performance by *number of communication acts of logged-on MoTiV-PTA agents per minute.* On this problem, Greedy-RAL is executed.

The Macro-Level RAL

The following resources are implemented for the macro-level:

- **Number of servers** (Range between 1 and 100 on an integer-valued and cardinal scale and a standard step size of 1),
- **Right to change the minimum migration distance:** The distance between to servers on the grid can be at most 30 units (on an integer-valued and cardinal scale and a standard step size of 30),
- **Abstract currency:** Servers may have between 30 and 300 credits at their disposal. We model a modification of that resource as an allocation shift from one server to another. This is based on the assumption, that it is always reasonable for a server agent as well as for the entire society to spend all of its abstract currency and hence an increase of spent resources must be compensated by an appropriate decrease elsewhere.

The overall system performance is measured by averaging the *duration time of a representative variety of MoTiV-PTA services over a time period.* Changing the number of servers of course influences the server topology on a map. Modifying these resources hence amounts to changing the server structure. We use the following heuristics to decide where to place an additional server or which server to shut down.

Putting up a new server: Whenever the monitor agent decides to increase the number of servers during the run of a macro-level RAL, it has to be determined where exactly to place this server. There are two sometimes conflicting criteria:

- *Distance between servers:* In order to minimize communication delay, the server topology should provide a uniform density, i.e., servers should be spread with almost equal distances over the map, considering the shaping of the boundaries of the map.
- *Work load of servers:* Overloaded workstations need support, hence it is reasonable to place new workstations closer to stations that are bottlenecks in the sense of Section 3.4.

Strategies for finding optimal solutions to this problem have been proposed in the field of OR (see for instance [JL85] or [DD96] for an overview). However, they run usually in an iterative way at rather high computational costs. For the purpose of demonstrating applicability of the optimization approach, we use a heuristic-based technique to find a position for a new server. It is not required to find the perfect position but a satisfying one without spending too much computational time and space.

The monitor agent maintains eight points for each server agents: Each point *candPoint* is the exact intermediate point between the position of a server and the boundary of the map according to the four chief points of the heavens and their intermediate directions NW, SW, SE, and NE. If bottlenecks are not considered, the position of a new server will be assigned to that point which maximizes the maximum distance to *all* existing servers. (This strategy is intended to meet the requirement of a relatively equally distributed topology.) In the example of Figure 6.9, a new server is placed in the Northeast of server 2. Finding such a server is performed in the background when a new server is introduced or a server is deleted.

In order to address the second requirement, the current status of each server has to be regarded. Hence, whenever a new RAL step is performed, the work load of the each server α_i is taken as a factor a_i by which the candidate position is narrowed to the server: servers with more than average MoTiV-PTA agents

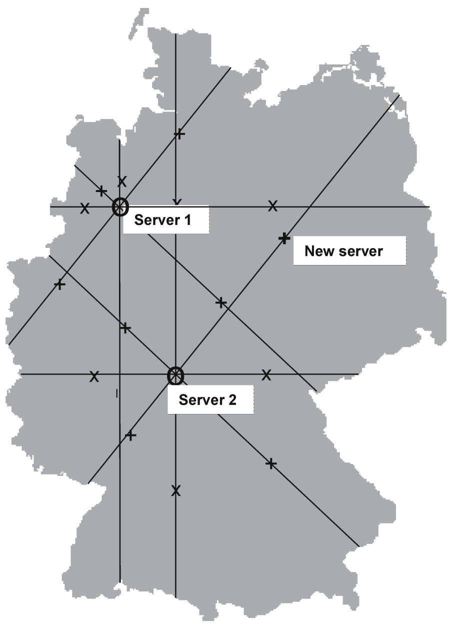


Figure 6.9: Finding a position for a new server

and less than average performance favor a closer position of the new server. The factor a_i is calculated according to the following function:

$$a_i = \min(1, \frac{performance(s_i)}{performance_{avg}} * \frac{|PTAAgentList_i|}{|PTAAgentList_{avg}|})$$

The *new* candidate position is assigned to $a * pos(s_i) + (1 - a) * pos(candPoint)$. Since the performance of a server changes during the run of the system, these factors are computed for all servers during at each RAL step; so is the determination of the one point that maximizes the maximum distance to a server in the system.

The new server agent of course needs abstract currency units for its local optimization. Since the overall number of abstract currency credits is limited, the new server agent earns 30 units that are removed from other agents in equal parts.

Shutting down a server: If the monitor agent decides during a RAL step to decrease the number of agents, a server to be closed down has to be found. As a simple heuristic we select the one server with minimal abstract currency credits, since this server is supposed to have only little contribution to the overall performance. Its abstract currency units are distributed equally to the other agents.

Before a server agent is terminated, all MoTiV-PTA agents that are currently logged on that server are forced to migrate. New servers are selected according to the previously stated trade-off between distance and server speed.

Integrating Heuristics from Bottleneck Analysis

As already described in Section 3.4, we can integrate knowledge about the internal state of a server agent into the RAL decision making process. Besides finding a location for a new server, server work load is also used to specify the shift of abstract currency: low performing servers (i.e., servers with relatively few communication acts of logged MoTiV-PTA agents) but with *many* logged MoTiV-PTA agents and services are most likely overloaded; hence shifting resources to such an agent is expected to increase not only its local performance, abut also global performance.

On the other hand, servers with few communication acts of logged MoTiV-PTA agents and *few* logged MoTiV-PTA agents and services are assumed to be too much idle: hence resources are taken from such an agent and assigned elsewhere. In this case, the performance of the agent will clearly further drop, but global performance can be expected to increase.[2]

In the heuristic-based Greedy-RAL for the MoTiV-PTA application the above observations are incorporated as follows: If the number of connected MoTiV-PTA agents that are connected to a certain server exceeds the doubled average number of MoTiV-PTA agents or services, then this server is kept in memory. In case several servers fulfill this requirement, the one with the highest resources is taken. If there exists another server which a) is less than half loaded than an average server and b) has minimal performance (if there exist several ones) then the former server receives credits from the latter. The amount of transferred credits is not determined by this heuristic, but by the main RAL procedure.

Similarly, if the number of connected MoTiV-PTA agents connected to a server is less than half of the average number of MoTiV-PTA agents or services, then this server looses credits from a server with a) a more than double of the average work load and b) has minimal performance (if there exist more than one).

If the amount of assigned credits of a certain server has dropped to the minimum value (which is set to 30 in this case) while the overall performance kept on increasing, that particular server is deleted and its remaining resources is distributed to other servers.

[2]This resource shift is possible since GRAIL member agents are benevolent to their monitor agent and accept guidelines, even if this reduces their local performance.

Integrating Heuristics from Machine Learning

In this scenario, machine learning techniques are integrated to find a promising direction and size for a step in the search space: a *memory base* module stores situations and actions which have lead to significant performance decreases or improvements. The key idea is to remember such a situation later in a new situation and to take the previously performed significant action as a suggestion for a future action.

We store situation-action tuples in hash table. To cut down the complexity of finding appropriately similar situations later on, we introduce *profiles* that are similarity classes of the situations. Since the number of possible profiles is still greater than the number of possible actions in this scenario, it is more efficient to store profile-action tuples by using the action as the key. Two profiles are merged to one, if an action has proven efficient in both situation classes.

Whenever a new step direction has to be determined, the current situation is matched against the memory base by parsing through all situation profiles: If a stored profile is found which includes the current situation, a corresponding action is performed if the former action had lead to a performance increase. If the stored action had decreased the performance, an action in the opposite direction is done.

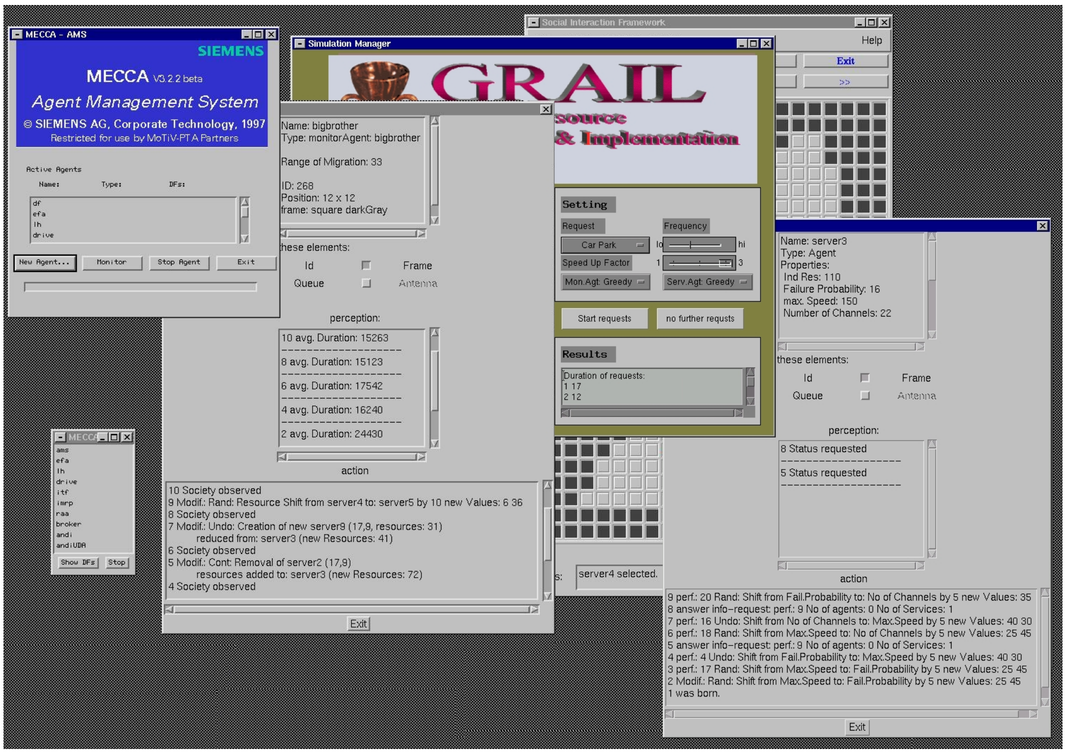
For long-term applications it is important to keep the memory base at a size that allows a fast situation matching. Since we assume a continuously changing environment, older experiences may be outdated and hence have to be removed from the memory base to keep it small enough. Hence we annotate each entry with a weight which is decreased over time. Whenever a profile has lead to some good advice, its weight is increased, otherwise it is even further decreased. Once the weight has reached 0, the corresponding profile is removed from the memory base.

One of the major advantages of this approach is that there is no explicit need for a training phase. However, this mechanism can only provide good advice if the memory base is adequately filled. Therefore, the system must have run for a while before good results can be expected.

6.4.3 The Extended User Interface and Script Parser

The SIF Object Info Window

This window displays the most recent actions and perceptions of an agent. For the monitor agent, the windows shows the perception of the society and the steps performed on the macro-level RAL; in case of a server agent, the object info



```
df
efa
lh
drive
```

```
Name: bigbrother
Type: monitorAgent: bigbrother

Range of Migration: 33

ID: 268
Position: 12 x 12
frame: square darkGray
```

```
10 avg. Duration: 15263
--------------------
8 avg. Duration: 15123
--------------------
6 avg. Duration: 17542
--------------------
4 avg. Duration: 16240
--------------------
2 avg. Duration: 24430
```

```
10 Society observed
9 Modif.: Rand: Resource Shift from server4 to: server5 by 10 new Values: 6 36
8 Society observed
7 Modif.: Undo: Creation of new server9 (17,9, resources: 31)
        reduced from: server3 (new Resources: 41)
6 Society observed
5 Modif.: Cont: Removal of server2 (17,9)
        resources added to: server3 (new Resources: 72)
4 Society observed
```

```
Duration of requests:
1 17
2 12
```

```
server4 selected.
```

```
Name: server3
Type: Agent
Properties:
 Ind Res: 110
 Failure Probability: 16
 max. Speed: 150
 Number of Channels: 22
```

```
8 Status requested
--------------------
5 Status requested
--------------------
```

```
9 perf.: 20 Rand: Shift from Fail.Probability to: No of Channels by 5 new Values: 35
8 answer info-request: perf.: 9 No of agents: 0 No of Services: 1
7 perf.: 16 Undo: Shift from No of Channels to: Max.Speed by 5 new Values: 40 30
6 perf.: 18 Rand: Shift from Max.Speed to: No of Channels by 5 new Values: 25 45
5 answer info-request: perf.: 9 No of agents: 0 No of Services: 1
4 perf.: 4 Undo: Shift from Fail.Probability to: Max.Speed by 5 new Values: 40 30
3 perf.: 17 Rand: Shift from Max.Speed to: Fail.Probability by 5 new Values: 25 45
2 Modif.: Rand: Shift from Max.Speed to: Fail.Probability by 5 new Values: 25 45
1 was born.
```

Figure 6.10: Screen shot of the entire system

window shows perceived information requests, as well as performed steps in the micro-level RALs, the answering of info requests and the realization of directives from the monitor agent.

Figure 6.10 shows a screen shot of the complete system: The MECCA agents DF and AMS are depicted on the very left; the figure also displays the scenario GUI and the simulation manager as well as the object info windows of the monitor agent (third window on the left) and one server agent (the very right window).

Additional Control Features of the Simulation Manager

For optimization purposes the simulation manager offers facilities to modify the type of RAL while running the optimization. For macro-level optimization, the experimenter can select between no optimization, Greedy-RAL, Greedy-RAL with bottleneck analysis optimization, Greedy-RAL with integrated bottleneck analysis, and Greedy-RAL with both machine learning and bottleneck analysis optimizations. On the micro-level he can choose between Greedy-RAL and no optimization.

Extensions of the SIF Script Parser

For running the optimization, the simulation script parser needs only to be extended to allow the introduction of a monitor agent (Server agents are automatically equipped with server agent facilities, since they now extend the class *MemberAgent*):

`MonitorAgent` *<name>, <xPos>, <yPos>* A script entry beginning with the keyword `MonitorAgent` leads to the creation of a SIFIRA monitor agent named *<name>*, located in the map at position [*<xPos>;<yPos>*]. Although this agent does not correspond to any real-world entity, we place it on an unused point of the map for visibility reasons.

6.5 Evaluation

In the following we present an empirical evaluation of the GRAIL approach in the MoTiV-PTA domain. Our focus lies on the demonstration of the feasibility and usefulness of GRAIL: we compare the throughput of the system when running the different GRAIL optimizations in various environmental constellations. The overall throughput of the system is well described by the average duration of query processing; thus we use this duration as the target function to be minimized in the optimization problem. Of course, these values depend heavily on the simulation parameters and therefore can hardly be use to draw *direct* conclusions to a real-world scenario. However, the *relative* performance differences of the optimization procedures can be transfered to make optimality statements.

6.5.1 The Analyzed Problem Classes

We compare the system's behavior for the ten combinations of implemented optimization procedures on the macro- and micro-level. In terms of spatial distribution of the servers, we employ a setting of UDAs located on five servers roughly equally spread over Germany. We distinguish between two distribution forms: First, requests are emitted equally often from each of these servers and second, requests are emitted from one server in 40% of the cases, and in 15% of the cases from another server, leading to a more irregular spatial distribution.

We also vary the types of requests: For one form of experiments we assume an equal distribution of all three type of requests, while for another form we take 60% travel information requests and 20% for the other two type of requests.

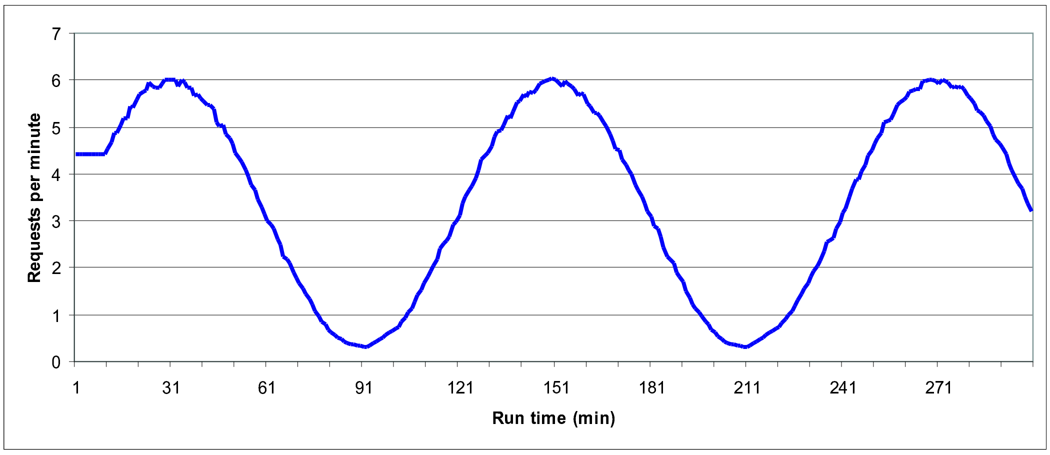


Figure 6.11: Temporal distribution of work load in the on-line case

We distinguish between two different *situations*: In the off-line scenario, GRAIL is used to configure a society of servers prior to the actual use of the system. During the run of the system, GRAIL is turned off. This situation can be classified as *resource-adapted behavior* (see Section 3.2). For our experiments we adjust the system work load to three requests per minute with a 3% variance. As stated above, for each of the ten combinations of optimization, we run two variations concerning spatial distributions and two variations for the type of requests, leading to 40 experiments, all of which are run for 60 minutes.

In the on-line case, GRAIL is applied while the system is actually running to allow an adaptation to changes in the environment. This case is called *resource-adapting behavior*. Since the macro-level optimization procedure continuously modifies the environment of the lower-ranked optimization procedure, the micro-level optimization can always be classified as resource-adapting. In this case we run a a setting where maximum work load is set to six requests per minute, minimum work load to 0.5 requests per minute, the period duration to two hours at a 3% variance. Figure 6.11 shows the temporal variation in the work load of the system. Considering all combinations, we again run 40 experiments, which however ran for five hours.

For the application of the machine learning approach, we have run all scenarios five times in order to collect training data. The exact data of the results can be found in Appendix B.1.

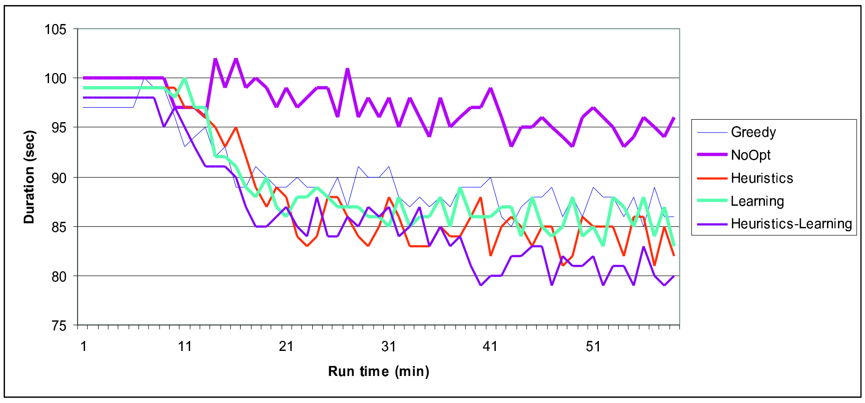


Figure 6.12: Different macro-level optimization approaches in the off-line case

6.5.2 Results

Off-line Optimization

For the off-line comparison mainly two aspects are of concern:

- Which combination of optimization mechanisms behaves best, i.e., leads eventually to the highest overall performance?
- Which combination of adapts fastest, i.e., reaches a sufficiently high value in reasonable time?

Figure 6.12 compares the different macro-level optimization approaches, while all other parameters are kept fixed: the micro-level optimization is turned on, the spatial origins and the type of the requests are distributed equally. In this situation, the processing of a request initially takes about 100 seconds and reduces to 95 seconds, if no macro-level optimization is applied. Using an optimization scheme reduces the duration by 10-15 %. Obviously, the simple Greedy-RAL performs not as well as the more complex RALs: The machine learning extension takes some time, but then shows a very steep descend in duration. The heuristic-based approach reduces the processing duration slower, but reaches roughly the same level. A combination of both extensions performs best in both, convergence speed, and performance level.

Applying the *micro-level* optimization reduces the average duration of answering a request by 5%, independent from the choice of the macro-level optimization at a rather slow convergence rate. As an example, the left side of Figure 6.13 compares the combinations of using Greedy-RAL on macro-and micro-level.

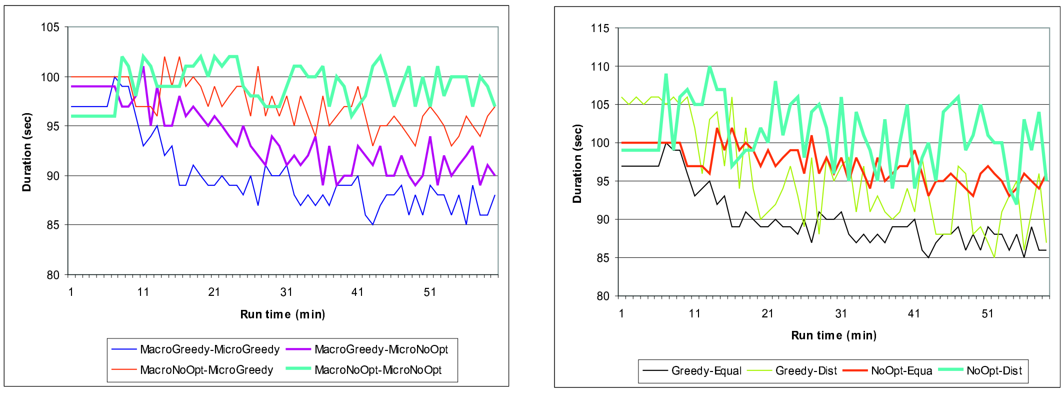


Figure 6.13: Micro-level optimization and spatial distribution in the off-line case

Modifications on the *type of requests* does not lead to significant performance changes, hence we do not provide a chart; a variation of the *spatial distribution* influences the absolute value and mainly, the variance of the result (see the right-hand side of Figure 6.13); however the above rankings on the performance quality remain valid.

On-line Optimization

In the on-line case we are interested in similar issues:

- Which combination of optimization mechanisms leads over the total running time to the best average performance?
- Which combination adapts fastest to environmental changes? This is in particular relevant when the environment changes rapidly.

The experiments show that in this simulation environment the duration of query processing depends heavily on the work load of the system: in heavy-duty times, the processing may take up to two minutes while in rather idle situations the process takes only about 50 seconds.

Figure 6.14 shows for cases of equal distributions of spatial original and type of requests and turned-on micro-level RAL, the quality ranking of the mechanisms is the same as in the off-line case, however, there is almost no performance difference recognizable for lower-duty times, while the performance difference is rather striking in heavy-duty cases.

In these cases, approaches that include a learning component, show a flatter rise to the peak and a steeper decrease after the peak. However there is a notable

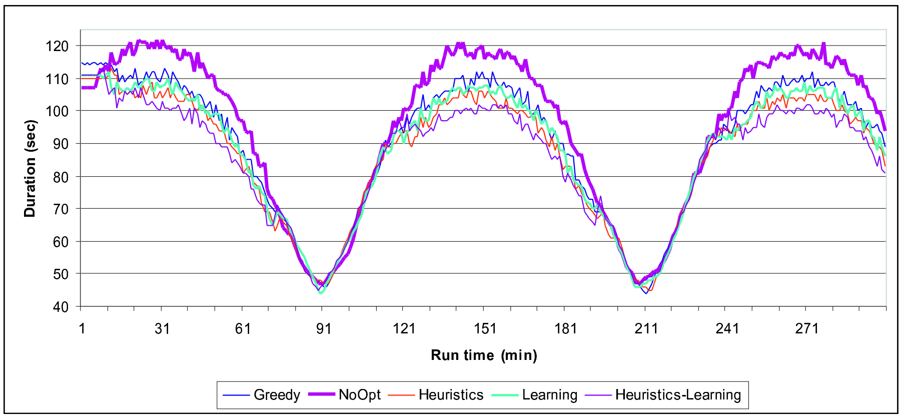


Figure 6.14: Different macro-level optimization approaches in the on-line case

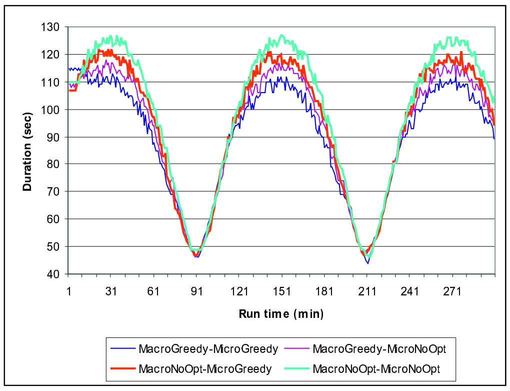
point in the period (in this case after about 70 and again after 195 seconds) where the processing duration suddenly jumps up before it decreases again. This phenomenon may be due to the fact that the environment changes rather rapidly at this time so that the learning component cannot make use of the learned strategies.

Just as in the off-line case, the use of a micro-level optimization reduces the processing duration independent from the chosen macro-level optimization. The left part of Figure 6.15 shows a comparison of the combinations of using Greedy-RAL at the macro- and micro-level. Running these experiments on different distribution of spatial origin and type of requests, does not change the principle results (as the right part of Figure 6.15 shows for different spatial distributions).

Bottom Line

In this chapter we have implemented the theoretical resource adaptation in agent societies of Chapter 3 by employing the SIFIRA tool set for resource-oriented agent programming of Chapter 4: the MoTiV-PTA case study employs the previously presented techniques: GRAIL runs in an integrated way on both, the macro- and the micro-level. On the individual level, every server is optimized according to its private resources, while on the societal level the entire server topology is optimized. GRAIL makes use of the presented SIFIRA development kit by employing the provided functionalities, such as the different RAL mechanisms and the RIL scheme.

We have evaluated our approach. Results underline the applicability and usefulness of the approach: especially for heavy-duty situations the total performance can be improved by running the GRAIL approach. The results show



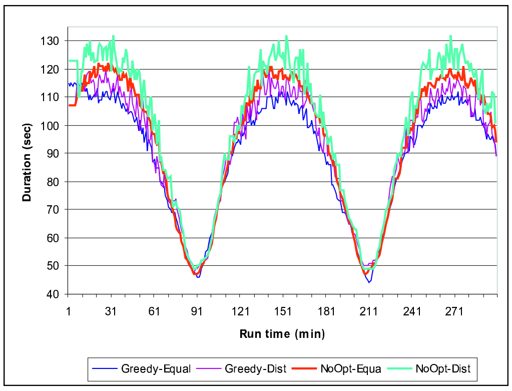


Figure 6.15: Micro-level optimization and spatial distribution in the on-line case

that for the RALs available in this case study, a combination of a heuristic-based approach and a machine learning approach performs best. However, the GRAIL architecture is designed so generically that new RALs can be integrated easily. Results also show that the combination of macro- and micro-level optimization leads to an additional performance gain, as it is supported by GRAIL.

Chapter 7

Case Studies: Two Holonic Agent Systems

In this chapter we present two industrial applications of holonic agent systems, the transport planning system TeleTruck and the job shop scheduling system IFMS. For both, we describe the domain, the design of the holonic agent structure, and the implementation of GRAIL. Furthermore, we evaluate both approaches.

7.1 The TeleTruck System

In contrast to the MoTiV-PTA and IFMS case studies, TeleTruck [BFV98a, BFV98b] is a pre-existing system which we extend in several ways to evaluate our resource adaptation scheme in a holonic domain. In the course of this section, we first describe the domain and the main components of the original system, then we show how existing problem solving strategies can be mapped onto the GRAIL scheme, and finally, we present and evaluate our own extensions.

7.1.1 The Domain

The increasing freight transport on European roads, partially caused by the European integration and the release of the iron curtain is leading to continously worsening traffic conditions. Many interstate routes are overloaded while additional delays such as accidents and road constructions result in more and more unpredictable transportation times. On the other hand, *just in time* delivery of goods is becoming more popular than ever in modern supply chain management, making a dynamic and flexible delivery inevitable.

In order to survive on the market, haulage companies must be able to offer a very flexible shipment, but they also must make as full use of their capacities as possible in order to cut down costs. Two recent trends are recognizable: first, nowadays small companies can hardly exist on the market, so they are forced to cooperate by forming *networks* of similarly interested forwarders that share a common infrastructure that helps to reduce costs. Second, there is also a tendency for larger enterprises to further grow by integrating out-sourced capacities.

All these effects lead to a grow in the complexity of freight allocation and tour planning making a manual processing almost impossible if high quality schedules have to be derived with short planning time. TELETRUCK is an agent-based fleet scheduling system designed to meet these challenges: It supports the dispatch officer of a shipping company or a network of independent forwarders in the scheduling of transportation tasks which may come in at any time.

Details of the Domain

Basically we can distinguish between different settings: In a *competitive* scenario, companies of small and medium size operate locally. In order to survive on the market, they are often forced to form temporary inter-regional cooperative networks by uniting their resources. Usually, such partners are self-interested, i.e. they prefer optimizing their private profit over increasing the overall profit of the network. So, in this scenario the *free-rider problem* occurs where participants might try to take advantage by betraying each other. Such an untruthful behavior would reduce the market opportunities of the network. In *cooperative scenarios* e.g., where the participating forwarders are subsidiaries of one company, the problem of selfishness is less dramatic. In this case mechanisms that allow global cost minimization can be applied.

In either case, the company or network has a limited set of vehicles at its disposal. The overall goal of the company or network is to gain as much profit as possible, i.e., to accept as many promising offers as possible due to the restriction of capacity. Furthermore, an efficient distribution of the load as well as efficient routes for each vehicle have to be found. The original TELETRUCK system has been developed only for the latter case. In this chapter however, we extend the scenario by considering both situations.

7.1.2 Overall System Architecture

The TELETRUCK system consists the following basic components:

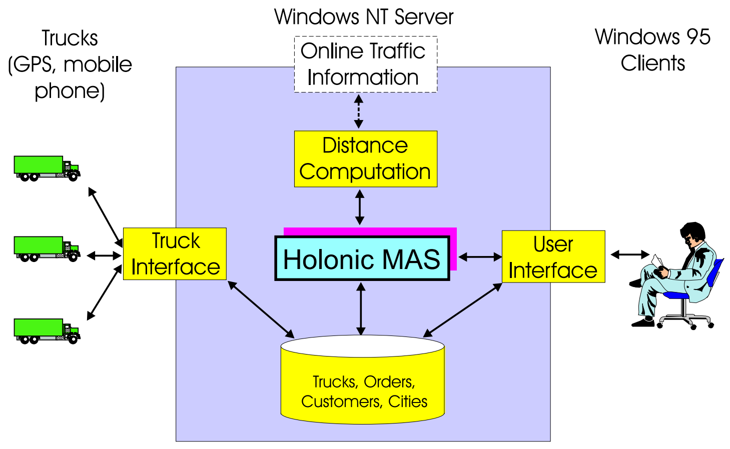


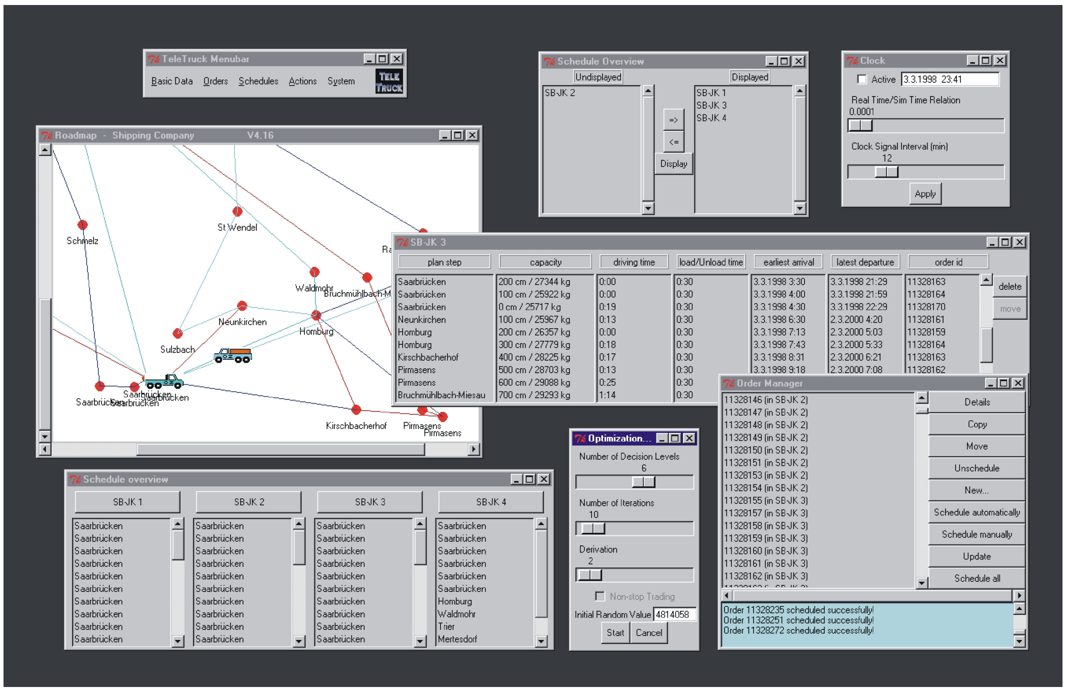
Figure 7.1: The TeleTruck architecture

Configuration and tour planner: The central component of TeleTruck is a holonic multi-agent system that manages the planning and optimization of the vehicle configuration and of the tour plans.

User interface: The user can interact with the planning system in several ways: he can insert new customer orders to the system, he can manually generate tour plans or trigger an automated planning of the multi-agent system. The user can also modify the generated plans or initiate an automated optimization procedure of the previously found solution. Furthermore, he can insert additional constraints, for example to book an order to a vehicle or a time slot.

Data base: Both, user and the central planning component have direct access to all data stored in a common SQL-database. Electronic maps and routing software are added to supply the tour planning system with geographical data.

Figure 7.1 shows the system architecture of TeleTruck while Figure 7.2 displays a screen shot of the system. A detailed system description can be found in [Vie00]. So far, we have described the architecture of the original TeleTruck system. In the next section, we show how TeleTruck can canonically be mapped onto the GRAIL scheme, although TeleTruck has initially not been designed using the scheme. Furthermore, we show how different resource allocation algorithms can be integrated into the system, which is a clear extension of the system as it is described in [BFV98a, BFV98b]. As a second extension we introduce a further structurization in the task allocation procedure by taking geographical dispersion of the fleet and the tasks into account. Both extensions are evaluated in Section 7.2.



Schedule Overview — Undisplayed: SB-JK 2; Displayed: SB-JK 1, SB-JK 3, SB-JK 4

Clock — Active 3.3.1998 23:41
Real Time/Sim Time Relation 0.0001
Clock Signal Interval (min) 12

SB-JK 3

plan step	capacity	driving time	load/Unload time	earliest arrival	latest departure	order id
Saarbrücken	200 cm / 27344 kg	0:00	0:30	3.3.1998 3:30	3.3.1998 21:29	11328163
Saarbrücken	100 cm / 25922 kg	0:00	0:30	3.3.1998 4:00	3.3.1998 21:59	11328164
Saarbrücken	0 cm / 25717 kg	0:19	0:30	3.3.1998 4:30	3.3.1998 22:29	11328170
Neunkirchen	100 cm / 25967 kg	0:13	0:30	3.3.1998 6:30	2.3.2000 4:20	11328161
Homburg	200 cm / 26357 kg	0:00	0:30	3.3.1998 7:13	2.3.2000 5:03	11328159
Homburg	300 cm / 27779 kg	0:18	0:30	3.3.1998 7:43	2.3.2000 5:33	11328164
Kirschbacherhof	400 cm / 28225 kg	0:17	0:30	3.3.1998 8:31	2.3.2000 6:21	11328163
Pirmasens	500 cm / 28703 kg	0:13	0:30	3.3.1998 9:18	2.3.2000 7:08	11328162
Pirmasens	600 cm / 29088 kg	0:25	0:30			
Bruchmühlbach-Miesau	700 cm / 29293 kg	1:14	0:30			

Order Manager

11328146 (in SB-JK 2)
11328147 (in SB-JK 2)
11328148 (in SB-JK 2)
11328149 (in SB-JK 2)
11328150 (in SB-JK 2)
11328151 (in SB-JK 2)
11328153 (in SB-JK 2)
11328154 (in SB-JK 2)
11328155 (in SB-JK 3)
11328157 (in SB-JK 3)
11328158 (in SB-JK 3)
11328159 (in SB-JK 3)
11328160 (in SB-JK 3)
11328161 (in SB-JK 3)
11328162 (in SB-JK 3)

```
Order 11328235 scheduled successfully!
Order 11328251 scheduled successfully!
Order 11328272 scheduled successfully!
```

Schedule overview

SB-JK 1	SB-JK 2	SB-JK 3	SB-JK 4
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Saarbrücken
Saarbrücken	Saarbrücken	Saarbrücken	Homburg
Saarbrücken	Saarbrücken	Saarbrücken	Waldmohr
Saarbrücken	Saarbrücken	Saarbrücken	Trier
Saarbrücken	Saarbrücken	Saarbrücken	Mertesdorf

Optimization...
Number of Decision Levels 6
Number of Iterations 10
Derivation 2
☐ Non-stop Trading
Initial Random Value 4814058

Figure 7.2: Screen shot of the TeleTruck system

7.1.3 Matching TeleTruck to the GRAIL scheme

Traditional fleet scheduling approaches use techniques developed in operations research, such as minimum cost flow algorithms (A survey can be found in [AMO93]), where a central decision making unit plans the entire schedule. However, such centralized techniques have some severe drawbacks: Once a plan has been generated, the algorithm is finished. The plan execution is usually not monitored by the planning approach. Hence, if a step of the plan fails due breakdowns of vehicles or traffic jams, or if some new task is incorporated during the execution phase, the planning procedure has to be started again in order to perform a re-calculation.

An agent oriented approach to model this scenario does not only provide plan generation, but also monitoring of plan execution: The execution of all partial plans is monitored a decentralized way by *agents*. If a step of the plan fails or if some new task is incorporated during the execution phase, a decentralized re-planning procedure is started on-line to adjust current plans to the new situation. In such a case, only a minor part of the plan is modified by the corresponding agents while the execution of remaining parts still goes on.

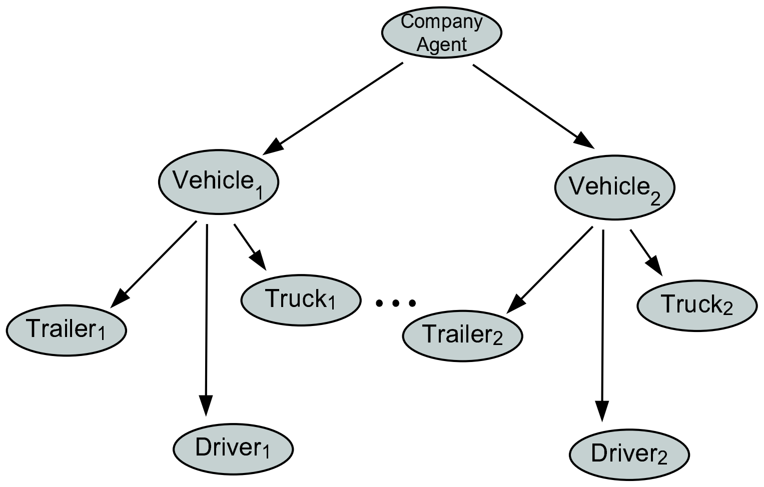


Figure 7.3: Agent society structure in TeleTruck

Construction of the Holonic Society

We show now the accordance of the design of the TeleTruck agent society with our GRAIL scheme. First, we present the structure as a result of the hierarchy construction scheme of Section 3.5, then we compare the result with the solution implemented in TeleTruck.

As already stated in Section 5.5, the following abstract resources express interdependencies among components of a vehicle: the daily *driving time* of the driver, the *loading space* of vehicles, the *chassis* and the *motor*. Therefore, we model each component by a *component agent*. The allocation of these resources constrains the behavior of any component leading to cyclic dependencies among the component agents.

According to our terminology, a *representative agent* is used to manage these dependencies, to represent the whole vehicle and to set up route plans. *Transportation tasks* as macro-level resources induce cyclic dependencies among the various vehicles. The company (in a cooperative setting) or the network (in a competitive setting) as the highest entity in this domain constrains the behavior of the vehicle entities by granting or rejecting tasks to vehicles. Hence, to our theory, a new representative agent is used to model the company or network.

Applying the graph transformation algorithm (Algorithm 5 of Section 3.5) on this situation and pruning away redundancies lead to the holon topology as displayed in Figure 7.3. (The initial dependency graph has already been shown in Section 5.5.)

Comparison to the TeleTruck implementation In TeleTruck, each transportation unit of the forwarding company is represented by a *component agent* that has private plans, goals, and communication facilities in order to actively contribute to the overall solution for the transportation plans and to administer its resources. The agents can merge with a *Planning'n'Execution Unit* (PnEU) and form a holon that represents a complete vehicle.

A PnEU plans routes, loading stops and driving times of the associated vehicle. Being the head of the vehicle holon, it represents the holon to the outside world, and is authorized to reconfigure it. A PnEU is equipped with planning, coordination, and communication abilities, but does not have physical resources on its own.

The original TeleTruck system has been designed for only a cooperative setting, where benevolence of the vehicle representatives can be assumed and hence, they can be integrated into an overall structure. Here, the vehicles are sub-holons of the super-holon that represents the entire transportation company. This holon is headed by a *company agent*, which announces and distributes the incoming orders, accepts the tenders, controls global optimization, and handles all communication with the user, i.e. the dispatch officer. It also coordinates the internal cooperation and interaction among the PnEUs. Thus, the implementation design of TeleTruck is in exact accordance to our scheme.

The RIL Scheme

This scenario requires a two-fold optimization scheme: on the macro-level the allocation of tasks to vehicles and on the micro-level the concrete vehicle formation. In contrast to the cases of the MoTiV-PTA and IFMS scenarios, these two aspects interact significantly: a vehicle holon is built *during* the macro-level negotiation; the constraints of this order impose guidelines to the holon formation.

Whenever a new customer order is entered, a task allocation procedure is triggered in order to determine which vehicle performs the additional task, which is to our understanding a *macro-level RAL*. In the original TeleTruck system, the *contract net protocol* and the *simulated trading* procedures are run for this task since a cooperative setting has been assumed. In this allocation procedure, vehicle agents give bids that base on possible vehicle configurations which have to be determined *during* the current run of a macro-level RAL. This configuration determination scheme can be classified as a *micro-level RAL*.

Micro-level RAL for Dynamic Holon Formation and Reconfiguration

For the scope of this thesis we do not extend the allocation scheme on the micro-level; therefore we briefly describe its functionality and show its corelation to

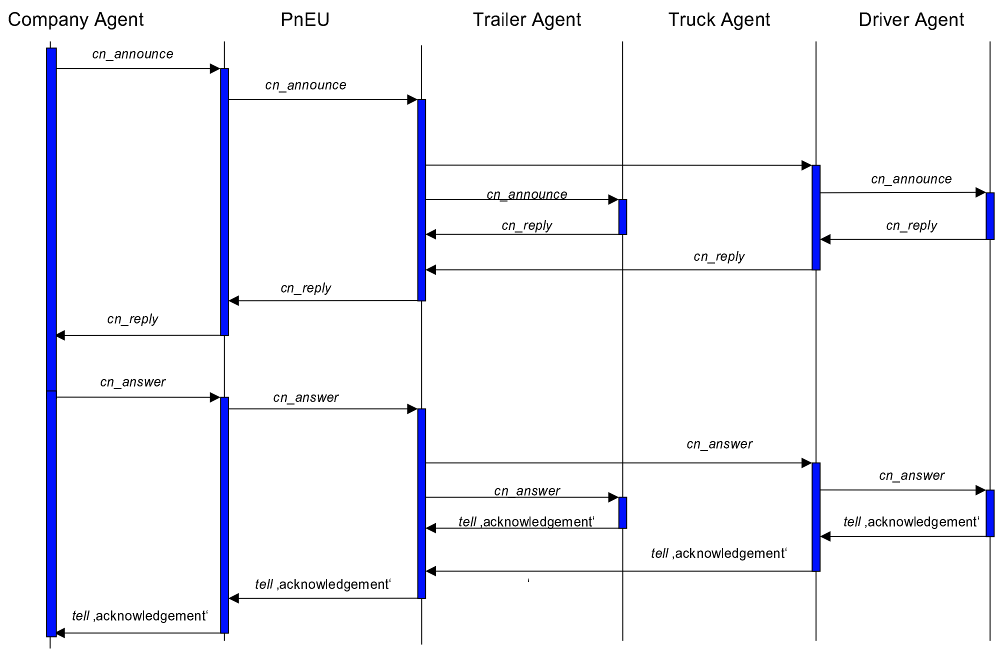


Figure 7.4: The hierarchical contract net protocol

the GRAIL scheme. During the task allocation to the various vehicles (i.e., the macro-level RAL, see below), resource allocation algorithms are performed at the stages of the vehicle components in order to decide on the allocation of the above resources and hence the configuration of the vehicles. In TELETRUCK, a recursive version of the contract net protocol is used to generate the detailed holon configuration and to allocate tasks to these holons (Figure 7.4). According to our scheme this contract net protocol implements the micro-level RAL.

If a task is proposed to a vehicle holon, its PnEU checks whether the resources of its current components are sufficient for the execution of the task. If so, it computes the cost of the execution and submits an appropriate bid to the company or the network representative. Consider the example of Figure 7.5: a company agent announces a new transportation task to two vehicle holons and the idle PnEU on the right. The already completed holon on the left hand side cannot incorporate further sub-holons, hence this PnEU does not perform a micro-level RAL, but sends an offer to the company agent with its valuation based on its current state.

If the resources of the PnEU are not sufficient, a configuration procedure of that sub-holon is initiated by running a contract net protocol, requesting for missing parts: a trailer has to supply loading space and the chassis. To do so, its corresponding agent needs a motor resource and hence announces the task to the set of idle vehicles. Vehicle agents may receive different announcements for

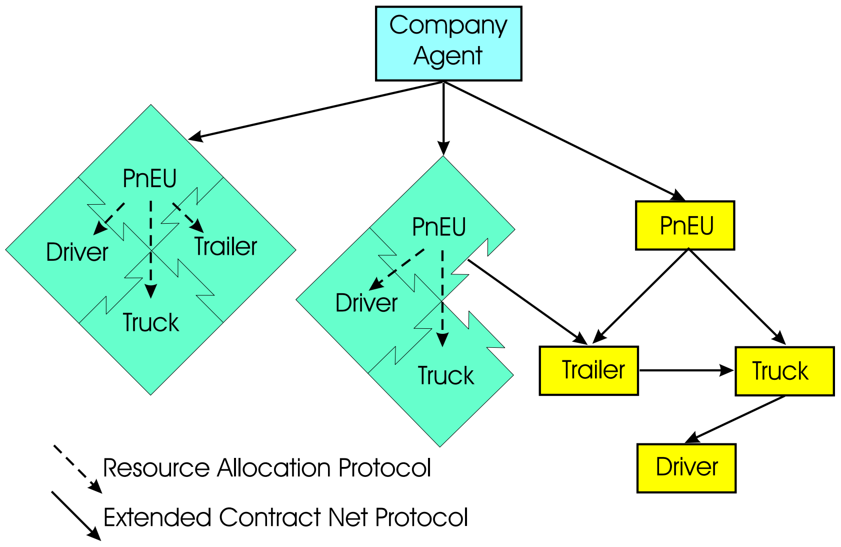


Figure 7.5: Holonic agents in TELETRUCK

the same task. In such a case, a vehicle agent can bid in any protocol since it can be sure that only one of the protocols will be successful. Therefore, the vehicle agent seeks for a driver, computes the cost for the different announcements, and returns a bid (in the example to both the PnEU and the trailer). Whenever the plan of a holon is finalized and executed, the components separate and the PnEU terminates.

The second vehicle holon in the middle of Figure 7.5 can additionally integrate a trailer. Hence, if its own resources are not sufficient for the new task, the head tries to collect the missing resources by performing a new contract net protocol with the idle trailers that supply the resources in need. The idle PnEU on the right first has to perform a protocol with those idle components that offer loading space; in the example a vehicle and a trailer.

7.1.4 Extensions of the TELETRUCK System

Whereas in the previous paragraph we have only described the functionality of the existing TELETRUCK system, we now extend the original system.

A First Extension: New Macro-level RALs for Task Allocation

As already stated above, in the original TELETRUCK system contract net and simulated trading are run in order to assign tasks to the concurrently forming vehicles. In our view these mechanisms are macro-level RALs. In this work, we extend the original TeleTruck design by employing the market-based RAL mechanisms *Vickrey auction* and *matrix auction*. (An overview of different RAL approaches has already been given in Section 3.4, a detailed classification of the different applied approaches can be found in [FRV98].) In the evaluation, we compare them to the simulated trading scheme.

Vickrey Auction: In the *sealed-bid second-price* or *Vickrey auction* every bidder submits a sealed bid for a single item in question. The bidder with the highest bid is assigned to the item, but in contrast to the *sealed-bid first-price* auction, the bidder who submitted the best bid receives the item for the price of the *second highest* bid.

This technique leads to the fact that a bidder whose bid exceeds his true valuation risks to be granted for this item at a price that exceeds the valuation as well. On the other hand, stating a bid lower than the true valuation might cause a rejection. In both cases a bidder cannot influence the price he has to pay. Hence, in order to get only grants that correspond to the own valuation, a rational bidder will submit accurate bids. This property is called *incentive-compatible* (see [MMT88]).

Matrix Auctions: This type of auctions has been designed for the simultaneous assignment of multiple items or tasks to bidders (see [GSW98]). Using these auctions promises to deliver more accurate results since the valuation of a set of items can differ significantly from the sum of the valuations of each single item if there exists some interdependency.

In a *matrix-k-auction* (MA-k) the auctioneer announces k items to the bidders that, in turn, calculate their valuations for each potential combination of items (hence, the bidders have to compute $2^k - 1$ valuations) and report them to the auctioneer. Using the reported valuations of the bidders the auctioneer sets up a matrix where the cells represent the bids for each combination of items. The auctioneer identifies the optimal allocation of all k items in the matrix; the price for each assigned subset of items equals the second-highest bid in the matrix column for that set of items. This *Vickrey pricing* assures again that the bidders reveal their true valuations.

The Vickrey auction and the matrix auction mechanisms base on the *Vickrey principle*: they are incentive-compatible allocation mechanisms, hence they are well-suited to generate cost-efficient allocations in competitive settings where the interacting forwarders are self-interested, nevertheless they can also be used in cooperative scenarios.

For an application of these approaches in the TeleTruck domain, the company agent (or network agent) takes over the role of the auctioneer while the vehicle agents work as bidders.

Simulated Trading: In contrast to the previous mechanisms, the *simulated trading* algorithm is not a pure allocation mechanism but combines the allocation of new items with the optimization of the existing allocation (which initially can be empty). It is a market mechanism where the participating contractors optimize a task allocation by successively "selling" and "buying" tasks. Trading is performed in a sequence of rounds, each of which consists of a number of decision cycles. In each cycle, the participants submit one offer to sell or buy a task. At the end of each round the central coordinating instance matches the incoming sell and buy offers of the contractors.

Simulated trading is only suited for cooperative settings, in which complete information about the participants' valuations is available. Originally, simulated trading was designed to improve an initial allocation. For our purpose we have extended the simulated trading algorithm such that an existing allocation is not required but can be generated during the trading process.

In the TeleTruck scenario, the central coordination instance is the company agent, the participants are of course the vehicle agents.

During the run of a RAL step (i.e., the announcement of a new task) at this stage, RALs at lower ranked stages of the holon hierarchy are initiated in order to determine in detail which components will be used by the vehicle (see the previous subsection).

A Second Extension: Geographical Structurization

As a further extension of the original TeleTruck system and as an additional way to map inherent structures of the domain onto the holonic solution, we also regard the geographical dispersion of the orders: If there is a clustering recognizable, a heuristic to achieve a reduction in complexity of the planning procedure can be applied by partitioning the vehicle fleet according to the current location of the vehicles. In such a case, not all vehicle holons participate in the macro-level RAL, but only those that are situated in the same region as the sender and receiver of the order.

Hence the holonic structure is extended by one additional stage on top of the PnEU stage (see Figure 7.6). A *subsidiary agent* matches the geographical location of the incoming tasks with the ones of the fleet and announces tasks only to possibly suitable PnEUs, i.e., those that are located closely enough.

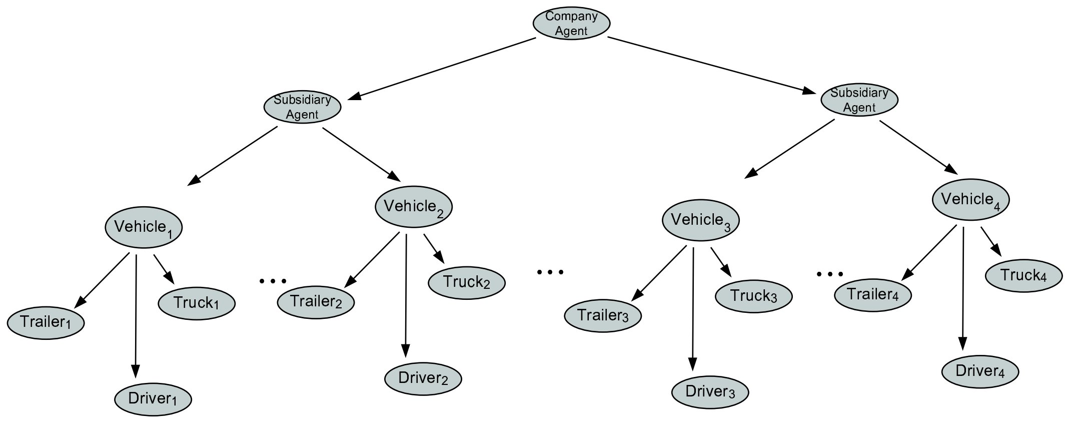


Figure 7.6: Extended holonic hierarchy in TELETRUCK

7.2 Evaluation of the Extensions to TELETRUCK

In this section, we evaluate our extensions of the TELETRUCK system: First we compare several macro-level RALs in different settings on a theoretical and an empirical basis. Then we examine the effects of the introduction of the above described subsidiary agent.

7.2.1 Theoretical Analysis of Different Macro-Level RALs

We investigate the communication complexities of the introduced market mechanisms. We examine how the complexity of *agent communication* depends on the *number n of agents* and the *number k of tasks to be allocated* in the system.

Estimating only *computational* complexity would be insufficient because the complexity of a computation an agent has to perform does not always have to effect the performance of another agent and the overall performance in a distributed system. *Agent communication* turns out to be a good performance indicator because in physically distributed domains, such as the transport domain, establishing and using communication channels can be very important limiting factors. As a measurement, we do not simply use the number of communication acts but the overall number of communication primitives an act consists of, since communication acts may vary in complexity.

We now discuss communication complexity of the *Vickrey auction*, *matrix auctions* and *simulated trading* protocols. For the sake of independence from the underlying computational model we assume that agent communication is only possible in a point-to-point fashion. Hence, in this model, broadcast communication can only be realized by sequentially sending messages to communication partners. Assuming that the effort for sending messages equals the effort for receiving them, the possibility of broadcasting messages reduces the total effort for communication at most with the constant factor 2, since the effort for receiving broadcasts remains.

The Vickrey Auction: During a Vickrey auction the following communication acts are sent: an auctioneer sends bid requests for a certain good or order to all bidders who reply with their bids. Then, the manager selects an appropriate partner, confirms the assignment of the order to this partner and sends rejects to all other bidders. Let n be the number of communicating agents. According to the point-to-point communication assumption, *(n-1)* requests (each of which consisting of one communication primitive) are made, followed by *(n-1)* bids, one confirmation and *(n-2)* rejects, all consisting of one communication primitive. Hence, the Vickrey auction has communication complexity of *O(n)* in terms of the number of participating agents.

The Matrix Auctions: In the matrix auction the auctioneer announces k items to $n-1$ agents $(O(k \cdot n))$. The bidders submit one offer for each of the $2^k - 1$ subsets of the item set which corresponds to one communication act, consisting of $2^k - 1$ primitives $(O(2^k \cdot n))$. After computing the optimal allocation, the auctioneer informs the bidders about the final allocation $(O(n))$. This leads to an overall communication complexity of $O(2^k \cdot n)$. The exponential computational complexity enforces a small k. After having fixed a sufficiently small k, a communication complexity of $O(n)$ remains with a possibly high constant depending on k.

Simulated Trading: A trading round within the simulated trading process consists of l decision levels. At each level, every contractor may announce a selling request or place a buying bid to the central instance. The central instance has to inform $n-1$ contractors of received selling requests. In the buying/selling announcement phase of each level, *O(n)* communication acts are performed. In the information phase, the stock manager has to send at most *O(n)* messages. Such a message may contain at most *O(n)* offers, all of which are communication primitives. Hence, the complexity of one trading round is $O(l \cdot n^2)$. Since l is fixed in advance, and, hence, can be treated as constant, the communication complexity for simulated trading is $O(n^2)$. The central instance's task to process the trading graph (which is known to be NP-hard) does not influence communication complexity.

Summary

From a theoretical point of view, all protocols have linear or squared complexity in terms of communication primitives. Speaking in practical terms, if only few agents take part in one of the market-based mechanisms described above, no communication bottlenecks should arise.

However, if further processing is dependent on the final result of the negotiation, the measure of communication primitives is insufficient. For these reasons, we next provide an empirical scalability evaluation of the mechanisms where we examine the processing time and the overall allocative efficiency of the market-based mechanisms.

7.2.2 Empirical Evaluation of Different Macro-Level RALs

We analyze the performance of the matrix auctions (MA-k) where $k \in \{1, ..., 5\}$ orders are assigned simultaneously in comparison with the Vickrey auction and the simulated trading algorithm. Three major aspects are of concern:

Number of vehicles: Since the size of the vehicle fleet is an important cost factor for a shipping company, we investigate how many vehicles are needed, using the different market-based mechanisms.

Tractability: We have shown in the previous section that the complexity of the protocols in use is at the most squared in terms of number of communication primitives. However, the actual computing time is for some of the agents exponential. If other agents have to delay their actions until the task allocation process has been completed, processing time can be a critical issue for the usefulness of a mechanism.

Efficiency: Obviously, this issue depends on the chosen setting:

- **A cooperative company:** In this setting, the auctioneer agent represents the company and tries to minimize the overall *cost per order*. The bidders represented by vehicle agents have no interest in optimizing their individual profits.
- **A competitive situation:** The auctioneer (the network agent) does not optimize his profit and vehicle agents represent independent forwarders and optimize their *surplus per order*.

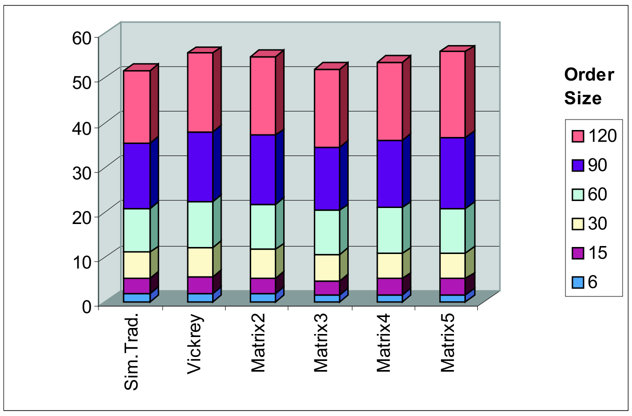


Figure 7.7: Number of granted vehicles

- **A hybrid scenario:** All, auctioneer and vehicle agents are rewarded. In such a case, the auctioneer tries to minimize the overall *payments per order* which is the sum of the costs and the forwarders' surplus, while the vehicle agents try to maximize their individual *surplus*.

The Analyzed Problem Classes

For our evaluation we use the benchmarks, Solomon [Sol87] generated for the vehicle routing problem with time windows. Those data build up on a set of problems that Christofides [CMT79] developed for the vehicle routing problem without time constraints.

Solomon's benchmarks include six different data sets of transportation orders that have distinct characteristics concerning geometry, number of destinations, and time constraints. In particular, we distinguish between *clustered* (several groups of clients lie closely together) and non-clustered settings. For our test runs we have averaged over six single problems[1]. We have chosen to analyze the system outcomes for maximally clustered test data (test set = {c 101, c 102, c 103}) and completely non-clustered test data (test set = {r 101, r 102, r 103}). In contrast to the original setting where the subordinate vehicles are initially located at a central depot, we have distributed the initial position of the vehicles over the map, since we take also into account the setting where the vehicles are independent forwarders.[2]

[1]The use of only three problems was sufficient since tests with more problems showed that the variation of the results were considerably small.

[2]Due to historical reasons these tests were actually run on the MAS-MARS [FMP96] system, the predecessor of TELETRUCK which has restricted capabilities but behaves equally concerning this issue.

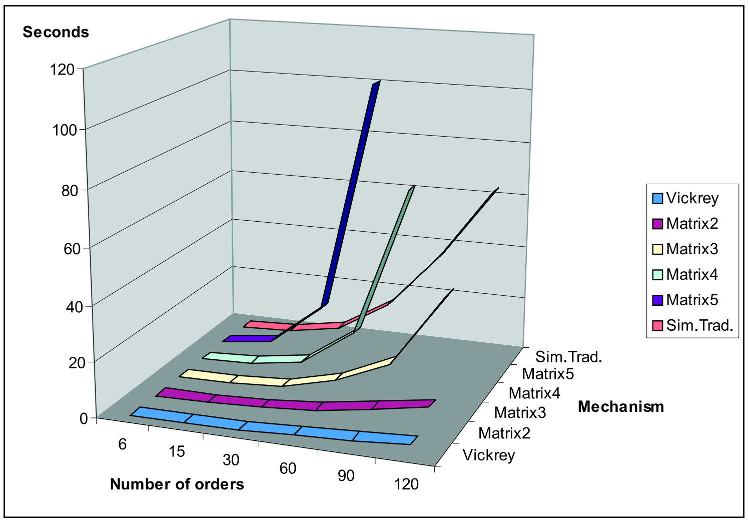


Figure 7.8: Run time of the different mechanisms

In order to examine scalability, we have run every combination of test sets for 6, 15, 30, 60, 90, and 120 orders[3], leading to a total number of 422 test runs. The experiments have been performed on a 233 MHz Dual Pentium II PC with 256 MB RAM under Linux. The results of the following experiments are listed in Appendix B.2.

Results

Number of vehicles: The number of vehicle agents, being the base for our theoretical complexity analysis, depends on the size of the order set: It is increased dynamically whenever the present number of vehicles is not sufficient for the planning of the task at hand.

As stated above, the number of used vehicles can also influence the choice of the mechanism, since the vehicle fleet produces also maintenance costs a shipping company has to minimize. However, the use of different protocols leads to roughly the same number of vehicles in action (Figure 7.7). MA-3 uses slightly fewer vehicles than the other mechanisms.

Tractability: We have measured the running time of the various protocols for different number of orders. Here (and in the following) we average over quantities that are out of focus: For this examination, we have averaged over twelve results from clustered and non-clustered data sets.

[3]For the latter order set we enlarged Solomon's original benchmarks.

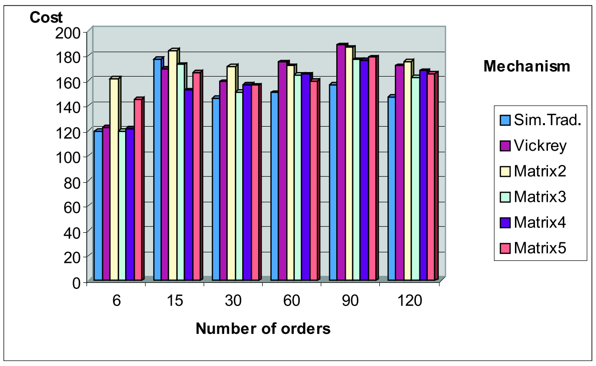


Figure 7.9: Overall cost per order

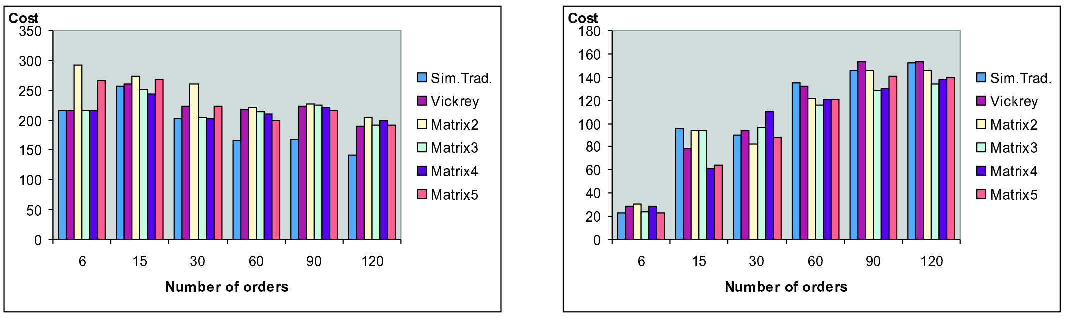


Figure 7.10: Overall cost per order for clustered and non-clustered settings

The Vickrey auction performs quite well, while the matrix auctions' running time is growing very fast with an increasing number of orders. (While the Vickrey mechanism could allocate 120 orders in 3.4 seconds, the test runs of MA-4 with 120 orders took about 15 minutes and MA-5 with 120 orders took more than 6 hours each.) Tractability is no longer fulfilled in such cases. Figure 7.8 shows the results, up to a maximum running time of 2 minutes. The figure does not contain all results from MA-4 and MA-5, since this would reduce the expressiveness of the figure.

Performance differences between clustered and non-clustered data sets were significant only for MA-4 and MA-5 with high numbers of orders. In extreme cases, processing non-clustered orders took up to 50 times longer than clustered ones. Similarly, starting with two initial vehicles only outperformed starting with m initial vehicles at MA-4 and MA-5 with high numbers of orders in a significant fashion. In all other cases, no significant difference could be found.

As the theoretical analysis has already indicated, the runtime of the MA-4 and MA-5 are intractable for large sets of orders while the others can be rated as tractable.

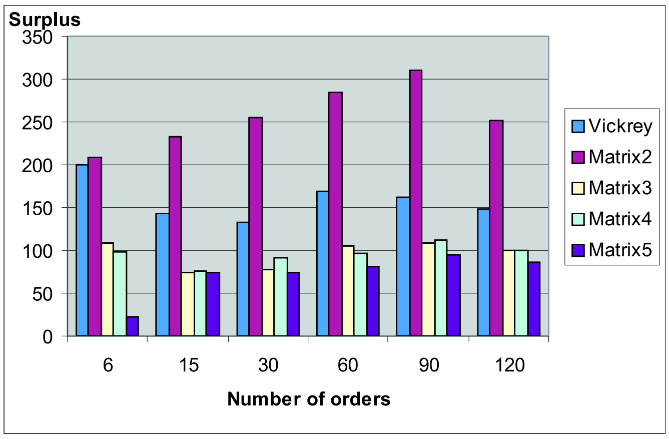


Figure 7.11: Overall surplus per order

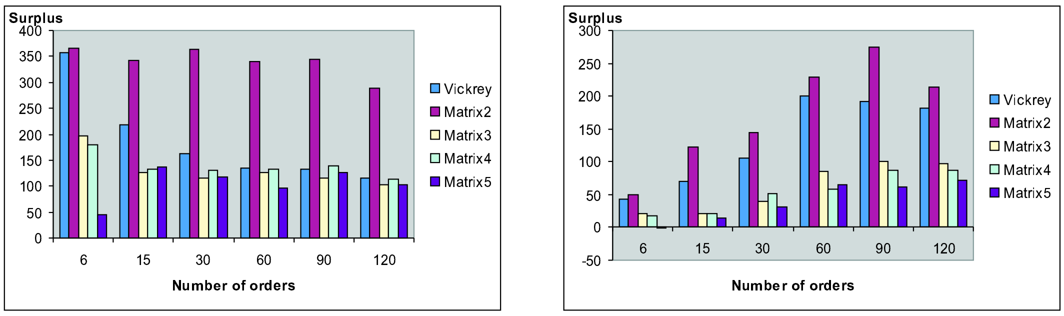


Figure 7.12: Overall surplus per order for non-clustered settings

Efficiency for the cooperative setting: As stated above, for such a setting, cost per order is the crucial issue. Generally speaking, all protocols show rather similar results. However, the simulated trading procedure is proved to be most efficient for large order sets where much optimization can be done. MA-3 and MA-4 perform slightly better than the remaining protocols. Hence, simulated trading would be the protocol of choice for the company agent. Figure 7.9 shows the results.

Interesting though, is the discrepancy between clustered and non-clustered settings (Figure 7.10). Generally speaking, the costs for the performance of clustered task sets are lower than in the non-clustered setting, since vehicles mostly act within one cluster. This effect vanishes for large order sets.

In the case of clustered orders simulated trading looses its advantages over the matrix auctions, in such settings MA-3 and MA-4 outperform the other mechanisms mainly for tasks of 60 and more orders. This matches the intuition that it is cost efficient to allocate bundled tasks in clusters.

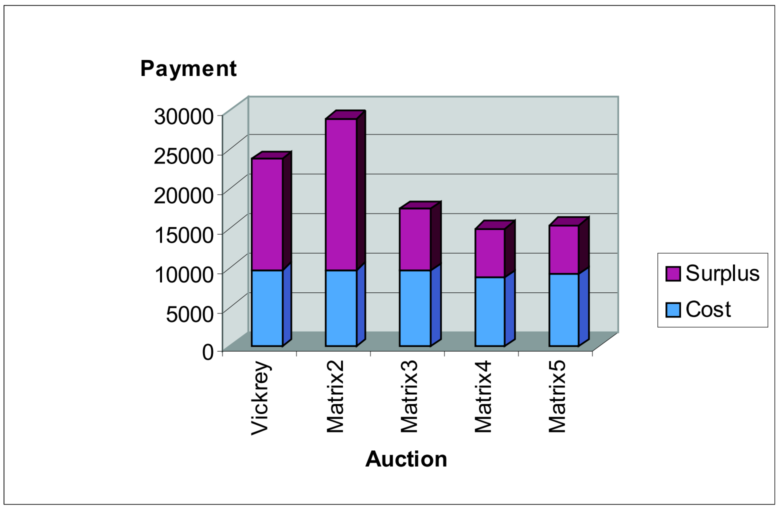


Figure 7.13: Payment for 60 orders

Efficiency for the competitive setting: In a setting, where only vehicle agents optimize their benefit, their individual surplus has to be compared (Figure 7.11). In this setting simulated trading is not applicable.

MA-2 outperforms dominantly all other mechanisms, followed by the Vickrey auction. Figure 7.12 shows differences between non-clustered and clustered cases: for clustered cases, the Vickrey auction performs almost as well as MA-2, which is not the case in the non-clustered case. In general, surplus in the non-clustered case is roughly independent of the number of vehicles, which again is not true for the clustered case.

Efficiency for the hybrid setting: In a setting where all, auctioneer and bidder agents compete for profit, both surplus and cost have to be taken into account, since vehicle agents strive to maximize surplus, while the auctioneer tries to minimize the overall payments, i.e., the sum of cost and surplus. The goals of auctioneer and bidders conflict; hence, the protocol to be chosen then depends on the influence or power of the auction participants.

However, costs do not vary significantly, as Figure 7.13 shows for a representative example, where 60 orders were processed (starting with $m = 10$ initial vehicles). Hence, the expected surplus will determine the common choice (which is MA-4 in the case that the auctioneer is dominant and MA-2 if the vehicle agents are dominant).

7.2.3 Evaluation of Different Geographical Structurizations

Now we evaluate a second extension of the original TELETRUCK system in which we empirically examine the usefulness of the introduction of a subsidiary agent in the hierarchy, as already shown in Figure 7.6. This subsidiary agent pre-classifies incoming tasks according to their geographical location into disjoint subsets of the fleet.[4] By applying the original TELETRUCK approach on these subsets of orders and suitable partitions of the sets of drivers, trailers, and chassis, a cut-down of computational complexity is desired.

We now validate the approach by comparing *de-centralized* settings where the subsidiary agent pre-classifies the order set (into five partitions in our case) and *centralized* settings, where the task allocation is performed on the entire sets of orders, vehicles, drivers, and trailers. In particular, we focus on the following issues:

Tractability: We examine the run times of the system in the centralized and the de-centralized configurations. Due to technical reasons, we only apply the contract net protocol as a RAL. However, we can use results of the examination from the previous section in order to estimate the applicability of other RALs, such as the simulated trading or the matrix action mechanisms, in particular with respect to very large order sets.

Efficiency: We compare the efficiency of the different configurations according to three criteria: the degree of used vehicle capacity, and the duration and distance of the scheduled vehicle routes which we measure per vehicle, and not in total over the entire fleet.

The Analyzed Problem Classes

We use a collection of orders sets that have been generated automatically according to a number of statistical parameters: In a first step, five German metropolitan areas have been picked as *centers*: Hamburg, Hanover, Berlin, Saarbrücken, and Munich. Based on an equal distribution, we have chosen 180 towns that are located no further away than a certain distance from each center. For *short-distance orders*, towns may only be located 20 km away from a center, while for *middle-distance* and *long-distance* orders, towns may be situated at most 50 or 100 km resp. away from a center.

[4]Due to implementation reasons, the pre-classification is actually not performed automatically so far; we have simulated this classification by running separate TELETRUCK engines on subsets of orders.

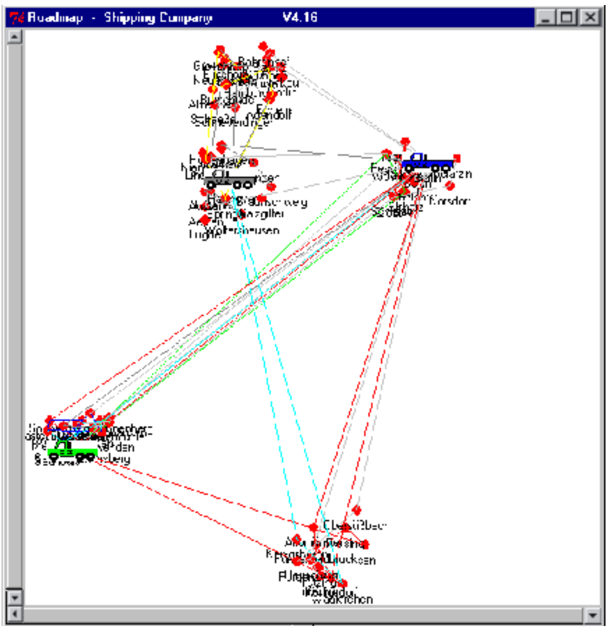


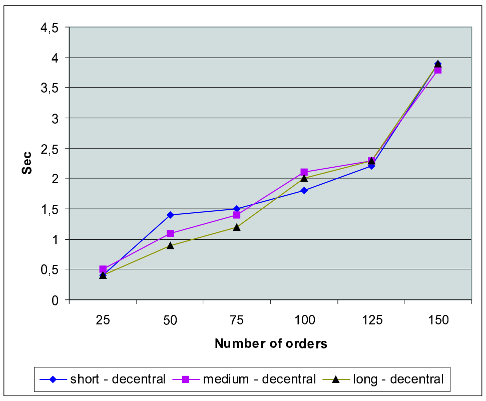
Figure 7.14: Vehicle schedules of medium-distance orders

For each of the 15 distance-center combinations we have generated 30 orders where the customer, sender and recipient are located in one of the previously selected towns. The earliest departure time and the latest arrival time of the freight as well as the load capacities of the vehicles have also been determined according to pre-specified probability distributions. For instance, Figure 7.14 shows a run of a middle-distance scenario without pre-classification by a subsidiary agent.

To compare the settings with and without a subsidiary agent in terms of an increase of orders, we perform test with 5, 10, 15, 20, 25, and 30 orders *per center*, i.e., the test sets contain between 25 and 150 orders. Hence, in total we have performed 150 test runs where we have used a 450 MHz Dual Pentium II PC with 512 MB RAM under Windows^{TM} NT. All values of the experiment results are listed in Appendix B.3.

Results

Tractability: For each of the test runs we have measured the run time of the TELETRUCK system. As Figure 7.15 shows, there is a significant run time difference between centralized and de-centralized settings: In the latter case, the run time is worse by more than one magnitude, which can be explained by the increased computation complexity and the fact that the processing of long-distance queries to the distance data base takes significantly longer than for shorter-distance queries.



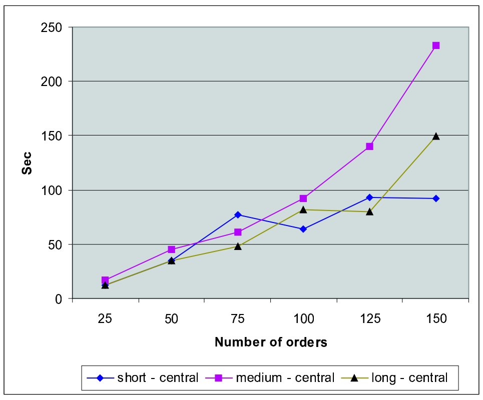
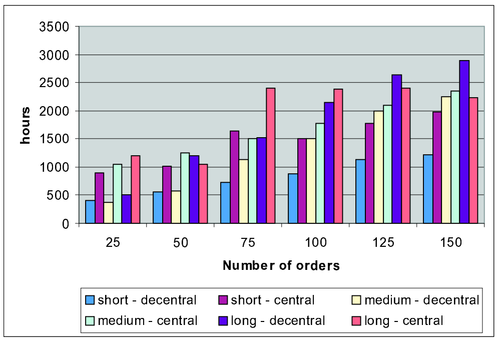


Figure 7.15: Run times

The left hand-side of the figure shows the results for the de-centralized cases. The run times do not differ severely for short-, medium-, or long-distance orders. In any case, the performance increases more than linearly. The right hand side shows the performance for the centralized cases. Surprisingly, orders of medium length need significantly more time to be scheduled than short-distance or long-distance orders.

Efficiency in terms of duration and distance of the scheduled routes: Figure 7.16 shows the quality of the solutions for both, travel duration and route distance which all lie in the same magnitude. For short-distance orders, the de-centralized approach clearly outperforms the centralized one, while for medium-distance orders no significant difference could be measured. In large long-distance cases however, the centralized approach (in which the tour duration and length per vehicle almost remain constant), behaves better than a de-centralized approach since in these cases, the order set and the fleet cannot always be partitioned optimally anymore: under certain circumstances, a vehicle that has originally been assigned to a different partition may perform an additional tour at smallest costs. In the decentralized case, this vehicle is not allowed to participate in the task allocation procedure, leading to a sub-optimal solution.

Efficiency in terms of use of vehicle capacity: This criterion describes the idle time of the vehicles during the execution of a schedule. Figure 7.17 shows the results of this examination, where again we distinguish between short-, middle-, and long-distance orders. In the first case, the decentralized approach behaves extremely badly, not only in comparison with the corresponding centralized case, but also with any other scenario. For middle-distance orders, the decentralized approach still behaves worse than the corresponding centralized approach. Both



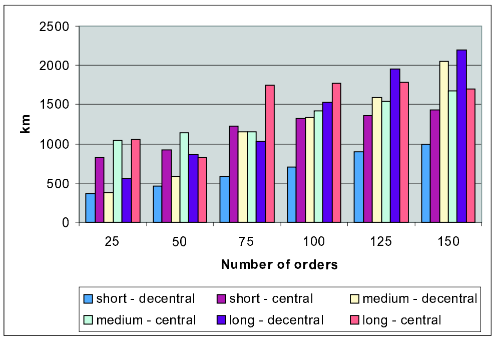


Figure 7.16: Travel times and distances

phenomena may be due to the fact, that in this case, often additional vehicles have to be employed for small tasks, leading to very large idle times in their schedules. Only for long-distance orders, a both approaches perform roughly equally well.

7.2.4 Summary

Empirical evaluations in the TeleTruck domain have shown that the usefulness of different RAL schemes, and also the usefulness of a partitioning of the system, (for instance through the introduction of a subsidiary agent) depend on two issues: the *nature of the target function* to be optimized and the *size of the task.* In Section 7.2.2, for instance, we could show that an optimization towards a cost reduction favors the MA-3 auction while an optimization according to surplus maximization leads to the MA-2 auction as the RAL of choice. In the experiments of Section 7.2.3 an optimization towards distance minimization supports the partitioning of the order sets an vehicle fleet while an optimization towards a high use of capacity supports the opposite approach.

Another important factor is the size of the application: In Section 7.2.2, we showed that matrix auctions with a high number of orders to be proposed at time are not feasible for large-scale tasks. Section 7.2.3 revealed that even for the contract net protocol, a high number of orders lead to infeasibility for a centralized approach. Combining these two results leads to the realization that for the employment of more complex RALs (such as the simulated trading or matrix auctions) a decomposition of the order set (e.g., according to a geographical dispersion) is inevitable to achieve and keep scalability.

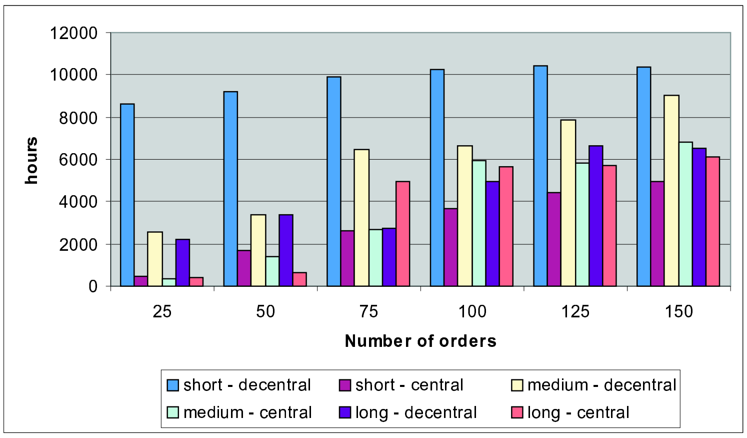


Figure 7.17: Idle times

7.3 The Intelligent Flexible Manufacturing System (IFMS)

This section focuses on holonic agents in the manufacturing domain. As in the previous sections, we first describe the domain, then present the holonic agent architecture and the GRAIL-based optimization scheme.

7.3.1 The Domain

Since the very beginning of research into the design and implementation of an *intelligent manufacturing systems (IMS)*, information technology has played an important part in this development. About twenty years ago the idea of *computer-integrated manufacturing (CIM)* was proposed by [Har93] (reprint). Over the years its original approach changed from a mainly centralized model to a more decentralized one for a number of reasons: faults in individual components can bring a centralized system to a halt, often expert knowledge is needed to run such a monolithic system, the introduction of a centralized system can hardly be performed step by step, etc.

During recent years, first attempts to incorporate multi-agent techniques to CIM have been formulated (e.g. [VDP87, Fis94, RCA98]). New developments in manufacturing do not only make use of improvements in computer science; they also impose new challenges to computer-based control systems. Two aspects are of great importance:

Dynamic rescheduling: Even if an efficient schedule has been determined for a set of production tasks prior to the actual manufacturing process, it is not guaranteed that this schedule will be fulfilled: work stations may fail, supply parts may be out of stock, workers might not show up for work or may be injured during their work time, etc. All these problems may lead to delays in the schedule execution which cannot be seen in advance. Techniques for *local re-scheduling* have to be developed and incorporated.

Plant configuration: In state-of-the-art production plants, an increasing effort is put on the optimization of the topological layout of work stations and robots. Often however, the configuration of a factory is determined in its planning phase and then remains fixed during the whole use of the factory. Today, manufacturers seek for more flexible structures to cope with frequent changes in production. Future generation work stations are supposed to become more mobile, leading to higher flexibility in terms of a configuration re-arrangement for a new product. In the 21^{st} century, re-configurating will be a question of hours and days, but not weeks and months as it is currently.

Solutions for both aspects have been proposed, however not in an integrated fashion: most systems focus on either one of the two issues. The holonic-based approach IFMS (*Intelligent Flexible Manufacturing System)* integrates both aspects.

Details of the Domain

IFMS has been developed in cooperation with manufacturing experts from Daimler-Chrysler and bases on their estimations of the development of work stations in the next ten or twenty years in the automobile production business. Although the original scenario is placed in automotive industry, our approach is designed in a generic manner allowing an easy adaptation to other production scenarios. Therefore we generally speak of work pieces instead of car components.

During the assembly procedure in a envisaged next generation manufacturing plant, seven processing operations have to be performed at seven *work stations.* Each of these work stations consists of one automated processing step and several (up to five) manual processing steps, depending on the functionality of the work station. A *topology* (Figure 7.18) specifies the concrete geometrical position and orientation for each work station and defines the directed paths which work pieces can take to pass from one production unit to the next one.

Overall goal of the system is to maximize the production, in particular to guarantee a minimum amount of produced units, independent from possible breakdowns of some work stations. Schedules are computed for the scope of one day.

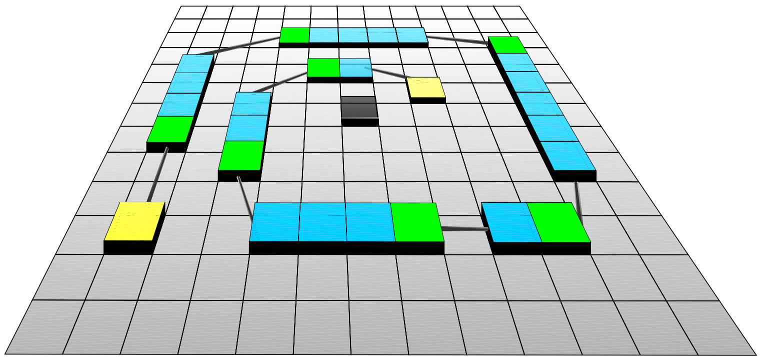


Figure 7.18: Idealized topology of a production plant

The production is organized in a *Just in Sequence (JIS)* fashion: we can assume that components are delivered to the work stations in time and in sufficient stock.

The space of the factory building is limited; each work station requires some square meters of space. The production process may be disturbed due to the failure of the automated parts of the work stations. Failures of the manual processing steps can be neglected since additional workers are assumed to be always available to stand in if an accident occurs. Each work station can perform one or more functionalities; multi-functional work stations have higher fixed costs. Each automated or manual processing step at a work station lasts for a certain time period (in the magnitude of a minute). The repair time of a work station takes time in the magnitude of half an hour. In addition to work stations, one or several *buffers* can be placed into a topology. A buffer can temporarily store a work piece if no work station is idle that could allow for a further assembly on that piece. In this visionary scenario, buffers are assumed to have infinite capacity.

Work pieces may be produced in a number of different variants. The transportation speed of work pieces between work stations is 1m/sec. The distances between work stations are not uniform in general which leads to asynchronous processing. It takes some seconds for a work piece to enter or to leave a work station or buffer. We assume mobility of work pieces at any time; by allowing the work pieces to move from one work station to another on the direct way. We do not consider collisions.

Although this scenario may generalize some aspects of a real-world production plant it is detailed enough to model realistically the two problems IFMS addresses: (1) to find a suitable topology that makes best use of its capacities given a certain failure probability of the work stations, and (2) to provide a technique for on-line re-scheduling in order to compensate the breakdown of a work station by other machines that have suitable time slots in their schedule.

7.3.2 Overall System Architecture

To cope with these two complex and interacting challenges, IFMS has the following system architecture:

Simulator: The central component of the system is the simulator where the schedules are executed on a sufficiently realistic simulation of the plant. In this simulation, machine breakdowns may occur based on a specified probability distribution.

Editor: With this component, the system designer can enter an initial topology. The designer can incorporate expert knowledge into this layout (by inserting a topology which is assumed to be efficient) or he can generate new topologies from scratch.

Off-line Scheduler: IFMS uses a simple scheduler based on the constraint-solving facilities of Oz to compute an initial schedule for a given set of tasks and topology.

Micro-Level Optimizer: Simulated faults induce an on-line re-scheduling. We employ an agent-based technique since this problem can be tackled in a distributed fashion. If for some reason a part of the schedule cannot be executed anymore, agents re-plan the schedule in a decentralized manner.

Macro-Level Optimizer: This component interprets the quality of the result of a simulation (i.e., the number of produced work pieces per time unit) and modifies the topology for another run of the simulation. Hence, in order to optimize the agent society, the simulation is often repeated with the same failure probability distributions of the work stations, but with different topologies. This iteration is performed until a topology configuration is found that satisfies the user's performance requirements, i.e., the amount of units to be produced per day.[5]

The first three components are now characterized, while Section 7.3.3 describes the GRAIL-based optimization scheme.

[5]The macro-level optimizer is only part of this architecture that has not been implemented yet; however we describe its design since it is an essential part of the architecture.

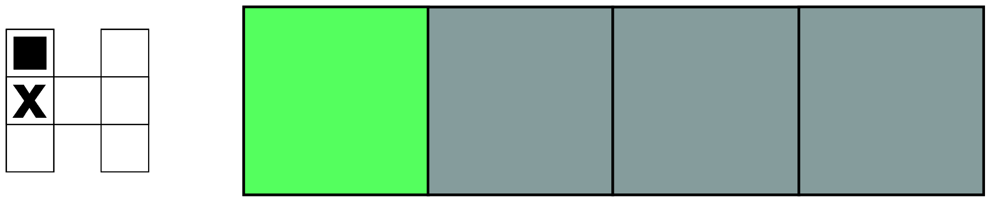


Figure 7.19: A work piece and a work station

The Simulator

If the simulator is provided with a topology and an initial schedule (which can be empty), a simulation run can be started. One of the most important features of the simulation is the simulation of failures of work stations. We employ agents to represent work stations: each agent controls autonomously the current status of its associated station: In our simulation, failures occur according to a predefined probability distribution. The agent modifies the state of the station accordingly.

In such a case, a work piece that is currently located within the automated segment of a broken work station has to linger until the failure is resolved. Work pieces which are processed manually, are further processed without any restrictions. A piece cannot access a broken work station. If it is heading towards such a work station, it will be detoured to a buffer.

Work pieces are visualized by seven small squares combined to the shape of an 'H'. Each square stands for one of the seven functionalities and a symbol in the square indicates if this functionality has been added successfully to the work piece. If an icon shows all seven symbols, all steps to generate the complete product have been established. The left part of Figure 7.19 shows such an icon with two tasks processed. The right part of this figure shows the visualization of a sample work station, which consists of an automated head and three spaces for workers that perform manual processing steps at the work pieces. If a station is currently down, the color of the automated segment of the work station turns from green to red. Figure 7.20 shows a screen shot of the simulator.

The Topology Editor

The topology editor provides graphical facilities to define a factory floor. The user can define up to seven different functionalities for a work station. These functionalities differ in the number of required human workers. Hence the layout of a work station is determined by the maximum worker number of all provided functionalities.

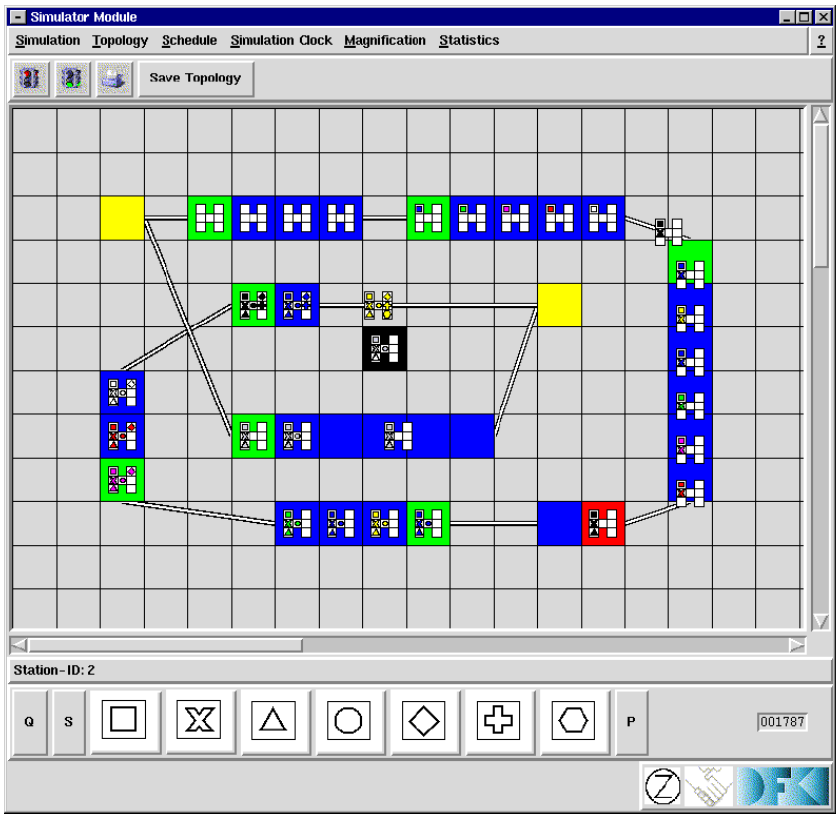


Figure 7.20: Screen shot of the IFMS system

The work flow can be specified by installing paths form one work station to another. To guarantee flexibility of the re-scheduling process, each specified buffer is automatically connected to all work stations. The editor additionally provides means to save and to load topologies. For convenience reasons we have chosen the same graphical representation in both the editor and the simulator. By mouse-click, work stations can be placed on the factory floor.

The Scheduler

Scheduling of processes on a factory floor is in general of great computational effort. For a given number of tasks and a given topology, the schedule specifies the date the corresponding work piece enters certain work stations.

We use constraint solving techniques provided by the Oz programming language. Using constraint solving techniques does not guarantee to produce an optimal schedule for a given topology. However, feasibility of a schedule is given, while the schedule quality can be improved by the use of agent techniques.

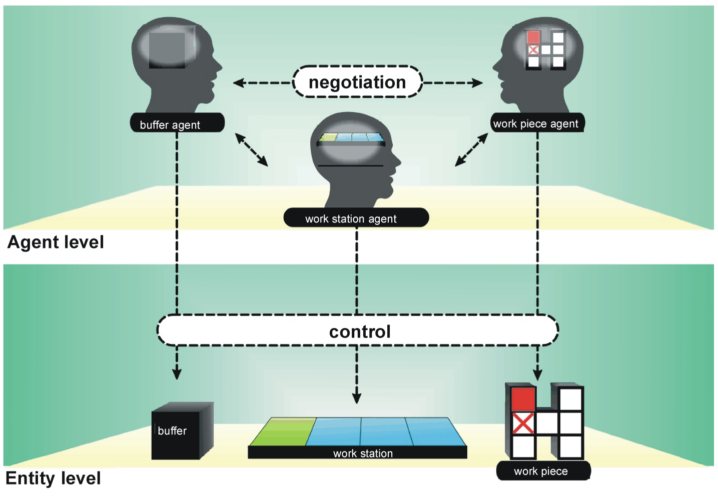


Figure 7.21: Agents and entities

7.3.3 Optimization with GRAIL

Construction of the Holonic Society

As stated before, there are two optimization strategies: on the macro-level, the topology of the work station society has to be optimized, while on the micro-level schedules have to be re-arranged due to system failures. Similar to the MoTiV-PTA scenario, the former optimization problem favors a centralized scheme, while the latter supports a decentralized procedure. We employ GRAIL, since it can naturally be mapped onto the holonic situation.

In the previous section, we have already introduced agents to simulate and represent *work stations*. For on-line re-scheduling on the micro-level, we also agentify *buffers* and *work pieces* (see Figure 7.21). For our purposes, this proves to be the right level of abstraction, because these are the entities that perform the decision making. A more detailed agentification such as agents for workers, does not make sense, since according to the problem specification, the allocation of workers does not influence the scheduling. The goal of agents that represent pieces or stations is to control the processing of the corresponding work piece or station and to initiate and perform a re-optimization in case of a failure.

On the lowest stage, we regard *idle time windows* and *special functionalities* as abstract resources of the work stations: During a re-planning process, they induce restrictions on the planning behavior of other work stations, which leads to cyclic dependencies. Furthermore, the partially fulfilled *work plan* of a work piece imposes additional constraints on the behavior of the work station agents and is hence regarded as abstract resources.

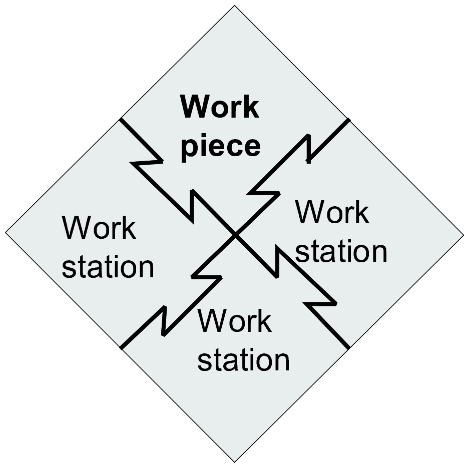


Figure 7.22: Holonic agents in IFMS

Following the arguments of Section 5.3, we consider the set of one work piece agent and seven work station agents that are needed for the construction of the work piece as a *holon* (Figure 7.22), headed by the work piece agent, while the work station agents are the *sub-holons*. Thus, exchanging a work station in the construction schedule of the work piece amounts to a holon reconfiguration. Note that a work station agent is usually a member of several holons at a time. In Section 5.5 we have already discussed the holonic properties of this architecture.

On the macro-level we install a central configuration instance to optimize the topology of the station society. Similar to the MoTiV-PTA domain, a centralized *monitor agent* optimizes the agent topology on the basis of status reports of the work stations according to the number of produced work pieces. Work station agents have to obey the guidelines of the monitor agent. Hence they depend from that agent. The left part of Figure 7.23 shows the resulting initial dependency graph, which is transformed to the one of the right part of the figure after the application of Algorithm 5 of Section 3.5.

The RIL Scheme

According to the GRAIL scheme, the monitor agent spreads guidelines to the members of the society, the work station agents. In IFMS, these guidelines are always sent after a complete simulation run. They contain modifications on position and functionality of the work stations. During the course of the simulation, work station agents log their profiles (i.e., the number of processed work pieces, etc). They have to obey guidelines and pass their local profiles to the monitor agent.

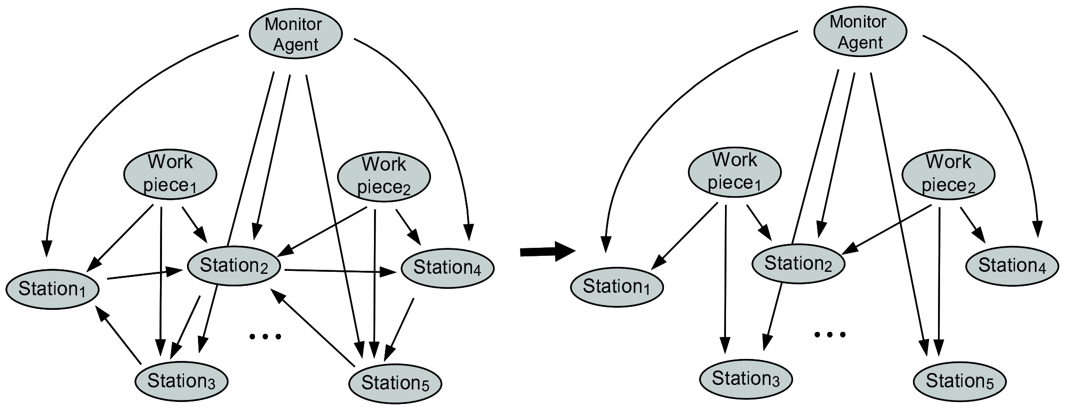


Figure 7.23: Building up hierarchies

Micro-level RAL: Dynamic Rescheduling and Holon Reconfiguration

After the simulator is provided with an off-line computed global schedule, the local schedules for all work pieces are extracted and provided to the associated agents. Such a schedule comprises information on processing steps of the work piece, in particular their assignment to a time slot on a certain work station. Work station agents are also equipped with their local sub-schedules. These schedules contain information on free time slots of work stations.

As stated in Section 5.5, work stations control *station functionalities* as an abstract resource (on a discrete scale where each position on that scale is a combination of possible functionalities) and *idle time slots* (on a discrete scale). For a work piece agent, *production tasks* are viewed as abstract resources to be allocated to the holon members.

Once the simulation is started, the global schedule is executed. In the course of the execution, work station failures may occur according to the previously defined probability distribution. If a work station fails, the information about the failure is propagated to its representative. The agent sends this information to the heads of all holons it is member of. The work piece of such a holon is now no longer expected to be processed according to the original schedule. Hence the head releases this work station agent off the holon society and unlocks the time slots that had been assigned to this station.

In the next step, agents start to negotiate about the vacant holon membership according to a predefined protocol. For this task we employ a the contract net protocol. Due to the modularity of our system, other protocol-based RALs could also be employed. As a first step the head of the holon sends a call for bids for the needed functionalities to adequate work station agents.

The work station agents determine possible time slots to provide the required functionalities. Multi-functional work station agents give bids for each providable functionality. If a work station is currently broken, its agent can bid for tasks, which start after the estimated repair time. If no estimation is possible, the agent cannot send a bid.

Once all bids have been received by the holon's head (or timed out), they are evaluated. The head has to take into account the start times of the slots and the distances to the stations. It tries to find efficient slot sequences to achieve a fluent processing. Finally, the head confirms the selected agent to join the holon and rejects all other applications. The agent of the selected work station has to update its time slot.

Macro-level RAL: Production Plant Configuration

For macro-level optimization, we plan to use the different variations of Greedy-RAL (see Section 3.4). The monitor agent has to reason over the configuration of the following abstract resources:

- **Number of work stations** (Range between 1 and 20 on an integer-valued and cardinal scale and a standard step size of 1),
- **Geographical position of the stations:** Work stations are placed on a two-dimensional grid. For the sake of simplicity, we split *position* into two quantities *x-coord* and *y-coord* (both on an integer-valued and cardinal scale between 1 and 100 and a standard step size of 5),
- **Functionalities of the stations:** In comparison to the MoTiV-PTA scenario, we use a more centralized approach: In the former case we had introduced *abstract currency* which has been spread centrally, but has been locally converted to server functionalities by each member agent. In this case, we assign the work station functionalities directly by the monitor agent, since there are a number of constraints that are obeyed best in a central way (e.g., certain mandatory sequences of functionalities in the manufacturing chain). We split this issue into seven sub-resources, each of which on a discrete an nominal scale, modeling the functionality of one work station.

The target function is expressed by the overall number of produced work pieces.

Integrating Heuristics

To break down the complexity of the search, certain efficient substructures have been identified by the experts from Daimler-Chrysler. The search space then

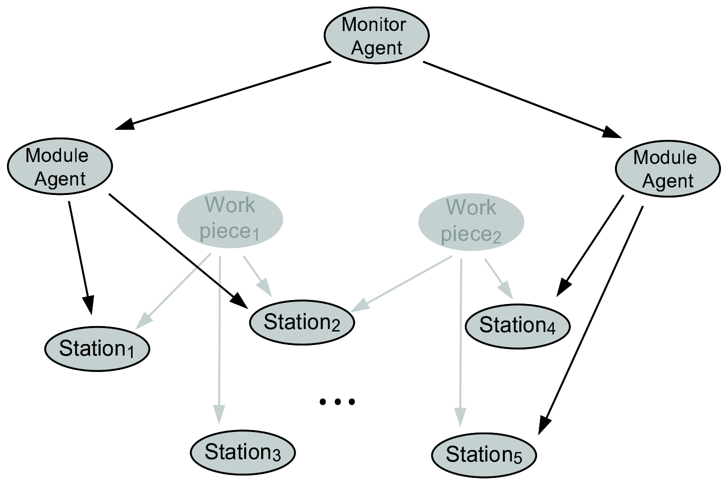


Figure 7.24: Extended agent hierarchy in IFMS

reduces to find an efficient the configuration of these *modules*. The following modules have been identified:

- **Chain:** The most widely-used module in current topologies is a linear sequence of work stations, each of which provides exactly one processing operation. The advantages of such a simple substructure are few requirements to logistics, high productivity, and high use of capacity. However, it does not provide flexibility to disturbances in the production process; delays are propagated throughout the entire sequence.

- **Scope model:** In the scope model, every work station in a module is additionally equipped with the functionality of its direct predecessor in the work flow. If a station fails, work pieces can be directly forwarded to the next station where two production steps are performed. This approach is expected to show a rather high use to capacity and productivity at relatively low additional cost in comparison to the simple chain model. Additional requirements to logistics should be rather small.

- **Parallelism:** This module places several work stations in parallel in order to achieve a higher flexibility in case of station failures. However, not using all work stations for most of the time is expected to a decrease of the average plant productivity.

- **Multi-functional stations:** Such a station can perform all necessary tasks to produce a complete work piece. Employing such a station requires only little additional space in the factory hall. Finding an efficient Schedule for such a work station is hard due to the bottleneck of having only one automated head.

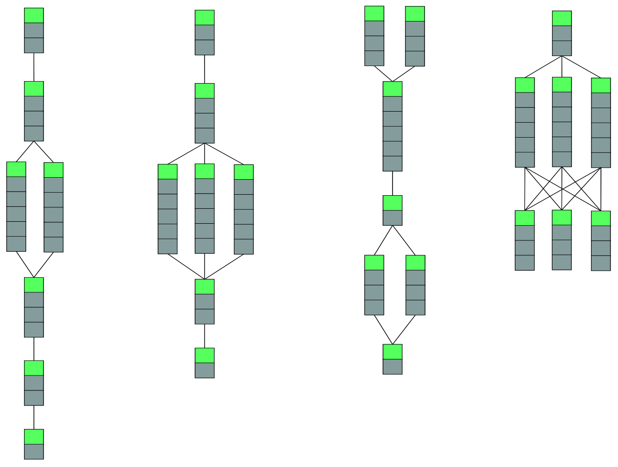


Figure 7.25: Four ways to integrate parallelism into a chain topology

These pre-defined modules implicate the need for more structure in the holonic society. We install *module agents* that are placed between the top-level monitor agent and the work station agents (see Figure 7.24). A module agent optimizes the internal structure of a sub-module of one of the above kinds. The dimensions of the corresponding RAL search space are *type of module*, its *position* and *functionalities* of the members. The dimensions of top-level search space (run by the monitor agent) reduce to *number of modules* and their positions.

7.4 Evaluation of the IFMS Approach

7.4.1 The Analyzed Problem Classes

As the macro-level RAL has not been implemented yet, we only evaluate different sub-modules. We compare the following topologies:

- **Chain:** All stations are ordered in a line; each on which has only one functionality.
- **Scope:** All stations are ordered in a line; each on which can perform two functionalities.
- **Parallelism:** We examine four different ways of parallelism: in a *(3;4)-parallelism*, two work stations are placed in parallel, each of which contains the third and fourth functionalities as shown on the very left of Figure 7.25; similarly, we test a *(3;4;5) parallelism*, *(1;2 - 5;6) parallelism*, and a *(2;3;4 - 5;6;7) parallelism*.

- **Multi-functional workstations:** We test the use of one, two, and three parallel stations that can perform all seven functionalities.
- **Chain with one additional multi-functional workstation:** The additional workstation is able to perform all seven production steps. It is connected to any station in the chain.
- **Scope with one additional multi-functional workstation:** We combine a scope module with a multi-functional workstation.

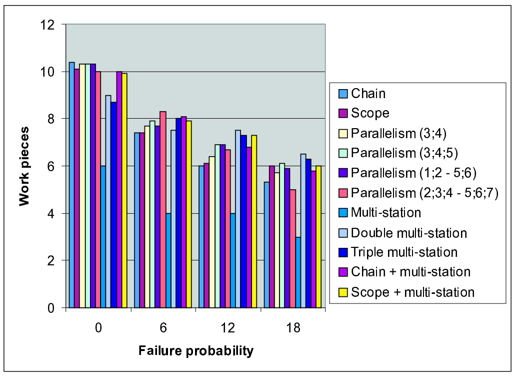
During execution of a simulation run, a sequence of machine faults occurs at predefined points in time and with a predefined duration. We apply test runs on four different patterns, in which stations are broken in 0%, 6%, 12%, and 18% of the working time. All topologies are additionally equipped with one buffer, since station failures would otherwise lead to a work-flow jam that inhibits any reasonable performance examination.

We must also consider the costs of a topology, since using a topology with more work stations obviously leads to a higher absolute production amount than using a topology with fewer stations. For these reasons we regard the ratio between the number of produced work pieces and the number of work stations in the topology. However, even the complexities of work stations vary since stations may provide different combinations of the seven possible functionalities. Hence, if a cost measure bases on the number of functionalities, not on work stations, we must examine the ratio between production performance and the number of involved functionalities. In the following we examine both cases.

We also consider the degree of capacity utilization of the various works stations as a performance indicator. Here we also distinguish between a course-grained measure that indicates the capacity utilization of the entire station, and a finer grained model that measures the utilization of the different functionalities of each work station.

7.4.2 Results

If we consider the coarse-grained case, the number of produced work pieces does not vary significantly for the various topologies (see the left part of Figure 7.26). Depending on the failure rate of the stations, different topology modules perform efficiently. Only the single multi-functional work station shows very little performance, whereas a combination of several multi-functional work stations promises robustness in cases with a high failure rate. In such cases, the traditional chain performs rather weak; the exact performance values can be found in Appendix B.4.



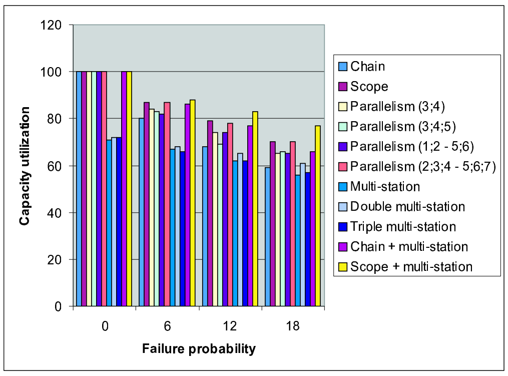


Figure 7.26: Performance per employed work station

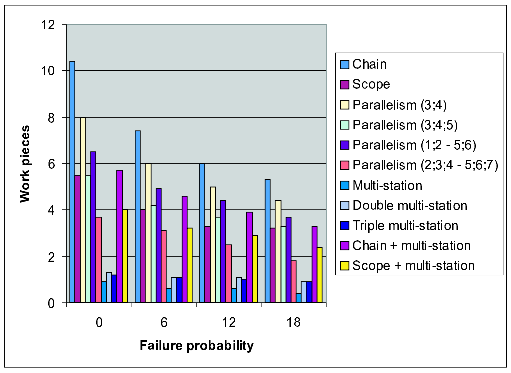
In terms of capacity utilization, a combination of the scope model and a multi-functional station performs best (see the right-hand side of Figure 7.26). This combination outperforms other topologies, in particular in scenarios with a high failure rate. Hence it guarantees robustness. Multi-functional work stations show a relatively low capacity utilization which is caused by the bottleneck of only one automated head in the entire topology.

In the finer-grained case, the chain model outperforms all other topologies, in both the number of produced work pieces as well as the capacity utilization (Figure 7.27). However, the performance difference vanishes if the failure probability increases. In such cases, parallel settings efficient, in particular in terms of capacity utilization. Multi-functional workstations show only little performance, since they have a very high number of functionalities per work station.

7.4.3 Summary

All the above mentioned approaches have their advantages and shortcomings. More complex topologies (with more work stations or with multi-functional work stations) provide redundancies that can be used in cases of machine faults. This however, induces sub-optimal capacity utilization of the work stations.

In practice we must take into account the significant costs which are produced by a work station with multiple functionalities. Therefore, one has to carefully evaluate whether an increase in the capacity utilization will actually pay off. If the costs of an additional functionality is rather small, a combination of the scope model and a multi-functional station can reach the efficiency of the chain model, in particular, if the failure probability is high or unknown. Thus, similar to the TeleTruck case study, the problem of finding an optimal topology boils down to the choice of the target function (in this case the question whether to optimize the use of capacity or the number of produced work pieces).



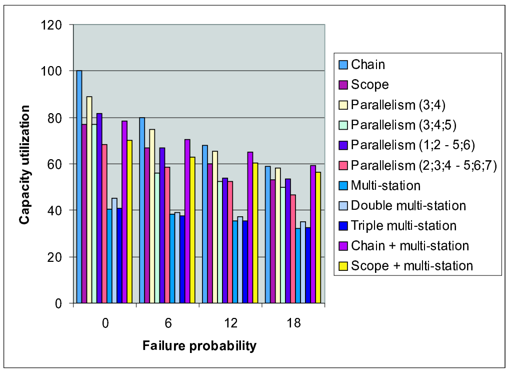


Figure 7.27: Performance per employed functionality

Bottom Line

In this chapter, we have presented and evaluated two case studies that implement the holonic paradigm of Chapter 5. First, we have demonstrated the TELETRUCK system. We have examined the theoretical complexity analysis of several market-based mechanisms for resource allocation. These mechanisms have also been empirically evaluated. Furthermore we have demonstrated how to further structurize the holonic agent society and have examined in which cases a further structurization is advisable.

In the second part of this chapter, we have presented a holon-based approach for job-shop (re)-scheduling embedded into the context of an optimization of plant topologies. Here we have compared several topology modules. As a result, interesting alternatives to the classical chain model could be found which are efficient in particular in scenarios with high work station failure rates. Such efficient modules can be used by a higher-level topology optimization for the layout of a more complex plant topology.

In summary, these case studies underline the applicability of the GRAIL approach in holonic agent structures. The evaluations show that in general a partitioning of the search space and more de-centralized agent structures might lead to results of a lower quality, but at a reduced run time of the system, which however might guarantee scalability in cases where more centralized architectures run out of feasibility. Using GRAIL, the structure of a holonic agent system can be adapted with respect to a target function specified by the user.

Chapter 8

Conclusion and Outlook

8.1 Conclusion

The topic of this dissertation is the development of techniques that allow the design of large, but scalable and efficient agent societies. To do so we have derived mechanisms for a flexible self-adaptation of these societies to dynamic environmental changes.

We have presented an extended concept of resources which not only incorporates computational time and space, but focuses mainly on mutual constraints on the interaction among agents. Using the concept of an *abstract resource*, we have derived a universally applicable hierarchical resource allocation scheme for the entire agent society. This scheme treats resource distribution as a *meta-level* action which interferes only indirectly with the object-level behavior of the agent society. Different search algorithms can be integrated into this scheme as resource allocation algorithms. We have derived and discussed a variety of approaches.

We have presented the SIF system, a toolkit for agent-based simulation. In contrast to other agent-based toolkits, this system provides a modular design through explicit representation of agent sensors and effectors, and the environment the agents act in. These features make SIF very suitable for modeling social interactions among agents. We have developed an extension that supports conveniently the setup of resource-adapting agent societies by providing adequate sensors, effectors and pre-defined resource allocation mechanisms as methods in an abstract base class.

We have defined the concept of holonic agent systems as an extension of resource-adapting multi-agent societies. By restricting the autonomy of the member agents in a suitable way, an efficient trade-off between central optimality and decentral flexibility can be achieved. We have elaborated this paradigm in terms of definitional issues concerning suitable domains and characteristic traits for holonic agents; furthermore we have shown how to integrate GRAIL into the holonic paradigm and have presented prototypical domains and solutions.

We have presented and evaluated three case studies that realize the paradigms of resource aware agent societies. We have shown how GRAIL can be used to optimize a collection of hardware servers in the distributed information system MoTiV-PTA. We have demonstrated resource- and task allocation methods in the TeleTruck system where we have also demonstrated how to further structurize the holonic agent society in order to increase efficiency. We have presented IFMS, a holon-based approach for job-shop (re)-scheduling embedded in the context of an optimization of plant topologies. In this case study we could also show the usefulness of our approach that integrates the macro-level plant topology optimization with micro-level rescheduling of production plans.

In summary, this thesis has provided a self-adaptation approach for multi-agent societies that unites recent developments in artificial intelligence and distributed artificial intelligence, such as bounded optimality, resource-oriented programming, agent-based simulation, and holonic agency.

8.2 Outlook for Future Work

GRAIL can be extended to be even more universally applicable. Two trains of thoughts have been identified and first ideas have already been derived: The incorporation of an evolution-based RAL mechanism and a relaxation of the benevolence requirement. This could be achieved by incorporating a concept from socionics, as we will briefly sketch later.

SIF can be extended to run several media at a time in order to achieve a higher flexibility, especially for the use in virtual reality applications. Such multiple media can be used to canonically simulate different communication channels, and can also be used to model effectively distributed virtual worlds such as *multi-user dungeons (MUDs)* since the wold representation will then be computed in a distributed fashion.

The holonic picture can be extended with a formal characterization of the agent merge process: Sub-holons can merge to one super-holon that displays the same structure to the outside world as the sub-holons. It is desirable to use the tuple-oriented formal agent model to formally define a merge function with Abelian Monoid properties.

The simulation of the MoTiV-PTA case study can be further refined and calibrated to realistic conditions. For instance the service and transmission duration can be simulated in a finer grained way. The task and resource allocation in the TeleTruck domain can be extended to incorporate an optimization of a network of cooperating shipping companies. Progress towards this direction will be derived in [Vie00]. The macro-level plant topology optimization approach in the IFMS scenario can be further developed and evaluated.

We now sketch two preliminary thoughts on the extension of the GRAIL approach in some more detail:

8.2.1 Application of Evolutionary Algorithms in GRAIL

The principle of *genetic algorithms* (GA) is based on the technique of natural selection and inheritance. An initial population of "individuals" consists of randomly generated solutions in the search space. Each solution (or individual) is measured through its performance or *fitness*. This initial population becomes improved during the run of the algorithm: GAs combine the survival of the fittest individuals of a population with the information exchange among individuals. In this process, individuals are selected according to a likelihood corresponding to their personal fitness'.

To model a problem with GAs, a relation has to be defined which maps a solution of the search problem to a bit string, the *genotype* of that solution. New solutions are generated through the use of pre-defined operators. Other, less fit solutions are removed from the system. Usually, the following operators are used:

- **Selection:** According to their fitness, a subgroup of individuals is selected to generate new individuals, their "offspring".
- **Cross-over:** Pairs of previously selected individuals are randomly taken as "parents": their genotypes are mixed according to a pre-defined rule and then used as their offsprings' genotypes.
- **Mutation:** During the run of the algorithm, the genotypes of existing individuals are slightly modified from time to time in order to avoid to run into local optima.

After having generated a new breed of individuals, the fitness of each individual is measured (which may decline for in-principle efficient, but elderly individuals), the least fit individuals are removed from the population, and the selection operation is activated again. This procedure is iterated until a solution of a certain quality has been found or a specified number of iterations has been performed.

Application to Multi-agent Resource Allocation

To apply this search technique to the GRAIL scheme, one could view a given agent society configuration as one “individual” and encode the society structure and relevant internal states of all agents into a genotype. However, traditional GAs are well suited for *off-line* search, where search time is not relevant. Unfortunately, the domains we are interested in are usually highly dynamic, since the environment may change over time. This imposes the need for a fast reaction to such changes. Furthermore, a traditional genetic algorithm needs to process at least some dozens of individuals. This would mean to store the configuration of many complete agent societies which is intractable. Therefore the search method has to be modified.

Due to these restrictions of the situation, it is only feasible to consider *one* possible solution at a time which must be optimized taking dynamic changes of the environment into account. Hence, the specification of what is considered to be an individual has to be re-defined. A straight-forward solution is to regard agents as the basic units in the system. The genotype of an agent can be used to represent a suitable excerpt of its internal state, while its fitness may denote how well an agent performs its task. The fitness can be used to express the solution quality of the given task, but also processing time. Information on internal states of an agent (e.g., communication queues or action queues) can be used as heuristics for such fitness estimations.

The three evolutionary operations *selection*, *cross-over* and *mutation* can be performed on the genotypes of the member agents. This is performed by a representative of an agent group. The genotypes can then be regarded as abstract resources the representative has to reason on. By applying the evolutionary operations, the search space of the RAL is trespassed.

Global feedback can be based on the overall performance of the agent society. If the global fitness decreases, the configuration of the system might be changed rather massively. This can be achieved by speeding up the age-based fitness decay of all society members. If, on the other hand, the global fitness increases over time, the decline of personal fitness due to age is kept low, leading to only small changes in the search space. The genotypes of the representative agents then reflect the macro-level state of a agent group they represent.

A further exploitation of this idea may not only lead to a better resource allocation procedure, but may also reveal interesting results in terms of macro-level simulation of a society in a virtual world.

Control Channel	Situation	Intention
Type of Sanction		
Positive	money	influence
Negative	decision-power	activation of commitments

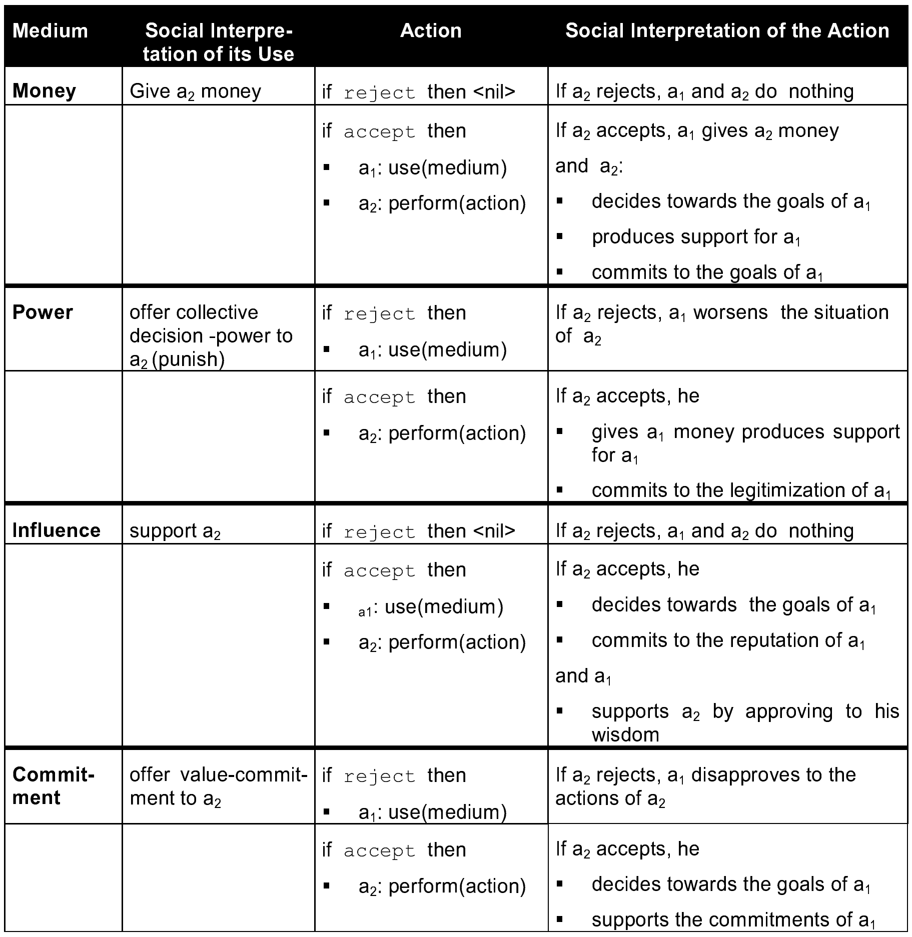
Figure 8.1: Four different generalized media of interaction

8.2.2 Incorporation of a Socionic Theory into GRAIL

The concept of *generalized media of interaction* (GMI) is part of a theory of human society provided by the sociologist Parsons [Par69]. The crucial issue of this concept is: *How can an agent α_1 control the actions of an agent α_2 so that the latter agent acts according to the intentions of the former?* Solutions to this problem setting may be transfered to the GRAIL approach in order to relax the requirements for benevolence where member agents have to obey guidelines put by the monitor agent.

Parsons introduced the problem of how two (human) agents, completely unknown to each other, manage to interact with one another so that they eventually act in a coordinated manner. This problem is often referred to as the problem of *double contingency.* Since these agents usually have some knowledge about the historically evolved world they live in, this problem never actually occurs: Even if a human agent meets a complete stranger in the streets he can still assume many features of that person to hold true. Still, the general problem remains how interaction partners establish expectations about one another's future actions. Generalized media of interaction propose a solution to this problem by providing agents with applicable strategies.

Parsons identifies four strategies that can be used in the interaction to control the actions of another person, namely *money*, *influence*, *power* and *commitments.* Parsons calls these strategies *generalized media of interaction.* They all are based on the possibility to sanction the actions of another agent. Parsons distinguishes between *positive* and *negative sanctions* (i.e., rewards and punishments). The different media of interaction can be distinguished according to their *channels of control.* An agent has the following social interaction abilities: he can give money, influence, take a collectively binding decision or commit itself to some value. Figure 8.1 displays the resulting matrix.



Medium	Social Interpretation of its Use	Action	Social Interpretation of the Action
Money	Give a_2 money	if `reject` then <nil>	If a_2 rejects, a_1 and a_2 do nothing
		if `accept` then ▪ a_1: use(medium) ▪ a_2: perform(action)	If a_2 accepts, a_1 gives a_2 money and a_2: ▪ decides towards the goals of a_1 ▪ produces support for a_1 ▪ commits to the goals of a_1
Power	offer collective decision -power to a_2 (punish)	if `reject` then ▪ a_1: use(medium)	If a_2 rejects, a_1 worsens the situation of a_2
		if `accept` then ▪ a_2: perform(action)	If a_2 accepts, he ▪ gives a_1 money produces support for a_1 ▪ commits to the legitimization of a_1
Influence	support a_2	if `reject` then <nil>	If a_2 rejects, a_1 and a_2 do nothing
		if `accept` then ▪ a_1: use(medium) ▪ a_2: perform(action)	If a_2 accepts, he ▪ decides towards the goals of a_1 ▪ commits to the reputation of a_1 and a_1 ▪ supports a_2 by approving to his wisdom
Commit-ment	offer value-commitment to a_2	if `reject` then ▪ a_1: use(medium)	If a_2 rejects, a_1 disapproves to the actions of a_2
		if `accept` then ▪ a_2: perform(action)	If a_2 accepts, he ▪ decides towards the goals of a_1 ▪ supports the commitments of a_1

Figure 8.2: Effects of generalized media of interaction

Application to Multi-agent Resource Allocation

The use of generalized media of interaction is a promising concept for an extension of the GRAIL scheme to optimize a society of *self-interested* artificial agents. To do so, this concept needs to be formalized. Figure 8.2 shows a first sketch how to implement the effects of the interaction media: The notion of a *currency* has already been introduced to the system in the case study of Chapter 6; it can also be seen as a generalized medium of interaction: If a member agent is not willing to accept limiting guidelines, the representative can transfer additional money to compensate profit loss. This phenomenon also occurs in human interaction and is known as *subsidizing* or *side-payment.* Similar to the attribute *money* the attributes *power*, *influence*, and *commitment* can be associated with society members. A powerful representative can force members to behave in a desired fashion by threatening with negative sanctions or supporting members.

Adding other media than just money to control an agent society may allow for a more flexible self-adaptation in more competitive scenarios and may also be of interest to sociologists in the sense that the extended GRAIL scheme might be used to empirically validate theories derived in sociology.

Appendix A

Proofs of the Corollaries

Corollary 3.4.1 *If the five requirements stated in Section 3.4.1 hold, Greedy-RAL (Algorithm 1), embedded in a RIL (Algorithm 6) that keeps calling a RAL step until a solution is found, will eventually find an optimal position in the local search space.*

Proof: Since the performance function is concave, there exists only one global maximum and no local maxima. Hence, a step in the search space which leads to a performance increase, always strictly decreases the distance of the current position to the global optimum (Step 4 of Algorithm 1).

Due to the concavity of the performance function there must be at least one such direction which leads to a performance increase, if the optimum has not been reached yet. Assuming only discrete dimensions, there is only a finite number of possible search directions. Steps 6 and 7 assure that if a step has been taken that has led to a performance decrease, this step is undone immediately and a step in a new direction is taken.

We never select a direction that has been tested previously; therefore, eventually a direction will be picked that leads to a performance increase, and hence to a reduced distance to the optimum.

The search space is discrete; so there exists only a finite number of points that have a smaller distance to the optimum than the current position. In each iteration one of these points is selected as the new current position. Hence, the number of points with smaller distance to the optimum than the current position strictly decreases and after a finite number of iterations the optimum is reached. □

Corollary 3.5.1 *The cycle replacement technique of Algorithm 5 proposed in Section 3.5.1 terminates and the resulting multi-graph does not contain any cycle.*

Proof: In each transformation (lines 14-24 of Algorithm 5) one cycle is removed (lines 15-16), while new arcs are only introduced from u to v_0 and from v_0 to v for a new node v_0 if there had been an arc from u to v previously that was not member of the cycle. Hence, no *new* cycles can be introduced. A graph can contain only a finite number of cycles, so the algorithm terminates.

We show that $G_{new} \setminus G_{rest}$ is acyclic by induction over the number of loop iterations n.

n = 0 : $G^0_{new} \setminus G^0_{rest} = \emptyset$. The empty graph contains no cycles.

n - 1 → n : We distinguish between two cases:

- There exists a v in V_{rest} with no incoming arcs (Line 5): In this case, v and all its outgoing arcs (v, w_i) are removed from G_{rest} while G_{new} remains unchanged. Hence $V^n_{new} \setminus V^n_{rest} = V^{n-1}_{new} \setminus V^{n-1}_{rest} \cup \{v\}$ and $E^n_{new} \setminus E^n_{rest} = E^{n-1}_{new} \setminus E^{n-1}_{rest} \cup \{(u_i, v)\}$ for nodes u_i in $G^{n-1}_{new} \setminus G^{n-1}_{rest}$. According to the induction assumption $G^{n-1}_{new} \setminus G^{n-1}_{rest}$ is acyclic. Furthermore, v is not member of $G^{n-1}_{new} \setminus G^{n-1}_{rest}$, hence $G^n_{new} \setminus G^n_{rest}$ is acyclic.
- all v in V_{rest} have at least one incoming arc (Line 10): In this case, G_{rest} and G_{new} are modified equally. Hence Hence $G^n_{new} \setminus G^n_{rest} = G^{n-1}_{new} \setminus G^{n-1}_{rest}$. Due to the induction assumption, $G^n_{new} \setminus G^n_{rest}$ is acyclic.

After termination, G_{rest} is empty, so $G_{new} = G_{new} \setminus G_{rest}$, therefore, G_{new} is acyclic. □

Corollary 3.5.2 *If an original dependency graph contains an arc (v, w), then the resulting multi-graph after an application of Algorithm 5 contains an undirected path $(v, v_0, \ldots, v_n, u)$ of the same grey scale where v_i are freshly integrated nodes.*

Proof: We show that if an arc $(v, w) \in E_{rest}$ is processed in n or less loop iterations, $E_{new} \setminus E_{rest}$ contains an undirected path $(v, v_0, \ldots, v_n, u)$, where v_i are freshly integrated nodes.

n = 0 : The assumption holds for the empty graph.

n - 1 → n : We distinguish between two cases:

- There exists a v in V_{rest} with no incoming arcs: In this case, all (v, w) of iteration level n are removed from E_{rest}. Hence $E_{new} \setminus E_{rest}$ contains (v, w). Arcs of levels $< n$ are left untouched in both, E_{new} and E_{rest}.

- All v in V_{rest} have at least one incoming arc: In this case, arcs (v, w) of level n are removed from both, E_{new} and E_{rest}, but arcs (v_0, u) and (v_0, w) of the same grey scale are introduced in E_{new}. Hence $E_{new} \setminus E_{rest}$ contains an undirected path (v, v_0, u), where v_0 is a freshly integrated node.

  Due to the assumption, for arcs (v, w) of level $< n$ exists an undirected path $(v, v_0, \dots, v_n, u)$ of the same grey scale in $E_{new} \setminus E_{rest}$. If an arc (v_i, v_{i+1}) is processed at level n, arcs $(\bar{v}_i, v_i)$ and $(\bar{v}_i, v_{i+1})$ are introduced in E_{new} for a new node $\bar{v}_i$. Hence, $E_{new} \setminus E_{rest}$ contains an undirected path $v, v_0, \dots, v_i, \bar{v}_i, v_{i+1}, \dots, v_n, u)$ of the same grey scale, where all intermediate nodes are freshly integrated.

After termination, G_{rest} is empty, so $G_{new} = G_{new} \setminus G_{rest}$, therefore, the corollary holds. □

Corollary 3.5.3 *The run-time complexity of a centralized resource integration approach of Section 3.5.3 is $O(d_{\text{ral}}(n, r)))$ which could be in the worst case $O(n^r)$, being r the total number of abstract resources and n the maximum number of different values a resource can have.*

Proof: The centralized resource integration scheme runs only one resource allocation algorithm RAL, leading to a total run time of $O(d_{\text{ral}}(n, r)))$ In the worst case no assumptions about $d_{\text{ral}}(n, r)$ (e.g., on convexity, etc.) can be made. Hence an algorithm might test every single position in the search space, before the optimal one could be found. Hence, the corollary holds. □

Corollary 3.5.4 *The complexity of the recursive decentralized resource integration approach of Algorithm 6 is $O(d_{\text{ral}}(n, \tau)^{log_\tau r})$ which could be in the worst case $O(n^{\tau * log_\tau r})$, being τ the minimal number of children a node can have in a dependency graph and n the maximum number of different values a resource can have.*

Proof: Let k denote the number of the currently observed stage of an agent in the hierarchy: agents with no successors are placed on stage $k = 0$, while the top-most agents are placed at stage $k = l$. The complexity of a RAL of agent α_j at stage $k = 1$ takes only $O(d_{\text{ral}}(n, \tau))$. In the worst case, this computation is undone in case of backtracking during the RAL of agent α_i at the level above. Backtracking happens, whenever α_i performs a step in its RAL ($O(n^\tau)$ times). Since backtracking affects all τ children of α_i, the complexity C_k at any stage k is

$$C_k = O((1 + C_{k-1} * \tau) * d_{\text{ral}}(n, \tau)),$$

if optimization of different branches at the same stage is performed sequentially. For the top-staged $k = l$, this can be re-written as

$$C_l = O(\sum_{i=1}^{l}(\tau^{i-1} * d_{\text{ral}}(n,\tau)^i)$$

We show this by induction over k where we set $d := d_{\text{ral}}(n,\tau)$ for the sake of readability:

k = 2 : $C_2 = O((1 + d * \tau) * d) = O(d + \tau * d^2) =$

$$= O(\tau^0 * d^1 + \tau^1 * d^2) = O(\sum_{i=1}^{2}(\tau^{i-1} * d^i))$$

k - 1 → k : $C_k = O((1 + C_{k-1} * \tau) * d) = O(d + \tau * d * \sum_{i=1}^{k-1}(\tau^{i-1} * d^i)) =$

$$= O(\tau^0 * d^1 + \sum_{i=2}^{k}(\tau^{i-1} * d^i)) = O(\sum_{i=1}^{k}(\tau^{i-1} * d^i))$$

According to the induction proof the transformation hold for every k, including $k = l$. So far, we have not considered yet the fact that subordinate problem solvers can run their RAL independently. Since we can assume the parallel computation of τ RALs at the same stage, the factor τ can be removed from the recursive formula leading to a complexity C_k at stage k:

$$C_k = O((1 + C_{k-1}) * d_{\text{ral}}(n,\tau)).$$

In analogy to the above case, this can be re-written for the top-staged $k = l$ as

$$C_l = O(\sum_{i=1}^{l} d_{\text{ral}}(n,\tau)^i) = O(d_{\text{ral}}(n,\tau)^l) = O(d_{\text{ral}}(n,\tau)^{log_\tau r})$$

If we have to assume the worst case, where no information about the allocation space is available, the run time of each RAL is atmost $O(n^\tau)$, leading to a total run time of $O(n^{\tau * log_\tau r})$. Hence the corollary holds. □

Corollary 5.3.1 *If $E = (\{\alpha_1, \ldots, \alpha_n\}, \mathcal{E}, \Pi, \Delta)$ is a multi-agent environment according to the definition in Section 4.1.2 then for each $i \leq n$ there exists an isomorphic multi-agent environment $(\{\alpha_i\}, \mathcal{E}', \Pi', \Delta')$, in the sense of the definition given in Section 5.3.3.*

Proof: Without loss of generality, let $i = 1$. We construct an environment $E' = (\{\alpha_1\}, \mathcal{E}', \Pi', \Delta')$ such that $\mathcal{E}' = \mathcal{E} \times S_2 \times \cdots \times S_n$, $\Pi' = \Pi^1$, and for all $(e, s_2, \ldots, s_n) \in \mathcal{E}'$ and $a_1 \in A_1$ the world function $\Delta' : \mathcal{E}' \times A_1 \to \mathcal{E}'$ is defined as

$$\Delta'((e, s_2, \ldots, s_n), a_1) = (e', s_2', \ldots, s_n')$$

where $e' = \Delta(e, a_1, \phi_2^2(s_2, \Pi^2(e)), \ldots, \phi_n^2(s_n, \Pi^n(e)))$ is the successor state of e in E with respect to a_1 and the other agents' actions $\phi_i^2(s_i, \Pi^i(e))$. $s_i' = \phi_i^1(s_i, \Pi^i(e))$ is the successor state of s_i in E for $2 \leq i \leq n$. The property of isomorphism follows directly from the construction. □

Corollary 5.3.2 *For every multi-agent environment* $(\{\alpha_1, \ldots, \alpha_n\}, \mathcal{E}, \Pi, \Delta)$ *and* $k \leq n$ *we can construct an isomorphic multi-agent environment* $(\{\alpha', \alpha_{k+1}, \ldots, \alpha_n\}, \mathcal{E}, \Pi', \Delta')$*, where* α' *is a holonic merge of* $\alpha_1, \ldots, \alpha_k$ *in the sense of the definition given in Section 5.3.3.*

Proof: We construct an environment $(\{\alpha', \alpha_{k+1}, \ldots, \alpha_n\}, \mathcal{E}, \Pi', \Delta')$ with an agent $\alpha' = (S', P', A', \phi')$ that emulates the agents $\alpha_1, \ldots, \alpha_k$ and we adapt the perception function and environment function accordingly: $S' = (S_1 \times \cdots \times S_k)$, let $P' = (P_1 \times \cdots \times P_k)$, $A' = (A_1 \times \cdots \times A_k)$, and for all $s' = (s_1, \ldots, s_k) \in S', p' = (p_1, \ldots, p_k) \in P'$, the agent function is defined as

$$\phi'(s', p') = ((\phi_1^1(s_1, p_1), \ldots, \phi_k^1(s_k, p_k)), (\phi_1^2(p_1, s_1), \ldots, \phi_k^2(p_k, s_k)))$$

where the first(second) arity of ϕ' is a tuple of the first(second) arities of the composed agents' functions. Furthermore, we define the perception function $\Pi' : \mathcal{E} \to (P', P_{k+1}, \ldots, P_n)$ by

$$\Pi'(e) = ((\Pi^1(e), \ldots, \Pi^k(e)), \Pi^{k+1}(e), \ldots, \Pi^n(e))$$

for all $e \in \mathcal{E}$. Here, the perception of α' is composed from the perception of $\alpha_1 \ldots \alpha_k$. Finally, the environment function $\Delta' : \mathcal{E} \times A' \times A_{k+1} \times \cdots \times A_n$ is defined as

$$\Delta'(e, a', a_{k+1}, \ldots, a_n) = \Delta(e, a_1, \ldots, a_n)$$

for all $e \in \mathcal{E}$, $a' = (a_1, \ldots, a_k) \in A'$, $a_{k+1} \in A_{k+1}, \ldots, a_n \in A_n$. Isomorphism follows directly from the construction. □

Appendix B

Results of the Evaluation of the Case Studies

B.1 Request Processing in MoTiV-PTA

Results of the Off-line Simulation

		MicroGreedy SpaceEqual TimeEqual					MicroNoOpt SpaceEqual TimeEqual					MicroGreedy SpaceDist TimeEqual					MicroNoOpt SpaceDist TimeEqual				
Run time	req / min	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Leam	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	M aH & L
1	2,97	97	100	99	99	98	99	96	101	99	98	106	99	107	98	104	106	104	97	107	107
2	2,97	97	100	99	99	98	99	96	101	99	98	105	99	106	98	104	105	104	96	107	107
3	2,97	97	100	99	99	98	99	96	101	99	98	106	99	107	98	104	106	104	97	107	107
4	2,97	97	100	99	99	98	99	96	101	99	98	105	99	107	98	104	105	104	96	107	107
5	2,97	97	100	99	99	98	99	96	101	99	98	106	99	107	98	104	106	104	97	107	107
6	2,97	97	100	99	99	98	99	96	101	99	98	106	99	107	98	104	106	104	97	107	107
7	2,94	100	100	99	99	98	99	96	101	99	98	105	109	107	98	104	105	104	96	107	107
8	2,97	99	100	99	99	98	97	102	101	99	101	106	99	107	105	104	106	110	97	105	107
9	2,94	99	100	99	99	95	97	101	101	99	97	105	106	106	107	104	105	105	96	100	107
10	2,97	96	97	99	98	97	98	98	101	97	101	106	107	107	97	97	106	100	97	107	98
11	3,03	93	97	97	100	95	101	102	101	100	94	102	105	98	99	100	100	104	100	107	95
12	2,97	94	97	97	97	93	95	101	95	98	99	96	105	94	98	103	111	102	97	107	107
13	3	95	96	96	97	91	99	99	97	98	94	103	110	104	96	95	101	107	100	104	94
14	2,97	92	102	95	92	91	95	99	94	96	94	104	107	102	98	96	100	98	97	96	107
15	3	93	99	93	92	91	95	99	98	96	94	97	107	105	98	100	104	103	104	99	95
16	2,94	89	102	95	91	90	98	99	94	94	96	106	97	104	89	91	107	106	106	102	104
17	3	89	99	92	89	87	96	101	98	97	92	94	98	95	97	90	99	98	93	92	97
18	2,97	91	100	89	88	85	97	101	97	93	96	102	99	93	92	92	95	100	103	94	98
19	3	90	99	87	90	85	96	102	95	93	94	94	99	93	93	98	100	107	92	93	96
20	2,94	89	97	89	87	86	95	100	96	95	96	90	102	90	98	91	96	111	95	96	102
21	3	89	99	88	86	87	96	102	95	93	90	91	100	95	98	85	104	109	105	93	101
22	2,94	90	97	84	88	85	95	101	91	94	93	92	108	90	91	90	106	99	102	96	101
23	3,03	89	98	83	88	84	94	102	91	90	93	94	101	95	92	88	100	105	91	99	91
24	3,03	89	99	84	89	88	93	102	90	93	91	97	105	90	88	90	102	105	102	91	89
25	2,94	88	99	88	88	84	95	99	91	92	89	93	106	92	95	89	102	103	102	92	93
26	3,03	90	96	88	87	84	93	98	94	93	92	89	98	94	98	92	102	100	90	93	97
27	2,94	87	101	86	87	86	92	98	92	93	90	98	104	86	98	92	99	101	99	96	93
28	3	91	96	84	87	85	91	97	91	88	87	88	105	84	87	90	100	111	99	98	92
29	3,03	90	98	83	86	87	94	97	90	89	90	98	102	88	95	95	99	106	101	93	98
30	2,97	90	96	85	86	86	93	97	87	88	88	95	96	86	90	84	91	106	99	92	100
31	3,03	91	98	88	85	87	91	99	90	88	87	97	106	93	90	83	103	107	91	88	96
32	2,97	88	95	86	88	84	92	101	90	90	85	98	95	90	98	87	100	104	89	92	92
33	3,03	87	98	83	85	85	91	101	85	89	89	91	104	89	93	87	102	110	93	89	90
34	2,97	88	96	83	86	87	92	100	87	89	88	97	101	87	87	88	99	98	96	89	86
35	2,94	87	94	83	86	83	94	100	89	89	87	91	99	95	86	86	97	97	87	96	90
36	2,94	88	98	85	88	85	89	101	88	91	88	93	95	89	85	89	92	102	91	94	96
37	2,94	87	95	84	85	83	93	97	88	87	90	91	103	86	89	83	102	97	85	88	94
38	2,97	89	96	84	89	84	89	100	88	89	89	90	94	90	91	88	90	101	86	88	98
39	3,03	89	97	86	86	81	90	99	88	90	89	91	100	84	85	81	92	100	86	98	85
40	2,94	89	97	88	86	79	90	96	86	87	86	94	105	91	90	91	93	97	97	95	96
41	2,97	90	99	82	86	80	93	97	88	89	89	91	94	93	86	88	94	97	86	93	88
42	2,97	86	96	85	87	80	92	98	88	88	87	98	98	84	84	83	91	106	91	100	95
43	2,94	85	93	86	87	82	91	101	86	91	89	93	100	88	88	82	92	100	89	99	94
44	2,94	87	95	85	84	82	93	102	86	89	89	88	95	86	90	81	100	100	92	89	97
45	3	88	95	83	88	83	90	100	88	90	89	88	104	89	95	80	91	109	85	92	85
46	3,03	88	96	85	85	83	90	97	85	87	88	88	105	84	85	84	90	98	97	95	90
47	2,97	89	95	85	84	79	92	99	85	90	86	97	106	91	91	84	93	106	94	98	94
48	2,97	86	94	81	85	82	90	101	87	90	87	96	99	85	96	81	102	109	89	94	94
49	2,94	88	93	82	88	81	89	97	87	92	89	88	101	82	94	81	96	107	98	93	87
50	3,03	86	96	86	84	81	90	100	86	91	88	89	105	82	90	80	94	106	88	88	90
51	3	89	97	85	85	82	94	97	88	87	86	87	101	87	83	83	91	108	87	98	96
52	2,94	88	96	85	83	79	89	101	85	88	87	85	100	83	83	89	101	97	89	95	97
53	2,97	88	95	85	88	81	92	98	86	90	88	91	100	87	95	87	94	111	97	88	97
54	3,03	86	93	82	87	81	90	100	86	91	88	93	94	87	85	85	90	110	96	93	97
55	2,94	88	94	86	85	79	91	100	85	91	86	95	92	89	95	88	102	101	90	97	95
56	2,97	85	96	86	88	83	92	100	86	89	88	86	103	82	95	89	97	104	90	97	90
57	3	89	95	81	84	80	93	97	88	91	88	91	99	85	84	87	97	99	86	92	85
58	2,94	86	94	85	87	79	89	100	88	91	85	96	104	82	90	78	90	106	97	93	97
59	2,97	86	96	82	83	80	91	99	86	91	89	87	95	91	84	83	98	111	95	95	91
60	2,97	88	97	80	83	80	90	97	87	88	86	92	92	82	88	84	93	106	94	100	90

Results of the On-line Simulation

		MicroGreedy SpaceEqual TimeDist					MicroNoOpt SpaceEqual TimeDist					MicroGreedy SpaceDist TimeDist					MicroNoOpt SpaceDist TimeDist				
Run time	req / min	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	M aH & L
1	4.43	115	107	110	111	111	109	110	111	113	113	107	123	114	109	122	117	119	121	125	110
2	4.43	114	107	110	111	111	109	110	111	113	112	107	123	114	108	122	117	119	121	125	110
3	4.43	115	107	110	111	111	108	110	111	113	113	107	123	114	109	122	117	119	120	125	110
4	4.43	114	107	110	111	111	109	110	111	113	113	107	123	114	109	122	117	119	121	125	110
5	4.43	115	107	110	111	111	108	110	111	113	113	107	123	114	109	122	117	119	120	125	110
6	4.43	115	107	110	111	111	109	110	111	113	113	107	123	114	109	122	117	119	121	125	110
7	4.43	114	110	110	111	111	109	110	111	113	112	116	123	114	108	122	117	119	121	125	112
8	4.43	115	113	110	111	112	110	109	111	113	113	112	110	114	109	122	117	119	117	125	116
9	4.43	114	112	110	110	113	111	112	111	113	113	114	110	114	121	122	117	119	121	125	108
10	4.43	115	114	110	111	110	112	113	110	114	110	112	119	112	112	122	125	118	110	119	111
11	4.53	114	112	111	112	105	113	116	110	113	112	115	119	121	110	120	123	127	118	119	112
12	4.69	114	118	113	107	106	111	113	111	114	113	112	128	116	107	119	119	126	113	120	114
13	4.88	112	114	111	105	107	114	116	112	113	112	116	124	114	110	116	117	122	115	119	113
14	4.88	113	115	111	109	101	113	118	111	112	110	118	129	107	114	111	122	121	112	116	118
15	5.05	109	115	105	107	104	114	122	114	115	112	119	125	113	113	114	120	124	113	112	122
16	5.2	111	119	106	106	103	113	119	112	114	110	119	130	110	116	114	122	123	117	125	112
17	5.19	109	117	106	106	106	112	118	114	110	112	115	123	112	109	111	115	127	117	116	116
18	5.43	110	119	105	107	105	114	122	112	112	111	116	126	103	114	116	122	133	118	118	117
19	5.41	111	121	106	106	106	114	120	110	114	109	115	122	114	113	112	120	124	112	111	115
20	5.59	110	117	106	107	105	114	125	110	112	111	110	123	111	110	112	123	129	119	117	114
21	5.69	113	118	104	109	107	116	122	112	110	107	111	123	108	114	105	116	128	120	121	107
22	5.74	108	122	105	107	104	113	124	108	110	108	116	128	106	111	111	121	130	121	115	116
23	5.77	109	121	104	107	107	114	123	111	112	110	118	129	109	115	109	118	128	116	122	106
24	5.94	109	121	108	109	105	112	124	108	110	107	115	128	112	112	116	119	121	116	115	120
25	5.86	112	121	105	110	102	116	126	109	113	110	116	123	116	113	108	127	130	114	115	111
26	5.83	110	119	107	110	102	115	125	111	113	109	115	127	111	117	111	127	124	121	114	116
27	5.85	112	121	109	106	103	118	124	111	111	108	120	128	108	113	101	120	133	120	115	114
28	5.92	112	119	104	106	102	116	126	109	113	111	115	123	113	115	105	116	131	114	116	108
29	6	110	122	104	109	101	117	124	110	111	109	117	127	118	110	105	117	132	112	117	114
30	6	109	119	106	107	100	113	127	112	113	107	113	130	107	119	103	121	131	112	120	111
31	6	111	122	104	108	103	117	126	111	114	111	111	119	109	115	107	122	133	111	113	114
32	6	113	120	105	108	101	117	123	107	114	112	115	132	110	113	110	118	135	109	119	110
33	5.88	111	119	107	110	101	115	124	109	113	111	112	127	113	117	108	126	135	116	117	107
34	6.01	108	120	104	107	100	114	124	110	112	109	113	125	109	111	102	125	129	113	118	115
35	5.94	112	118	106	105	101	113	127	107	113	108	118	120	111	108	105	121	134	111	115	118
36	5.85	110	119	104	109	101	114	124	110	111	110	111	118	107	112	109	122	128	114	111	109
37	5.83	109	116	103	107	102	114	123	111	111	107	114	120	113	114	106	119	131	109	118	114
38	5.68	106	116	103	106	99	113	126	110	111	107	110	119	106	116	105	118	132	120	121	112
39	5.69	108	119	103	105	97	112	124	106	111	106	110	124	109	110	108	117	130	117	116	107
40	5.61	107	117	105	104	98	112	124	108	111	105	112	122	103	106	103	118	128	107	117	111
41	5.55	105	118	105	103	101	110	125	106	108	105	107	127	103	113	99	118	129	118	114	104
42	5.46	106	116	103	104	96	112	121	108	111	103	108	125	100	106	99	120	122	115	116	114
43	5.46	104	112	101	103	101	111	119	105	108	107	104	120	106	105	99	123	128	116	116	107
44	5.38	105	118	100	105	99	113	120	106	110	103	112	117	98	104	95	116	128	112	112	104
45	5.09	105	113	102	100	95	110	120	102	107	104	113	118	106	111	97	118	125	110	110	109
46	5.03	104	115	101	102	97	110	118	103	108	100	108	123	102	108	96	114	125	108	108	105
47	5.04	101	112	100	99	96	111	117	102	108	101	105	116	99	108	101	113	128	103	117	109
48	4.86	102	110	97	100	94	108	116	105	107	102	103	120	98	99	94	113	117	106	109	107
49	4.77	103	111	98	98	93	107	116	100	106	102	110	116	98	101	98	107	124	107	111	103
50	4.66	98	111	96	99	93	104	118	100	100	99	103	120	95	105	99	104	117	107	104	103
51	4.45	102	107	96	99	90	102	115	103	105	97	98	118	96	98	98	114	112	104	112	107
52	4.34	97	106	94	99	90	103	114	99	100	99	105	106	101	105	99	110	119	98	107	98
53	4.26	99	106	94	96	89	104	111	100	103	94	109	114	97	94	89	103	110	102	109	96
54	4.09	94	105	94	93	89	101	110	97	101	93	103	116	92	96	88	112	111	103	97	99
55	3.93	98	105	90	95	89	101	110	94	100	96	98	106	93	102	94	98	115	99	101	99
56	3.78	91	104	90	93	87	100	104	95	98	95	104	107	91	100	91	105	119	100	107	93
57	3.66	95	103	89	91	84	98	106	95	95	92	94	103	97	89	85	109	110	104	103	96
58	3.47	93	101	88	89	85	95	105	93	96	89	95	110	96	97	90	106	102	102	92	99
59	3.3	92	98	87	90	84	96	102	91	94	86	100	109	91	88	89	97	111	96	95	96
60	3.15	87	97	87	87	83	94	102	86	89	89	88	98	89	89	80	95	108	88	92	85
61	3	88	97	81	86	81	94	100	90	92	88	86	104	87	89	86	100	103	97	89	93
62	2.91	85	94	82	85	81	90	100	84	91	82	83	100	85	89	78	92	100	95	86	82
63	2.82	85	93	83	81	80	88	94	83	85	81	89	93	83	93	86	92	96	84	94	81
64	2.61	85	94	80	82	76	87	94	84	85	82	85	102	81	91	78	86	95	91	94	84
65	2.48	80	87	78	77	76	84	94	83	85	80	81	87	78	80	74	93	94	91	95	81
66	2.25	80	87	77	76	73	83	93	83	80	77	79	89	75	80	73	91	99	86	86	79
67	2.17	79	83	76	77	72	81	89	81	79	76	78	84	75	82	78	84	99	77	81	78
68	2.09	78	83	72	75	71	80	86	75	78	75	84	84	71	79	74	84	94	78	78	81
69	1.89	75	84	69	74	71	79	86	75	77	75	79	84	69	76	71	85	91	79	80	79
70	1.78	73	76	69	70	65	76	86	75	75	72	75	81	68	69	75	77	89	72	83	78
71	1.68	71	74	69	67	65	74	81	70	74	70	73	81	69	68	69	75	82	74	81	76
72	1.56	70	73	66	65	65	73	78	72	72	66	72	82	71	68	63	74	80	76	80	71
73	1.44	69	71	63	67	68	74	76	75	71	69	72	76	67	69	69	76	79	76	74	67
74	1.33	69	70	65	69	71	75	74	72	72	72	73	71	63	71	76	79	75	76	75	71
75	1.23	69	67	68	68	69	71	73	70	74	76	73	72	68	67	72	78	72	75	76	76

		MicroGreedy SpaceEqual TimeDist					MicroNoOpt SpaceEqual TimeDist					MicroGreedy SpaceDist TimeDist					MicroNoOpt SpaceDist TimeDist				
Run time	req / min	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	M aH & L
76	1.09	67	64	66	67	68	68	71	68	70	72	74	73	74	64	69	77	69	74	78	74
77	0.96	66	63	65	67	66	66	69	66	67	69	70	71	72	65	67	75	70	70	71	72
78	0.86	63	62	63	65	65	65	67	65	64	66	67	70	71	67	65	73	72	67	64	71
79	0.78	61	62	61	63	64	64	65	62	62	63	65	69	70	69	64	71	69	64	63	68
80	0.69	59	59	59	61	61	61	63	60	60	60	63	65	67	66	63	67	66	62	63	65
81	0.63	57	57	57	59	58	59	61	57	58	57	61	62	64	63	62	63	63	60	63	62
82	0.58	55	55	55	58	55	57	60	55	56	55	59	59	62	60	61	59	61	58	63	59
83	0.52	54	53	53	56	53	55	57	53	54	53	57	56	59	58	60	56	58	57	60	57
84	0.47	52	52	51	55	51	53	55	51	52	51	55	54	55	56	57	54	55	54	57	55
85	0.42	51	51	50	53	50	52	53	49	51	50	53	53	53	55	55	52	52	52	55	53
86	0.4	50	50	49	51	49	51	51	47	50	49	51	52	50	53	52	50	49	50	52	51
87	0.38	49	49	49	49	47	49	49	46	49	48	49	51	47	52	50	48	47	48	50	51
88	0.36	48	48	48	47	46	48	49	46	48	48	48	50	44	50	47	46	47	46	50	51
89	0.35	48	47	48	45	45	47	49	46	48	47	48	49	44	49	45	45	48	44	50	52
90	0.33	47	47	48	44	46	46	49	47	47	46	49	49	45	50	46	46	49	45	50	52
91	0.31	46	47	47	45	47	47	49	47	46	46	49	50	46	51	48	47	50	47	50	52
92	0.3	46	48	47	47	48	48	49	48	46	47	50	50	47	52	49	49	51	49	51	53
93	0.34	47	49	47	48	50	49	49	48	47	48	50	51	48	54	51	50	52	51	51	53
94	0.39	49	50	49	50	51	50	49	49	49	50	51	52	49	55	53	52	53	53	51	53
95	0.43	51	51	51	52	52	52	50	49	50	51	51	52	50	56	54	53	54	54	51	54
96	0.48	53	52	53	53	54	53	51	50	52	53	52	53	51	58	56	54	55	56	51	55
97	0.52	55	53	55	55	55	54	53	52	54	54	52	53	52	59	57	56	57	58	52	57
98	0.56	56	54	57	56	56	55	55	54	55	55	53	54	53	60	59	57	59	60	54	59
99	0.61	58	55	59	58	58	56	57	56	57	57	55	55	55	62	61	59	61	62	56	61
100	0.66	60	56	61	60	60	58	59	58	58	58	58	57	57	63	63	62	63	64	58	62
101	0.7	62	58	63	61	62	60	61	60	60	60	61	60	60	65	65	65	65	66	60	64
102	0.75	64	61	65	63	65	62	63	62	62	62	64	63	62	66	68	69	67	68	62	66
103	0.84	66	64	67	64	67	64	65	64	64	64	67	66	65	68	70	72	70	70	64	68
104	0.94	69	67	69	66	70	66	67	67	66	66	70	69	68	70	73	76	71	72	67	71
105	1.04	71	70	72	68	71	68	69	70	68	68	71	71	71	73	75	76	73	73	71	75
106	1.14	74	72	75	71	72	70	71	73	71	71	72	73	74	76	77	77	74	74	76	78
107	1.27	76	75	76	75	73	72	73	76	73	74	73	76	78	79	80	78	76	75	81	82
108	1.41	78	78	78	76	76	74	77	76	75	78	77	80	79	82	81	81	81	81	80	82
109	1.54	79	79	80	77	79	76	81	77	77	78	82	85	80	86	83	84	87	87	80	83
110	1.68	80	81	82	78	82	79	81	80	80	79	85	90	84	88	85	87	89	89	81	87
111	1.8	81	83	84	82	83	81	81	84	83	82	88	88	89	90	85	89	92	92	83	92
112	1.92	83	86	84	86	84	84	85	85	87	85	88	88	86	89	86	92	94	92	84	91
113	2.05	87	90	85	89	87	86	89	86	87	88	88	88	84	89	91	88	96	93	86	90
114	2.19	91	90	90	88	88	87	88	91	87	91	86	96	89	87	90	88	91	96	100	95
115	2.4	89	90	89	87	90	89	90	91	89	90	90	96	94	87	90	89	90	94	96	93
116	2.52	90	93	89	88	93	93	93	91	91	92	95	96	99	88	95	98	90	92	93	92
117	2.59	92	97	90	92	93	95	95	92	94	93	101	94	94	91	94	105	95	93	103	102
118	2.67	94	95	90	94	93	96	98	95	92	95	101	97	94	100	102	104	95	101	98	101
119	2.91	94	98	92	94	97	99	99	93	99	96	103	100	101	98	94	104	109	107	96	96
120	2.97	95	97	93	93	95	100	101	99	93	101	101	98	94	96	96	97	108	99	105	110
121	3.12	93	101	94	90	95	101	103	99	99	100	96	101	98	99	100	102	106	108	107	99
122	3.36	98	100	94	94	99	99	101	100	95	101	96	107	102	95	105	102	105	99	103	110
123	3.51	96	101	90	94	94	105	105	99	98	105	105	104	93	102	103	112	112	106	97	104
124	3.66	98	102	89	96	95	100	108	102	102	104	97	109	91	100	105	105	109	99	101	112
125	3.75	100	108	92	97	96	102	110	101	100	109	95	111	95	95	102	111	113	110	100	106
126	3.9	101	108	92	94	95	102	109	100	99	107	107	110	95	107	97	113	114	103	107	118
127	4.09	101	107	98	101	94	103	110	105	103	110	102	121	103	102	100	113	117	104	99	117
128	4.32	105	112	95	98	92	110	111	101	102	113	108	111	100	100	99	117	111	107	112	120
129	4.38	100	108	98	97	93	106	114	102	101	111	103	109	104	112	99	109	111	106	107	123
130	4.45	102	109	95	97	94	105	116	106	104	113	110	115	106	111	95	118	112	110	113	118
131	4.59	100	114	96	101	95	106	116	105	103	114	109	118	101	101	94	109	122	108	109	128
132	4.78	104	112	98	104	94	109	118	104	102	116	102	119	110	108	105	121	128	110	105	123
133	4.9	105	116	99	102	97	109	117	105	105	115	109	119	101	111	104	112	130	109	105	114
134	4.94	104	113	102	104	99	106	119	103	105	113	107	119	103	109	101	111	122	104	114	115
135	5.05	105	113	102	102	97	106	122	104	108	116	115	127	107	111	96	118	123	117	116	116
136	5.25	106	117	104	103	97	110	119	105	110	118	119	114	107	107	105	116	131	102	106	115
137	5.19	106	120	102	103	97	112	117	107	105	118	118	123	104	109	106	115	123	114	113	126
138	5.34	106	118	103	106	98	113	121	108	107	119	111	121	108	103	102	113	126	105	113	126
139	5.52	106	117	104	107	98	112	122	109	108	120	111	121	109	113	100	118	131	116	114	121
140	5.55	110	118	104	106	100	111	124	106	112	119	113	118	107	104	105	113	131	107	116	120
141	5.71	110	121	104	107	100	114	121	109	112	123	120	124	102	114	102	120	120	111	115	125
142	5.64	110	119	107	107	99	114	126	109	113	120	110	125	108	114	105	123	128	110	116	130
143	5.76	111	121	105	105	100	115	124	111	109	119	117	126	110	110	103	123	123	116	111	124
144	5.73	106	117	104	106	101	113	123	111	111	123	114	129	107	111	104	118	122	113	115	133
145	5.77	106	117	106	108	100	115	125	107	110	123	113	125	103	111	105	116	124	109	124	133
146	5.92	110	118	106	107	101	116	125	109	115	123	113	123	111	111	100	115	132	113	114	128
147	5.94	108	116	103	107	99	114	123	107	110	122	116	126	108	109	103	116	128	112	115	129
148	5.98	112	119	102	106	99	114	124	111	113	121	113	130	107	109	105	121	131	120	117	122
149	6.02	110	119	106	107	102	117	125	111	113	124	110	125	109	106	104	124	138	112	118	132
150	6.04	112	120	106	107	101	114	127	112	111	124	114	124	108	108	105	124	136	111	124	129

		MicroGreedy SpaceEqual TimeDist					MicroNoOpt SpaceEqual TimeDist					MicroGreedy SpaceDist TimeDist					MicroNoOpt SpaceDist TimeDist				
Run time	req / min	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	M aH & L
151	6.02	110	116	104	108	101	116	127	107	111	122	114	126	112	110	105	128	132	117	125	126
152	5.94	110	118	106	108	100	114	126	109	111	122	109	120	110	118	104	127	135	114	120	128
153	5.88	107	117	105	105	100	114	124	109	114	123	116	132	109	113	111	120	132	114	119	126
154	5.98	112	117	104	106	102	114	125	111	112	123	117	130	111	113	105	123	132	114	117	124
155	5.91	109	116	103	105	102	116	125	111	112	122	114	122	109	116	106	124	128	115	114	130
156	5.88	108	118	101	107	101	114	124	108	113	121	110	119	110	109	106	118	132	113	114	132
157	5.82	107	119	101	108	102	115	124	109	111	121	110	118	106	111	107	122	130	116	118	126
158	5.68	109	116	101	105	101	115	125	110	113	121	114	115	111	106	105	121	128	118	119	128
159	5.73	107	118	104	104	102	113	121	109	111	116	113	122	114	114	100	112	130	112	119	126
160	5.71	108	118	101	104	99	113	124	107	112	119	109	124	109	113	104	112	131	112	118	122
161	5.49	107	116	101	105	99	113	122	112	111	117	110	124	114	103	103	113	131	111	113	122
162	5.43	104	117	100	103	97	113	118	110	110	115	118	116	109	107	99	122	134	114	107	116
163	5.31	106	114	99	101	96	111	121	105	109	116	112	127	107	111	108	122	127	107	112	129
164	5.31	107	114	100	104	99	112	121	105	108	116	112	121	107	114	106	121	122	110	111	117
165	5.25	102	112	100	101	94	110	118	104	108	115	106	127	109	106	104	116	124	111	117	118
166	5.11	103	116	99	102	96	112	121	107	105	115	107	111	104	100	100	114	116	113	109	121
167	5.01	106	112	98	100	96	110	120	104	107	114	109	115	103	111	99	118	121	106	113	116
168	4.85	99	110	98	100	94	107	120	103	104	112	114	122	104	106	104	113	122	115	105	117
169	4.71	99	110	99	97	91	108	119	105	104	111	110	112	103	100	103	114	115	102	113	116
170	4.53	100	108	98	100	93	106	114	103	102	110	111	115	102	102	99	113	116	105	115	117
171	4.51	102	108	93	99	92	105	114	99	105	110	108	108	98	99	100	106	113	99	107	105
172	4.32	97	108	95	96	91	104	113	101	100	110	105	108	102	101	92	109	117	101	108	105
173	4.19	98	105	94	97	90	102	109	98	99	106	108	105	92	104	95	100	120	107	99	107
174	4.09	97	103	91	97	90	99	113	97	98	108	99	104	101	96	92	103	115	108	100	113
175	3.99	93	103	89	93	87	99	109	93	99	101	96	103	97	92	94	105	121	102	97	106
176	3.78	91	103	92	95	85	98	107	92	95	100	99	103	98	96	84	99	115	105	97	101
177	3.65	94	101	88	93	86	97	107	95	93	102	94	111	100	95	85	102	107	93	102	104
178	3.51	91	102	90	92	85	94	106	90	94	100	91	99	86	88	91	106	102	100	96	107
179	3.36	90	97	86	88	86	96	105	89	95	97	90	105	84	94	83	98	106	92	101	103
180	3.21	90	96	82	88	80	95	99	86	91	98	97	100	87	88	83	102	107	91	91	95
181	3.09	86	97	83	87	78	94	98	89	89	95	96	98	92	95	87	90	101	93	89	101
182	2.88	87	95	83	81	81	92	95	85	92	94	89	93	89	91	79	97	105	86	92	104
183	2.82	86	90	83	83	79	91	98	87	88	89	91	94	89	85	78	94	106	88	90	102
184	2.67	86	89	80	79	77	87	94	82	83	93	88	96	77	88	78	95	93	81	88	90
185	2.43	79	88	76	78	76	83	94	82	81	87	88	92	80	79	79	93	94	81	84	95
186	2.25	79	86	77	77	74	82	91	82	82	88	89	88	78	77	79	91	92	81	81	89
187	2.17	77	87	75	77	74	83	89	78	81	84	82	90	76	76	78	88	91	85	86	90
188	2.09	76	82	73	73	69	80	89	76	80	83	81	86	74	82	75	88	95	79	78	91
189	1.89	74	80	71	72	68	78	83	74	75	83	81	86	72	77	72	81	87	81	76	80
190	1.8	73	78	70	72	67	74	81	72	74	79	81	86	70	73	74	75	83	83	74	82
191	1.71	73	76	69	70	66	74	80	70	74	77	73	77	73	75	68	79	79	75	74	85
192	1.5	69	74	68	70	65	74	79	72	71	76	73	77	74	74	63	80	81	71	75	81
193	1.36	69	72	67	71	69	74	78	75	72	75	74	77	76	73	68	81	84	68	74	77
194	1.23	69	71	68	69	74	74	76	73	73	74	70	74	74	74	74	79	81	71	76	78
195	1.13	68	69	69	68	70	71	75	71	75	71	67	71	73	75	72	77	79	74	78	79
196	1.04	67	68	65	67	67	69	71	70	72	69	66	69	69	73	71	75	73	73	80	73
197	0.96	66	66	62	66	66	66	68	67	70	66	65	67	65	71	70	69	68	72	72	68
198	0.86	64	65	61	65	65	63	65	65	66	63	65	66	62	69	69	64	66	71	64	67
199	0.78	62	64	61	62	65	61	63	62	63	61	63	65	60	66	69	63	64	67	63	66
200	0.69	61	61	61	60	62	59	61	60	60	58	62	64	59	63	66	63	62	64	63	65
201	0.62	59	58	61	58	59	57	59	57	58	56	61	63	59	60	63	63	61	61	62	62
202	0.55	57	56	58	56	57	55	57	55	56	54	60	63	56	57	60	63	58	58	62	59
203	0.48	55	54	55	54	55	54	55	53	54	52	58	60	54	55	57	59	56	55	59	57
204	0.42	53	52	52	52	53	52	53	51	52	51	56	57	52	53	55	56	54	52	57	54
205	0.4	51	51	50	50	51	51	51	49	50	50	54	54	50	51	54	53	52	49	55	52
206	0.38	50	50	49	48	49	50	50	47	49	49	52	51	48	49	53	50	51	47	53	51
207	0.36	49	48	48	46	47	49	49	46	48	48	50	49	48	49	52	47	51	47	51	50
208	0.35	47	47	48	46	47	48	49	46	47	47	50	49	49	49	51	48	51	47	51	49
209	0.33	46	47	47	46	47	47	48	47	46	46	50	49	49	50	50	49	51	47	51	48
210	0.31	45	48	46	47	48	47	47	47	45	46	50	49	50	50	51	50	50	48	52	47
211	0.3	44	49	46	47	48	48	47	48	44	47	51	49	50	50	52	51	50	48	52	46
212	0.34	45	49	45	48	49	49	46	48	45	48	51	49	51	51	53	52	50	48	53	45
213	0.39	47	50	45	48	49	50	46	49	47	49	51	49	51	51	54	53	50	48	53	46
214	0.43	49	51	47	49	50	51	48	49	48	50	52	49	52	52	56	54	52	49	54	48
215	0.48	51	51	49	49	50	52	50	50	50	52	52	50	53	53	57	55	54	50	54	50
216	0.52	53	52	51	50	51	53	52	51	52	53	53	52	55	55	58	57	56	52	55	51
217	0.56	54	53	53	52	53	54	54	53	53	54	55	55	57	57	59	58	59	54	57	53
218	0.61	56	54	55	54	55	55	56	55	55	55	57	57	59	58	60	60	61	55	60	55
219	0.66	58	56	57	56	57	56	58	57	57	56	60	60	61	60	62	61	63	57	63	56
220	0.7	60	58	59	58	59	58	60	59	58	58	62	62	63	62	64	63	66	59	66	58
221	0.75	62	61	61	60	61	60	62	61	60	60	64	65	65	63	67	64	68	60	69	60
222	0.83	63	63	63	62	63	63	64	63	62	63	67	67	67	65	70	66	70	62	72	62
223	0.91	65	65	66	64	65	65	67	66	64	65	69	70	70	67	73	68	73	64	75	65
224	0.99	67	68	68	66	68	68	69	68	66	68	72	71	73	70	76	70	73	66	74	68
225	1.08	69	70	70	69	69	70	71	71	68	71	75	73	77	74	78	73	74	69	74	71

		MicroGreedy SpaceEqual TimeDist					MicroNoOpt SpaceEqual TimeDist					MicroGreedy SpaceDist TimeDist					MicroNoOpt SpaceDist TimeDist				
Run time	req / min	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	Macro H & L	Macro Greedy	Macro NoOpt	Macro Heur	Macro Learn	M aH & L
226	1.2	71	72	72	72	71	72	73	73	70	73	78	74	81	77	80	75	75	72	73	74
227	1.31	73	74	74	75	73	75	75	76	75	75	82	76	80	81	82	78	76	73	73	80
228	1.44	76	77	76	77	75	77	77	77	80	78	82	80	79	83	85	81	77	75	78	87
229	1.56	78	81	78	80	78	79	80	78	80	78	82	85	79	85	89	85	78	77	84	86
230	1.68	81	81	81	82	81	81	80	80	80	79	85	87	86	84	87	87	81	82	85	86
231	1.8	82	81	84	84	85	83	81	82	81	82	89	90	93	84	86	90	84	87	86	87
232	1.92	84	83	84	86	86	86	84	85	84	86	86	89	92	87	86	89	88	90	86	89
233	2.04	85	86	85	88	88	90	88	88	85	87	84	89	92	91	86	89	92	93	87	93
234	2.16	87	92	86	91	90	91	92	89	87	88	89	93	95	85	94	91	99	96	87	97
235	2.34	89	92	89	92	91	93	93	91	93	89	90	96	95	89	94	96	98	100	92	94
236	2.54	90	92	91	93	93	90	94	92	90	94	91	99	96	94	94	101	100	99	98	97
237	2.59	90	92	93	92	91	96	95	94	91	94	100	103	98	90	91	103	102	99	101	96
238	2.64	91	99	92	92	95	95	97	97	92	97	90	96	91	96	100	103	96	91	102	95
239	2.84	91	97	95	92	93	98	99	93	97	100	100	102	92	90	92	95	98	103	95	99
240	2.97	96	99	94	92	95	98	102	99	98	101	102	109	97	103	96	100	111	107	94	108
241	3.18	95	98	95	91	94	100	102	95	97	103	102	112	98	92	100	109	112	95	109	110
242	3.27	93	101	94	93	100	97	105	99	98	100	92	101	91	101	104	103	108	110	102	113
243	3.51	93	101	93	91	97	101	102	100	97	103	102	107	89	98	104	111	111	100	109	115
244	3.69	96	106	92	95	98	99	108	102	102	107	103	106	99	106	94	100	112	108	102	116
245	3.87	100	104	96	94	96	101	110	100	98	109	103	109	100	98	101	107	111	103	112	117
246	3.87	100	109	94	95	96	104	108	101	104	107	110	110	100	98	100	110	121	103	104	107
247	4.06	100	110	96	96	91	104	112	102	104	107	109	119	96	96	102	110	122	104	105	116
248	4.19	99	113	96	96	95	108	111	102	105	110	102	121	106	99	94	107	119	106	104	112
249	4.38	100	112	96	97	93	106	109	106	103	112	101	122	96	97	95	108	111	107	108	124
250	4.5	102	114	99	98	94	105	113	102	102	110	106	121	107	108	106	109	111	101	105	118
251	4.65	102	110	100	101	95	107	117	104	101	114	105	120	101	110	94	114	124	114	102	112
252	4.75	106	115	100	103	95	106	116	104	103	115	108	123	111	101	104	107	125	107	108	122
253	4.87	104	115	103	102	99	105	115	102	105	113	107	127	106	105	99	121	125	109	117	116
254	5.09	108	115	101	101	96	109	117	104	104	116	113	116	104	105	107	113	125	102	116	126
255	5.13	106	114	101	102	95	113	116	105	106	117	109	120	103	114	108	112	128	109	109	120
256	5.17	106	114	104	106	98	109	120	104	106	119	109	120	108	109	107	120	120	108	104	121
257	5.27	107	113	100	104	99	112	117	105	108	120	107	120	107	113	98	117	126	104	105	126
258	5.46	106	117	103	105	99	111	124	105	109	118	112	120	110	107	104	113	128	105	109	127
259	5.53	109	116	102	108	100	114	121	106	109	119	113	124	107	107	105	123	125	112	108	120
260	5.58	109	116	104	106	102	115	124	108	109	121	109	126	110	111	106	113	129	119	118	127
261	5.64	107	118	103	107	100	113	123	105	110	123	111	123	114	110	107	118	132	117	124	120
262	5.76	111	117	105	106	100	112	125	112	111	121	115	117	108	106	110	119	135	113	121	125
263	5.77	108	117	101	104	99	113	122	106	110	120	112	125	111	114	101	118	127	112	118	125
264	5.8	108	116	104	104	100	115	123	111	110	120	111	120	106	108	106	121	131	114	113	119
265	5.82	109	118	104	106	100	116	122	107	112	122	113	129	112	113	109	120	126	117	117	127
266	5.97	110	117	104	106	99	115	124	109	114	119	112	126	109	112	105	122	130	119	121	130
267	5.97	109	120	103	107	101	113	126	112	114	120	117	122	107	111	108	120	128	113	115	124
268	6.01	109	120	104	106	101	114	126	109	111	123	112	132	113	113	104	121	130	112	114	126
269	6	111	118	104	106	100	113	126	110	111	123	111	127	105	112	109	117	128	118	118	127
270	5.98	109	117	103	109	101	114	123	109	112	120	115	120	107	109	108	121	133	110	120	135
271	5.91	110	118	105	105	102	116	123	111	112	121	113	127	106	105	107	117	132	115	119	124
272	6	110	116	105	108	102	116	124	109	110	121	116	127	108	111	109	118	130	117	117	130
273	5.97	112	119	105	106	101	112	126	108	112	124	116	119	115	107	107	118	134	113	116	126
274	5.94	107	118	105	106	99	116	126	110	112	120	115	125	110	110	106	121	138	121	117	125
275	5.83	110	116	103	107	99	113	124	110	111	121	113	127	108	114	108	118	138	119	124	127
276	5.86	108	116	105	107	100	112	123	109	113	119	109	129	102	112	98	123	126	121	120	119
277	5.85	108	121	103	107	99	114	125	109	112	120	114	126	107	111	106	111	124	114	114	125
278	5.86	109	114	103	104	101	115	125	108	112	119	113	122	102	109	107	112	126	119	115	125
279	5.82	108	115	104	104	100	112	125	107	113	119	111	122	107	105	104	115	135	107	113	125
280	5.67	109	113	104	104	98	116	123	107	110	117	107	123	109	103	97	112	132	111	119	124
281	5.62	109	115	102	106	99	112	125	107	112	117	111	119	106	105	103	115	128	107	117	125
282	5.5	106	117	100	107	95	111	121	107	107	117	103	122	107	108	99	119	121	117	116	126
283	5.37	103	116	100	101	97	110	122	109	108	118	105	120	109	109	108	113	133	119	110	124
284	5.34	104	114	102	105	97	111	119	106	107	115	103	117	108	103	99	117	124	106	116	117
285	5.25	106	116	100	102	99	108	120	105	109	113	107	118	107	105	103	119	126	105	110	117
286	5.14	102	112	99	99	98	107	121	105	106	113	113	120	104	105	105	108	122	108	118	113
287	5.02	102	111	96	101	95	109	116	107	104	111	109	110	102	104	98	117	129	101	117	111
288	4.86	102	111	97	102	95	109	116	104	103	115	113	109	102	99	96	115	127	106	108	119
289	4.74	103	111	98	98	94	107	115	100	104	109	107	115	103	100	94	112	123	110	103	114
290	4.65	100	108	97	99	94	108	113	102	102	106	111	120	107	105	92	111	117	110	105	113
291	4.57	101	111	97	100	94	106	112	100	102	112	99	106	103	109	95	113	121	105	101	110
292	4.43	97	107	93	94	91	101	114	102	99	104	101	104	103	105	92	109	126	108	106	114
293	4.21	95	108	95	97	92	101	111	97	98	110	96	108	100	97	95	103	115	98	103	113
294	4.08	97	104	94	97	89	104	110	96	99	104	100	105	97	100	100	105	120	101	104	111
295	3.9	95	104	92	92	88	101	109	95	99	105	101	108	95	99	93	105	120	96	104	106
296	3.81	95	103	90	92	86	97	108	94	98	105	105	106	100	99	91	109	107	96	108	107
297	3.69	96	99	89	88	83	98	107	92	95	102	97	112	95	96	93	101	115	94	101	112
298	3.51	94	100	89	92	86	96	104	90	97	102	95	111	94	94	83	101	112	102	104	100
299	3.38	93	98	88	89	82	96	102	91	91	98	92	101	93	91	81	103	104	99	95	96
300	3.21	89	94	83	86	81	94	105	89	89	96	89	101	92	94	91	103	105	89	99	104

B.2 Task Allocation Protocols in TeleTruck

Results of the Vickrey Auction

		2 initial trucks					n initial trucks				
no. of orders	order set	run time	no. of trucks	cost	surplus	payments	run time	no. of trucks	cost	surplus	payments
6	r101	0.03	2	1577	1700	3277	0.03	2	1577	1700	3277
	r102	0.04	2	1149	2368	3517	0.04	2	1149	2368	3517
	r103	0.04	2	1149	2368	3517	0.04	2	1149	2368	3517
	c101	0.03	2	140	290	430	0.03	2	140	290	430
	c102	0.04	2	191	239	430	0.04	2	191	239	430
	c103	0.04	2	191	239	430	0.04	2	191	239	430
15	r101	0.1	6	5173	1805	6978	0.11	6	2270	3822	6092
	r102	0.11	5	4526	3255	7781	0.11	5	3304	3810	7114
	r103	0.1	5	4812	3390	8202	0.12	5	3304	3473	6777
	c101	0.11	2	895	1028	1923	0.11	2	895	1028	1923
	c102	0.07	2	1324	871	2195	0.07	2	1324	871	2195
	c103	0.08	2	1324	1273	2597	0.08	2	1324	1273	2597
30	r101	0.35	11	10395	2393	12788	0.51	9	4572	4699	9271
	r102	0.35	9	8300	4439	12739	0.41	8	5266	5950	11216
	r103	0.28	7	7372	5885	13257	0.45	7	4406	5716	10122
	c101	0.19	3	1695	1028	2723	0.23	4	2080	4765	6845
	c102	0.22	4	3619	2716	6335	0.33	9	3621	3316	6937
	c103	0.18	3	2639	3300	5939	0.25	3	3247	3734	6981
60	r101	1.38	17	16949	4085	21034	1.86	16	12533	9064	21597
	r102	1.16	14	14844	9800	24644	1.43	14	12281	7256	19537
	r103	1.22	12	13785	8661	22446	1.35	12	7819	9960	17779
	c101	0.73	6	5352	2012	7364	0.98	7	8024	16721	24745
	c102	0.64	7	8020	6321	14341	1.11	7	9368	20063	29431
	c103	0.87	7	8624	6299	14923	0.97	6	8235	20935	29170
90	r101	2.49	24	26585	5169	31754	3.16	24	19166	12042	31208
	r102	2.3	20	23409	15598	39007	3.32	19	18727	10561	29288
	r103	2	17	19465	13283	32748	2.68	18	13053	14964	28017
	c101	1.14	10	10936	6615	17551	2.4	12	15044	22103	37147
	c102	1.58	10	12779	13157	25936	2.14	11	14151	23160	37311
	c103	1.13	10	14259	13733	27992	1.92	10	15762	24715	40477
120	r101	4.22	27	29427	5821	35248	4.59	25	21409	13149	34558
	r102	3.63	22	25812	16328	42140	4.37	21	21437	12142	33579
	r103	3.53	21	22166	17735	39901	4.89	18	16590	17757	34347
	c101	2.51	12	13623	10882	24505	2.98	14	18010	24690	42700
	c102	2.22	12	15800	19234	35034	2.76	13	20106	29106	49212
	c103	2.71	12	19163	19327	38490	2.8	13	23683	27845	51528

Results of the MA-2 Auction

		2 initial trucks					n initial trucks				
no. of orders	order set	run time	no. of trucks	cost	surplus	payments	run time	no. of trucks	cost	surplus	payments
6	r101	0.05	2	1753	2199	3952	0.05	2	1753	2199	3952
	r102	0.05	2	1753	2199	3952	0.05	2	1753	2199	3952
	r103	0.05	2	1753	2199	3952	0.05	2	1753	2199	3952
	c101	0.05	2	180	300	480	0.05	2	180	300	480
	c102	0.06	2	185	300	485	0.06	2	185	300	485
	c103	0.07	2	185	300	485	0.07	2	185	300	485
15	r101	0.28	6	5481	6030	11511	0.36	6	4406	5157	9563
	r102	0.25	5	4937	5504	10441	0.42	4	3401	4929	8330
	r103	0.2	3	3038	3636	6674	0.34	4	3401	5540	8941
	c101	0.16	2	1404	1842	3246	0.16	2	1404	1842	3246
	c102	0.13	2	1404	1842	3246	0.13	2	1404	1842	3246
	c103	0.18	2	1404	1842	3246	0.18	2	1404	1842	3246
30	r101	1.33	11	10589	12605	23194	1.5	10	6914	11390	18304
	r102	1.35	11	10589	12605	23194	1.65	11	5664	9682	15346
	r103	0.82	7	8109	9288	17397	1.01	7	5040	9999	15039
	c101	0.51	3	1735	3221	4956	0.51	3	1762	4986	6748
	c102	0.48	3	2204	3165	5369	0.64	4	3248	5490	8738
	c103	0.42	3	2674	3798	6472	0.68	3	3197	5325	8522
60	r101	3.88	17	15306	20548	35854	4.36	16	14400	25024	39424
	r102	3.02	13	14606	21785	36391	3.18	13	11761	18040	29801
	r103	2.47	10	13429	17276	30705	3.28	12	10250	19407	29657
	c101	1.36	6	5392	8412	13804	1.65	6	5332	19886	25218
	c102	1.5	7	7022	8516	15538	2.08	7	7497	15364	22861
	c103	1.53	7	9515	13066	22581	2.23	7	9066	16991	26057
90	r101	7.57	24	22149	31215	53364	8.62	23	21952	34915	56867
	r102	6.95	21	23831	34404	58235	7.36	20	19288	30542	49830
	r103	5.49	18	19325	26237	45562	6.58	18	15822	28943	44765
	c101	3.2	10	10973	17427	28400	4.51	10	10528	32526	43054
	c102	3.09	10	13260	15615	28875	4.69	11	13670	26712	40382
	c103	3.15	10	15499	23642	39141	4.96	10	14861	32655	47516
120	r101	13.7	24	25507	35998	61505	-	-	-	-	-
	r102	15.7	24	25507	35998	61505	-	-	-	-	-
	r103	9.75	20	22639	32212	54851	-	-	-	-	-
	c101	4.43	12	13791	22200	35991	-	-	-	-	-
	c102	4.9	13	18443	24438	42881	-	-	-	-	-
	c103	5.48	12	20268	30238	50506	-	-	-	-	-

Results of the MA-3 Auction

		2 initial trucks					n initial trucks				
no. of orders	order set	run time	no. of trucks	cost	surplus	payments	run time	no. of trucks	cost	surplus	payments
6	r101	0.17	2	1577	1176	2753	0.17	2	1577	1176	2753
	r102	0.11	2	1149	1176	2325	0.11	2	1149	1176	2325
	r103	0.14	2	1149	1176	2325	0.14	2	1149	1176	2325
	c101	0.12	1	140	127	267	0.12	1	140	127	267
	c102	0.11	1	145	122	267	0.11	1	145	122	267
	c103	0.11	1	145	122	267	0.11	1	145	122	267
15	r101	0.77	6	4377	2037	6414	0.81	5	2987	1682	4669
	r102	0.54	4	3807	2117	5924	0.54	4	3816	2364	6180
	r103	0.52	4	3804	1466	5270	0.54	4	3813	1713	5526
	c101	0.27	2	1505	295	1800	0.27	2	1505	295	1800
	c102	0.39	2	1364	320	1684	0.39	2	1364	320	1684
	c103	0.3	2	1364	320	1684	0.3	2	1364	320	1684
30	r101	2.7	11	9222	4064	13286	3.31	9	5959	3096	9055
	r102	2.2	8	7996	2842	10838	2.08	7	4873	4377	9250
	r103	1.4	6	6414	3810	10224	1.67	6	2452	2761	5213
	c101	0.92	4	3362	932	4294	0.59	4	2860	1304	4164
	c102	0.73	3	2186	899	3085	0.66	4	2918	1552	4470
	c103	0.55	3	2897	725	3622	1.19	4	3216	1739	4955
60	r101	9.58	16	14244	7660	21904	8.84	15	13325	6565	19890
	r102	7.16	13	13982	7578	21560	9.79	13	11000	7709	18709
	r103	5.52	11	13407	8287	21694	10.06	11	11000	7709	18709
	c101	3.46	7	6882	932	7814	5.08	8	8659	8901	17560
	c102	2.82	6	5922	2648	8570	4.49	7	6936	8899	15835
	c103	3.16	6	6732	2336	9068	4.78	6	6511	6934	13445
90	r101	19.09	22	23043	10226	33269	11.67	11	20952	9626	30578
	r102	28.03	22	22842	10456	33298	30.66	22	20474	10898	31372
	r103	13.61	16	18139	10885	29024	21.38	17	15994	10771	26765
	c101	7.08	10	10207	1748	11955	11.33	11	12874	8786	21660
	c102	7.56	10	10233	4018	14251	12	10	10572	18410	28982
	c103	4.7	9	12793	4873	17666	12.15	9	12676	16432	29108
120	r101	67.84	25	26421	11067	37488	90.55	26	23637	10473	34110
	r102	76.24	24	25993	12308	38301	84.64	24	23392	13175	36567
	r103	37.76	18	20487	12961	33448	49.56	19	17612	13224	30836
	c101	12.12	12	12531	3959	16490	20.17	13	14957	11557	26514
	c102	11.95	12	16378	6917	23295	20.99	13	16091	22646	38737
	c103	11.25	11	18416	7535	25951	20.66	12	18152	17906	36058

Results of the MA-4 Auction

		2 initial trucks					n initial Trucks				
no. of orders	order set	run time	no. of trucks	cost	surplus	payments	run time	no. of trucks	cost	surplus	payments
6	r101	0.24	2	1577	970	2547	0.24	2	1577	970	2547
	r102	0.23	2	1149	1128	2277	0.23	2	1149	1128	2277
	r103	0.39	2	1149	1128	2277	0.39	2	1149	1128	2277
	c101	0.2	1	159	135	294	0.2	1	159	135	294
	c102	0.35	1	175	85	260	0.35	1	175	85	260
	c103	0.2	1	175	85	260	0.2	1	175	85	260
15	r101	1.81	6	4377	2974	7351	1.81	4	2918	1122	4040
	r102	1.11	4	3854	2035	5889	1.06	4	3454	1862	5316
	r103	0.9	4	3854	2035	5889	1.32	4	3454	1862	5316
	c101	0.34	3	108	312	420	1.08	3	1052	350	1402
	c102	0.82	3	1135	251	1386	0.921	3	1044	358	1402
	c103	0.83	3	1135	251	1386	1	3	1044	358	1402
30	r101	4.75	8	7066	3466	10532	6.54	7	4480	3685	8165
	r102	2.41	6	6793	4114	10907	4.99	7	6172	5361	11533
	r103	3.2	6	7659	3229	10888	4.96	6	4406	3599	8005
	c101	5.48	7	6172	5361	11533	7.25	6	2492	852	3344
	c102	1.94	4	2501	680	3181	2.29	4	2901	744	3645
	c103	1.72	3	2869	959	3828	2.31	3	2923	642	3565
60	r101	23.78	14	14794	8619	23413	3.28	15	10822	8431	19253
	r102	28.6	14	14351	8365	22716	57.96	15	10464	6140	16604
	r103	16.52	12	15820	7112	22932	33.32	11	9172	9016	18188
	c101	7.4	7	6808	2295	9103	14.34	8	7009	4225	11234
	c102	0.8	7	7011	2022	9033	14.61	7	7681	5442	13123
	c103	6.79	6	7575	3693	11268	1.28	7	7217	3540	10757
90	r101	85.38	22	22996	14327	37323	114.4	23	19130	13140	32270
	r102	82.11	19	21768	15097	36865	262.5	23	19996	8492	28488
	r103	111.8	18	20876	11660	32536	60.4	15	15234	11932	27166
	c101	15.29	10	10157	4672	14829	45.39	11	12535	11469	24004
	c102	17.6	10	10697	2501	13198	48.5	10	12035	12039	24074
	c103	16.86	10	14594	5704	20298	5.62	10	10082	10817	20899
120	r101	974.3	25	26829	15238	42067	1003	25	21739	14046	35785
	r102	410.4	22	24523	17337	41860	1063	25	22795	10975	33770
	r103	254.2	20	24303	14810	39113	10.49	20	22639	9573	32212
	c101	47.81	12	13409	6743	20152	79.37	13	15150	13103	28253
	c102	38.12	12	15424	8569	23993	68.69	12	17868	14946	32814

Results of the MA-5 Auction

		2 initial trucks					n initial trucks				
no. of orders	order set	run time	no. of trucks	cost	surplus	payments	run time	no. of trucks	cost	surplus	Payments
6	r101	0.64	2	1577	411	1988	0.64	2	1577	411	1988
	r102	0.68	2	1614	197	1811	0.68	2	1614	197	1811
	r103	0.73	2	1614	197	1811	0.73	2	1614	197	1811
	c101	0.43	2	140	1	138	0.43	2	140	1	138
	c102	0.69	1	140	1	128	0.69	1	140	1	128
	c103	0.75	1	140	1	128	0.75	1	140	1	128
15	r101	4.82	5	4639	1829	6468	6.23	6	3170	3232	6402
	r102	5.38	5	4403	2047	6450	0.66	4	3322	2087	5409
	r103	4.23	4	4715	1280	5995	3.96	4	3955	1781	5736
	c101	1.88	2	895	357	1252	1.88	2	895	357	1252
	c102	2.66	3	977	193	1170	2.66	3	977	193	1170
	c103	0.26	2	1006	66	1072	0.26	2	1006	66	1072
30	r101	26.14	11	9254	3073	12327	77.78	10	5280	4586	9866
	r102	13.77	8	8204	2800	11004	32.29	8	4686	4586	9272
	r103	10.37	6	8117	1539	9656	32.29	8	4686	4586	9272
	c101	3.65	3	1695	923	2618	4.24	3	2064	1010	3074
	c102	5.67	4	2906	1589	4495	4.94	3	2458	1392	3850
	c103	4.45	3	3465	70	3535	5.83	3	3339	622	3961
60	r101	14.52	15	13945	6394	20339	245.1	16	11999	7771	19770
	r102	96.6	13	13134	4427	17561	264	14	11518	7957	19475
	r103	97.59	13	13134	4427	17561	298.7	8	7834	3810	11644
	c101	20.92	7	6237	3039	9276	61.21	7	6558	6546	13104
	c102	33.58	7	6758	2023	8781	52.9	6	6991	6573	13564
	c103	22.2	6	7169	810	7979	44.9	7	9774	4581	14355
90	r101	928.8	24	22281	10417	32698	2002	23	18489	11321	29810
	r102	231.5	21	19558	12339	31897	2470	22	20729	13020	33749
	r103	552.4	17	20302	9588	29890	907.8	22	15380	11798	27178
	c101	58.75	10	10296	6669	16965	254.5	11	12111	11105	23216
	c102	68.65	9	13034	2228	15262	70.29	9	13034	2228	15262
	c103	91.84	11	12488	2350	14838	390.7	10	15273	8463	23736
120	r101	19402	26	24623	12013	36636	14516	25	21202	12582	33784
	r102	-	-	-	-	-	-	-	-	-	-
	r103	-	-	-	-	-	-	-	-	-	-
	c101	317.6	12	13201	7932	21133	525.4	13	15013	12436	27449
	c102	251.6	13	17655	5221	22876	513.7	13	18570	14021	32591

Results of the Simulated Trading Procedure

		2 initial trucks			n initial trucks		
no. of orders	order set	run time	no. of trucks	cost	run time	no. of trucks	cost
6	r101	0.27	2	1577	0.27	2	1577
	r102	0.2	2	1149	0.2	2	1149
	r103	0.23	2	1149	0.23	2	1149
	c101	0.2	2	140	0.2	2	140
	c102	0.2	2	140	0.2	2	140
	c103	0.28	2	140	0.28	2	140
15	r101	1.04	5	4380	1.31	6	2973
	r102	0.81	4	3960	0.91	4	3960
	r103	0.84	4	3960	0.85	4	3960
	c101	0.54	2	895	0.54	2	895
	c102	0.6	2	1712	0.6	2	1712
	c103	0.49	2	1712	0.49	2	1712
30	r101	8	10	8573	6	10	4227
	r102	5	8	7388	6	8	5318
	r103	4	7	6562	4	7	4400
	c101	3	3	1717	3	4	2592
	c102	3	4	2597	3	3	3094
	c103	3	3	3166	2	3	2970
60	r101	28	14	12205	23	14	10650
	r102	16	12	10594	18	13	9912
	r103	14	10	9457	19	10	6946
	c101	12	7	6121	14	8	7930
	c102	9	8	8947	15	9	9530
	c103	10	6	6523	11	7	9388
90	r101	74	21	18085	88	22	16701
	r102	52	18	14556	48	20	16538
	r103	27	14	13537	31	15	11240
	c101	17	10	10608	22	12	14094
	c102	17	10	12483	21	11	16869
	c103	21	9	11879	28	11	12857
120	r101	115	22	20300	175	23	17540
	r102	75	20	18644	97	21	17123
	r103	58	16	15668	58	16	12726
	c101	29	12	12931	36	14	16982
	c102	29	13	19595	30	13	20519
	c103	30	13	16434	30	13	23018

B.3 Geographical Structurizations in TELETRUCK

Short-distance Settings

	Run time (msec)	Travel time (h)	Travel dist. (km)	Idle time (h)
Hamburg				
5 orders	413	476	409	11377
10 orders	1187	653	508	11493
15 orders	1750	959	747	11384
20 orders	1766	1076	825	11267
25 orders	2274	1187	1053	11794
30 orders	4896	1284	1184	11865
Hanover				
5 orders	438	378	312	11568
10 orders	1297	587	519	11359
15 orders	1453	750	645	11996
20 orders	1765	1012	856	11734
25 orders	2360	1375	1134	11371
30 orders	4954	1501	1231	11245
Berlin				
5 orders	500	623	588	8642
10 orders	1398	835	602	9655
15 orders	1672	1090	652	10644
20 orders	1969	1352	790	10382
25 orders	2255	1875	1011	9858
30 orders	4344	1905	1075	9829
Saarbrücken				
5 orders	234	253	219	5603
10 orders	1641	359	326	7365
15 orders	625	484	440	9085
20 orders	1578	504	494	11526
25 orders	1875	491	508	11539
30 orders	1343	582	586	11448
Munich				
5 orders	478	288	266	5893
10 orders	1286	318	378	6187
15 orders	1812	384	413	6363
20 orders	1703	468	528	6279
25 orders	2034	712	809	7555
30 orders	4203	829	929	7438
Centralized				
25 orders	11860	902	823	451
50 orders	34579	1011	924	1661
75 orders	76781	1642	1225	2645
100 orders	63906	1511	1318	3663
125 orders	92672	1771	1356	4442
150 orders	92547	1979	1433	4934

Medium-distance Settings

	Run time (msec)	Travel time (h)	Travel dist. (km)	Idle time (h)
Hamburg				
5 orders	407	320	327	941
10 orders	1188	772	775	2277
15 orders	1140	1239	1217	3893
20 orders	2453	1500	1529	6195
25 orders	1953	2945	1911	6719
30 orders	4015	3231	2210	9866
Hanover				
5 orders	266	275	253	599
10 orders	1188	318	342	692
15 orders	1766	1328	1343	8323
20 orders	2031	2349	1411	7302
25 orders	2594	2684	1798	6967
30 orders	3797	2893	2036	6758
Berlin				
5 orders	406	437	394	1198
10 orders	1219	646	578	2693
15 orders	1218	1075	959	4381
20 orders	2516	1362	1151	5404
25 orders	2203	1775	1464	8244
30 orders	4484	2235	2883	9376
Saarbrücken				
5 orders	907	462	536	9234
10 orders	1078	454	536	8441
15 orders	812	1044	1224	11340
20 orders	1500	974	1128	8722
25 orders	2938	823	973	11561
30 orders	2953	792	909	10452
Munich				
5 orders	515	362	366	895
10 orders	844	675	680	2794
15 orders	1860	979	1013	4277
20 orders	1782	1326	1474	5516
25 orders	2047	1712	1817	5960
30 orders	3969	2069	2228	8614
Centralized				
25 orders	16888	1055	1042	369
50 orders	44672	1249	1136	1409
75 orders	60703	1500	1158	2687
100 orders	92265	1772	1422	5944
125 orders	139870	2099	1547	5830
150 orders	233469	2354	1676	6795

Long-distance Settings

	Run time (msec)	travel time (h)	Travel dist. (km)	Idle time (h)
Hamburg				
5 orders	313	326	483	2694
10 orders	656	398	517	2974
15 orders	1484	454	565	3226
20 orders	1922	1441	948	4347
25 orders	3047	2260	1571	5704
30 orders	4109	2585	1917	5065
Hanover				
5 orders	468	555	593	630
10 orders	1093	1037	1145	1825
15 orders	1250	1197	1295	2281
20 orders	2359	1630	1718	3917
25 orders	2250	2096	2162	5461
30 orders	3922	2587	2619	7904
Berlin				
5 orders	328	238	165	297
10 orders	1031	1572	447	1444
15 orders	1000	2091	924	2306
20 orders	2422	2524	1303	4207
25 orders	2094	3525	2158	7115
30 orders	4156	3770	2366	7595
Saarbrücken				
5 orders	375	815	920	5692
10 orders	703	887	1027	7793
15 orders	1234	1510	875	2438
20 orders	750	2422	1741	8198
25 orders	2203	2400	1781	10173
30 orders	2672	2102	1387	4487
Munich				
5 orders	407	611	660	1785
10 orders	1219	2090	1158	2957
15 orders	1156	2400	1501	3440
20 orders	2485	2750	1914	4189
25 orders	2047	2909	2068	4829
30 orders	4422	3453	2701	7631
Centralized				
25 orders	12438	1201	1058	412
50 orders	35069	1056	829	632
75 orders	47551	2402	1745	4957
100 orders	81972	2383	1771	5633
125 orders	79577	2409	1787	5687
150 orders	149670	2238	1701	6116

B.4 Topology Modules in IFMS

	Produced work pieces per station	Used capacity per station	Produced work pieces per functionality	Used capacity per functionality
Chain				
0% Fault time	10.4	100	10.4	100
6% Fault time	7.4	80	7.4	80
12% Fault time	6	68	6	68
18% Fault time	5.3	59	5.3	59
Scope				
0% Fault time	10.1	100	5.5	76.9
6% Fault time	7.4	87	4	66.8
12% Fault time	6.1	79	3.3	59.9
18% Fault time	6	70	3.2	53.1
Parallelism (3;4)				
0% Fault time	10.3	100	8	88.9
6% Fault time	7.7	84	6	74.7
12% Fault time	6.4	74	5	65.5
18% Fault time	5.7	65	4.4	58.3
Parallelism (3;4;5)				
0% Fault time	10.3	100	5.5	76.9
6% Fault time	7.9	83	4.2	55.9
12% Fault time	6.9	69	3.7	52.5
18% Fault time	6.1	66	3.3	50
Parallelism (1;2 - 5;6)				
0% Fault time	10.3	100	6.5	81.8
6% Fault time	7.7	82	4.9	67
12% Fault time	6.9	74	4.4	53.8
18% Fault time	5.9	65	3.7	53.6
Parallelism (2;3;4 - 5;6;7)				
0% Fault time	10	100	3.7	68.4
6% Fault time	8.3	87	3.1	58.6
12% Fault time	6.7	78	2.5	52.4
18% Fault time	5	70	1.8	46.6

	Produced work pieces per station	Used capacity per station	Produced work pieces per functionality	Used capacity per functionality
Multi-station				
0% Fault time	6	71	0.9	40.6
6% Fault time	4	67	0.6	38.3
12% Fault time	4	62	0.6	35.4
18% Fault time	3	56	0.4	32
Double multi-station				
0% Fault time	9	72	1.3	45.1
6% Fault time	7.5	68	1.1	38.9
12% Fault time	7.5	65	1.1	37.4
18% Fault time	6.5	61	0.9	34.9
Triple multi-station				
0% Fault time	8.7	72	1.2	41
6% Fault time	8	66	1.1	37.7
12% Fault time	7.3	62	1	35.4
18% Fault time	6.3	57	0.9	32.4
Chain with multi-station				
0% Fault time	10	100	5.7	78.6
6% Fault time	8.1	86	4.6	70.6
12% Fault time	6.8	77	3.9	65.1
18% Fault time	5.8	66	3.3	59.3
Scope with multi-station				
0% Fault time	9.9	100	4	70
6% Fault time	7.9	88	3.2	62.9
12% Fault time	7.3	83	2.9	60.2
18% Fault time	6	77	2.4	56.5

Bibliography

[AD98] F. Aeken and Y. Demazeau. Minimal multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems*, 1998.

[AFHS95] O. Arnold, W. Faisst, M. Härtling, and P. Sieber. Virtuelle Unternehmen als Unternehmenstyp der Zukunft? In *Handbuch der modernen Datenverarbeitung*, volume 185 of *Theorie und Praxis der Wirtschaftsinformatik*. Heidelberg: Hüthig-Verlag, 1995.

[AMO93] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows*. Prentice-Hall, 1993.

[Aus62] J. L. Austin. *How to do Things with Words*. Oxford University Press, 1962.

[Axe84] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[BB89] G. Bamberg and F. Baur. *Statistik*. Oldenbourg, München Wien, 6th edition, 1989.

[BFG99a] T. Bohnenberger, K. Fischer, and C. Gerber. An agent-based approach for production scheduling and plant topology optimization in manufacturing. In *Proceedings of the Fourth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents (PAAM)*, 1999.

[BFG99b] T. Bohnenberger, K. Fischer, and C. Gerber. Agents in manufacturing: Online scheduling and production plant configuration. In *Proceedings of the 1st International Symposium on Agent Systems and Applications (ASA/MA)*, 1999.

[BFV98a] H.-J. Bürckert, K. Fischer, and G. Vierke. TeleTruck: A holonic fleet management system. In *Proceedings of the 14th European Meeting on Cybernetics and Systems Research*, 1998.

[BFV98b] H.-J. Bürckert, K. Fischer, and G. Vierke. Transportation scheduling with holonic MAS – the TeleTruck approach. In *Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM)*, 1998.

[BHM92] A. Bachem, W. Hochstättler, and M. Malich. Simulated Trading: A New Approach For Solving Vehicle Routing Problems. Technical Report 92.125, Mathematisches Institut der Universität zu Köln, Dezember 1992.

[BM99] K. Barber and C. Martin. Applying dynamic planning frameworks to agent goals. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[Bra87] M. E. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Mass., 1987.

[BRJ97] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User guide.* Addison Wesley Longman, 1997.

[Cas90] C. Castelfranchi. Social power - a point missed in multi-agent, DAI and HCI. In Y. Demazeau and J. Müller, editors, *Decentralized A. I.* North Holland, 1990.

[Cas95] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecure. In M. Wooldridge and N.R. Jennings, editors, *Intelligent Agents: Theories, Architectures and Languages*, volume LNAI Volume 890, pages 45–70. Springer-Verlag, Heidelberg, Germany, 1995.

[Cd96] B. Chaib-draa. Interaction between agents in routine, familiar and unfamiliar situations. *International Journal of Intelligent and Cooperative Information Systems*, 5(1), 1996.

[CDC99] A. Cesta, D. D'Aloisi, and M. Collia. Adjusting autonomy of agent systems. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[CL90] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. In *Artificial Intelligence*, volume 42, pages 213–261. 1990.

[CML93] M. Cini, R. Moreland, and J. Levine. Group staffing levels and responses to prospective and new group members. *Journal of Personality and Social Psychology*, 65:723–734, 1993.

[CMT79] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In *Combinational Optimizations.* John Wiley & Sons, 1979.

[DB88] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[DBK+98] G. Dorais, R. Bonasso, D. Kortenkamp, P. Pell, and D. Schreckenghost. Adjustable autonomy for human-centered autonomous systems on Mars. In *Proceedings of the Mars Society Conference*, 1998.

[DD96] W. Domschke and A. Drexl. *Logistik: Standorte.* Oldenbourg, Munich, Vienna, 1996.

[Dee94] S. M. Deen. A cooperation framework for holonic interactions in manufacturing. In S. M. Deen, editor, *Proceedings of the Second Internatinal Working Conference on Cooperating Knowledge-Based Systems.* DAKE Centre, University of Keele, 1994.

[DGMT95] J. Doran, N. Gilbert, U. Müller, and K. Troitsch. Social science microsimulation: A challenge to computer science. Technical Report 12, Dagstuhl-Seminar Report, 1995.

[DP95] J. Doran and M. Palmer. The EOS project: integrating two models of palaeolihic social change. In N. Gilbert and R. Conte, editors, *Artificial Societies*, pages 103 –125. VCL Press, 1995.

[Esh98] K. Eshghi. Abductive planning with event calculus. In *Procedings of the Fifth International Conference on Logic Programming*, pages 562–578, 1998.

[Fal95] J. Falk. *Ein Multi-Agentensystem zur Transportplanung und -steuerung bei Speditionen im Trampverkehr.* PhD thesis, Friedrich-Alexander-Universität, Erlangen, Nürnberg, 1995.

[FC99] R. Falcone and C. Castelfranchi. Levels of delegation and levels of help for agents with adjustable autonomy. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[FF94] T. Finin and R. Fritzson. KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the 13th Intl. Distributed Artificial Intelligence Workshop*, pages 127–136, Seattle, WA, USA, 1994.

[FG98] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems*, 1998.

[FGCd$^+$99] K. Fischer, C. Gerber, B. Chaib-draa, J.P. Müller, and M. Pischel. A simulation approach based on negotiation and cooperation between agents: A case study. *IEEE Journal on transaction on Modeling and Computer Simulation–Simulation of Scalable Systems*, 1999.

[FGLS98] P. Funk, C. Gerber, J. Lind, and M. Schillo. SIF: An agent-based simulation toolbox using the EMS paradigm. In *Proceedings of the 3rd International Congress of the Federation of EUROpean SIMulation Societies (EuroSim)*, 1998.

[FIP98] FIPA 98 Specification. volume 1-13. The Foundation for Intelligent Physical Agents, 1998. Version 1.0.

[Fis94] K. Fischer. The design of an intelligent manufacturing system. In S. M. Deen, editor, *Proceedings of the 2nd Intl. Working Conference on Cooperating Knowledge-based Systems (CKBS) (Selected Papers)*, pages 83–99. DAKE Centre, 1994.

[FMP96] K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling: an application domain for DAI. *Journal of Applied Artificial Intelligence. Special issue on Intelligent Agents*, 10(1), 1996.

[Fre87] J. Free. *Pheromones of Social Bees.* Chapman and Hall, London, 1987.

[FRV98] K. Fischer, C. Ruß, and G. Vierke. Decision theory and coordination in multiagent systems. Research Report RR-98-02, DFKI, 1998.

[Gar83] M. Gardner. *Wheels, Life and other Mathematical Amusements.* W.H. Freeman and Company, 1983.

[Ger97] C. Gerber. An artificial agent society is more than a collection of “social” agents. In *Socially Intelligent Agents - Papers from the 1997 AAAI Fall Symposium.* Technical Report FS-97-02, AAAI, 1997.

[Ger98a] C. Gerber. Bottleneck analysis for self-adapting multi-agent societies. In *Proceedings of the IEEE Joint Conference on the Science and Technology of Intelligent Systems*, 1998.

[Ger98b] C. Gerber. Evolution-based self-adaption as an expression for the autonomy degree in multi-agent systems. In *Proceedings of the IEEE Joint Conference on the Science and Technology of Intelligent Systems*, 1998.

[Ger99] C. Gerber. Performance optimization in the MoTiV-PTA agent system. In *Proceedings of the First International Symposium on Impact of agent Technology on telecommunications*, 1999. Invited paper.

[GI98] C. Gerber and P. Imhof. Generalised media of interaction and abstract resources: Two concepts on social control derived in sociology and computer science. In *Proceedings of the Workshop on Socionics at the German Conference for Artificial Intelligence (KI'98)*, 1998.

[GJ97] C. Gerber and C. G. Jung. Towards the bounded optimal agent society. In K. Fischer, C. G. Jung, and S. Schacht, editors, *Working Notes of the KI´97 Workshop on Distributed Cognitive Systems*, 1997.

[GJ98] C. Gerber and C. Jung. Resource management for boundedly optimal agent societies. In *Proceedings of the ECAI Workshop on Monitoring and Control of Real-Time Intelligent Systems*, 1998.

[Goo71] I. Good. Twenty-seven principles of rationality. In V. Godambe and D. Sprott, editors, *Foundations of Statistical Inference.* Holt, Rinehart, and Winston, Toronto, 1971.

[Gor69] G. Gordon. *System Simulation.* Prendice-Hall, Englewoods Cliff, NJ, 1969.

[GR97] C. Goldman and J. Rosenschein. Evolving organizations of agents. In *Proceedings of the Workshop on Machine Learning at the fourteenth National Conference on AI*, 1997.

[GRHL89] L. Gasser, N. Rouquette, R. Hill, and J. Lieb. Representing and using organizational knowledge in distributed ai systems. In L. Gasser and M. Huhns, editors, *Distibuted Artificial Intelligence*, volume II. Pitman, 1989.

[GRV99] C. Gerber, C. Ruß, and G. Vierke. On the suitability of market-based mechanisms for telematics applications. In *Proceedings of the International Conference on Autonomous Agents (Agents'99)*, 1999.

[GS98] I. Gilboa and D. Schmeidler. Case-based decision: An extended abstract. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI)*, 1998.

[GSB99] C. Gerber, D. Steiner, and B. Bauer. Resource adaptation for a scalable agent society in the MoTiV-PTA domain. In A. Hazelden and J. Bigham, editors, *Software Agents for Future Communication Systems.* Springer Verlag, 1999.

[GSV99a] C. Gerber, J. Siekmann, and G. Vierke. Flexible autonomy in holonic multi-agent systems. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[GSV99b] C. Gerber, J. Siekmann, and G. Vierke. Holonic multi-agent systems. Research Report RR-99-01, German Research Center for Artificial Intelligence, 1999.

[GSW98] P. Gomber, C. Schmidt, and C. Weinhardt. Efficiency incentives and computational tractability in the coordination of multi-agent systems. In *Proceedings of the Workshop Kooperationsnetze und Elektronische Koordination*, 1998.

[Har93] J. Harrington. Computer integrated manufactoring. Technical report, Malabar, Florida, 1993. Reprint.

[Hay98] A. Hayzelden. Telecommunications multi-agent control system (TeleMACS). In *Procedings of the European Conference on Artificial Intelligence (ECAI)*, 1998.

[HLT99] H. Hexmoor, M. Lafary, and M. Trosen. Adjusting autonomy by introspection. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[Hor87a] R. Horst. Nichtlineare Programmierung. In *Grundlagen des Operations Research.* Springer, 1987.

[Hor87b] E. Horvitz. Reasonig about beliefs and actions under computational resource constraints. In *Proceedings of the Third Workshop on Uncertainty in Artificial Intelligence*, 1987.

[HS98] M. Huhns and M. Singh. Agents and multiagent applications: themes, approaches, and challenges. In M. Huhns and M.Singh, editors, *Readings in Agents*, pages 1–23. Morgan Kaufmann, San Fransisco, CA, 1998.

[Hub99] M. Huber. Considerations for flexible autonomy within BDI intelligent agent architectures. In *AAAI Spring Symposium on Agents with Adjustable Autonomy*, 1999.

[HZ96] E. Hansen and S. Zilberstein. Monitoring anytime algorithms. *SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling*, 1996.

[IEE81] IEEE. *Transactions on Systems, Man and Cybernetics*, volume 11, 1981.

[Jam97] A. Jameson. Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen. In R. Schäfer and M. Bauer, editors, *Modeling the user's processing resources: Pragmatic simplicity meets psychological complexity*, pages 149–160. Universität des Saarlandes, 1997.

[JB98] A. Jameson and K. Buchholz. Einleitung zum Themenheft "Ressourcenadaptive kognitive Prozesse". *Kognitionswissenschaft*, 7(3):95–100, 1998.

[JL85] H. Juel and R. Lowe. The facility location problem for hyperrectilinear distances. *IEEE Transactions*, 17:94 – 98, 1985.

[JLG+99] C. Jung, J. Lind, C. Gerber, M. Schillo, P. Funk, and A. Burt. An architecture for co-habited virtual worlds. In *Proceedings of the Virtual Worlds and Simulation Conference*, 1999. Invited paper.

[JLG+00] C. Jung, J. Lind, C. Gerber, M. Schillo, P. Funk, and A. Burt. Eine integrierte Systemarchitektur für Agenten und Benuter in virtuellen Welten. *KI - Künstliche Intelligenz; Schwerpunktthema Intelligente Virtuelle Umgebungen*, 2000. Invited paper, to appear.

[Jun98] C. G. Jung. Experimenting with layered, resource-adapting agents in the robocup simulation. In *Proceedings of the ROBOCUP'98 Workshop*, 1998.

[Jun99] C. Jung. *Theory and Practice of Hybrid Agents.* PhD thesis, Universität des Saarlandes, Saarbrücken, 1999.

[JW98] N. Jennings and M. Wooldridge. *Agent Technology: Foundations, Applications, and Markets.* Springer Verlag, 1998.

[KAK+97] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of The First International Conference on Autonomous Agents (Agents'97)*, Marina del Ray, 1997. The ACM Press.

[KE96] A. Klein and H. Eckardt. Engpaßanalyse - eine neue Methode zur Systembewertung. Research report, Siemens AG, Zentrabteilung SW2, 1996.

[Ket93] S. Ketchpel. Coalition forming among autonomous agents. In *Proceedings of the 5th European Workshop on Modelling Autonomous Agents and Multi Agent Worlds (MAAMAW'93)*, Neuchatel, Switzerland, August 1993.

[Koe67] A. Koestler. *The Ghost in the Machine.* Hutchinson, 1967.

[Lan98] D. Lange. Mobile agents: Environments, technologies, and applications. In *Proceedings of the Third International Conference on Practical Applications of Intelligent Agents and Multiagents (PAAM)*, 1998.

[Ld95] M. Luck and M. d'Inverno. A formal framework for agents and autonomy. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[LJG99] J. Lind, C. Jung, and C. Gerber. Adaptivity and learning in intelligent real-time systems. In *Proceedings of the International Conference on Autonomous Agents (Agents'99)*, 1999.

[Mey90] R. Meyer. *Meyers Konversationslexikon*, volume 14. Verlag des Bibliographischen Instituts, 4th edition, 1890.

[MMT88] C. Ma, J. Moore, and S. Turnbull. Stopping agents from "cheating". In *Journal of Economic Theory 46: pp. 355-372*, 1988.

[Mor96] R. Moreland. Creating the ideal group: Composition effects at work. In E. Witte and J. Davis, editors, *Understanding Groupd Behavior*, volume 2. Lawrence Erlbaum Associates, Publishers, Mahway, NJ, 1996.

[MP82] J. Müller and H. Peters. *Einführung in die Volkswirtschaftslehre.* Verlag Neue Wirtschafts-Briefe, 1982.

[Mül96] J. P. Müller. *An Architecture for Dynamically Interacting Agents.* PhD thesis, Universität des Saarlandes, Saarbrücken, 1996.

[Nag61] E. Nagel. *The structure of science.* Harcourt, Brace and World, 1961.

[Nas88] D. Nasser. How to run a focus group. *Public Relations Journal*, 44:33–34, 1988.

[Neu92] O. Neumann. Theorien der Aufmerksamkeit: von Methaphern zu Mechanismen. *Psychologische Rundschau*, 43:81–103, 1992.

[Num92] C. Numaoka. Conversation for organizatorial activity. In E. Werner and Y. Demazeau, editors, *Decentralized A. I.*, volume 3. North Holland, 1992.

[NW88] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Programming.* Wiley, New York, 1988.

[Par69] T. Parsons. *Politics and Social Structures.* Free Press, New York, 1969.

[Pic93] A. Picot. Organisationsstruktur in der Wirtschaft und ihre Anforderungen an die Informations- und Kommunikationstechnik. In *Handbuch des Informationsmanagements, Aufgaben - Konzepte - Praxislösungen* , pages 49–68. A.W. Scheer, 1993.

[RCA98] R. Rabelo, L. M. Camarinha, and H. Afsarmanesh. Multiagent perspectives to agile scheduling. In L. M. Camarinha, H. Afsarmanesh, and V. Marik, editors, *Intelligent Systems for Manufacturing*, pages 51–66. Kluwer Adademic Publishers, 1998.

[RG91] A. S. Rao and M. P. Georgeff. Modeling agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of the 2rd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484, Cambridge, Mass., April 1991. Morgan Kaufmann.

[RN95] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[RS95] S J. Russell and D. Subramanian. Provably Bounded Optimal Agents. *Journal of Artificial Intelligence Research*, 2, 1995.

[Rum96] R. Rummer. *Kognitive Beanspruchung beim Sprechen.* Beltz, Weinheim, 1996.

[RV98] C. Ruß and G. Vierke. Agent-based configuration of virtual enterprises. In *Proceedings of the Workshop on Intelligent Agents in Information- and Process Management at the German Conference for Artificial Intelligence (KI'98)*, 1998.

[RW89] S. Russell and E. Wefald. Principles of metareasoning. In R. Brachman, editor, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann, 1989.

[RW91] S J. Russell and E. Wefald. *Do the Right Thing.* MIT Press, 1991.

[San96] T. Sandholm. *Negotiation among Self-Interested Computationally Limited Agents.* PhD thesis, University of Massachusetts at Amherst, Department of Computer Science, 1996.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, Chichester, 1986.

[Sch87] O. Schwemmer. *Handlung und Struktur.* Suhrkamp, Frankfurt, 1987.

[Sch89] A. Scharf. How to change seven rowdy people. *Industrial Management*, 31:20–22, 1989.

[See85] T. Seeley. *Honeybee Ecology - A Study on Adaptation in Social Life.* Princeton University Press, Princeton, NJ, 1985.

[Sho91] Y. Shoham. Agent-oriented programming. In *Proceedings of the 11th International Workshop on DAI*, pages 345–353, 1991.

[Sie95] Siemens AG, Zentralbereich Forschung und Entwicklung SN5 Mensch-Maschine-Kooperation. *Personal Trip Assistant: Dokumentation zur ZFE-Pressekonferenz Verkehr*, 1995.

[SLG$^+$99] M. Schillo, J. Lind, C. Gerber, P. Funk, and C. Jung. SIF - the social interaction framework system description and user's guide to a multi-agent system testbed. Research Report RR-99-02, German Research Center for Artificial Intelligence, 1999.

[SMH90] D. Steiner, D. Mahling, and H. Haugeneder. Human computer cooperative work. In *Proceedings of the 10th International Workshop on Distributed Artificial Intelligence*, ACT-AI-355-90, MCC, 1990.

[Smi80] R. G. Smith. The contract net protocol: High level communications and control in a distributed problem solver. *IEEE Transactions on Computers*, 29:1104–1113, 1980.

[Sol87] M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 1(35):254–265, 1987.

[SSR95] G. Smolka, C. Schulte, and P. Van Roy. PERDIO: Persistent and Distributed Programming in Oz. Technical report, German Research Center for Artificial Intelligence, Saarbrücken, March 1995.

[Sta79] C. Starr. Origin and evolution of insect sociality: A review of modern theory. In H. Herrmann, editor, *Social Insects*, volume 1, chapter 2, pages 35–80. Academic Press, 1979.

[Ste92] D. Steiner. MEKKA: Eine Entwicklungsumgebung zur Konstruktion kooperativer Anwendungen. In J. Müller and D. Steiner, editors, *Kooperierende Agenten*, D-92-24, pages 17–21. Saarbrücken, 1992.

[SW86] Stanfill and Waltz. Towards memory-based reasoning. *Communications of the ACM*, 29(12), 1986.

[Tam98] M. Tambe. Implementing agent teams in dynamic multi-agent environments. *Applied Artificial Intelligence*, 12, 1998.

[TJ77] B. Tuckman and M. Jenson. Stages of small-group development revisited. *Group and Organization Studies*, 2:419–427, 1977.

[Var87] H. Varian. *Intermediate Microeconomics.* W. W. Norton and Company, 3rd edition, 1987.

[VDP87] H. Van Dyke Parunak. Manufacturing experience with the contract net. In M. Huhns, editor, *Distributed Artificial Intelligence*, pages 285–310. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1987.

[Vie00] G. Vierke. *Cooperative and Competitive Resource and Task Allocation in the Haulage Domain with a Holonic Multi-Agent System.* PhD thesis, Universität des Saarlandes, Saarbrücken, 2000. To appear.

[Wer89] E. Werner. Cooperating Agents: A Unified Theory of Communication and Social Structure. In L. Gasser and M. Huhns, editors, *Distibuted Artificial Intelligence*, volume II. Pitman, 1989.

[Whe11] W. Wheeler. The ant-colony as an organism. *Morphology*, pages 307–325, 1911.

[Wil71] E. Wilson. *The Insect Societies.* The Belknap Press of Harvard University Press, Cambridge, MA, 1971.

[WJ95] M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[Wöh81] G. Wöhe. *Einführung in die Allgemeine Betriebswirtschaftslehre.* Verlag Wahlen, 14th edition, 1981.

[Woo94] M. Wooldridge. This is myworld: the logic of an agent-oriented DAI testbed. In M. Wooldridge and N.R. Jennings, editors, *Intelligent Agents: Theories, Architectures and Languages*, volume LNAI Volume 799, pages 401–456. Springer-Verlag, Heidelberg, Germany, 1994.

[WT97] W. Wahlster and W. Tack. SFB 378: Ressourcenadaptive Kognitive Prozesse. In M. Jarke, K. Pasedach, and K. Pohl, editors, *Informatik '97 - Informatik als Innovationsmotor, 27. Jahrestagung der Gesellschaft für Informatik*, pages 51–57. Springer, 1997.

[Zil93] S. Zilberstein. *Operational Rationality through Compilation of Anytime Algorithms.* PhD thesis, University of California at Berkley, 1993.

[Zou76] G. Zoutendijk. *Mathematical Programming Methods.* North-Holland, Amsterdam, 1976.

[ZR96] S. Zilberstein and S. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 1996.