Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

A Case Study in Distributed Planning for Autonomous Cooperating Agents

Peter Breuer

SEKI Report SR-91-2 (SFB)

A Case Study in Distributed Planning for Autonomous Cooperating Agents

Peter Breuer

Department of Computer Science
University of Kaiserslautern
P.O. Box 3049
D-6750 Kaiserslautern
Germany

Abstract

A distributed solution is appropriate for many planning and search problems where large search spaces occur. These search spaces often cannot be bounded with adequate global heuristics, so "traditional" strategies may become very expensive.
We show that a distributed setting, where problem knowledge and information about the domain is decomposed among individual entities, can cope with such snags in a natural way. Simple local heuristics suffice to achieve good global solution.
This thesis gives an overview about the basic methodology of Distributed Artificial Intelligence. In order to clarify the intentions a larger sample scenario will be discussed in detail, a distributed realization of the "Towers of Hanoi" puzzle. The results support the feasibility of this approach.

Viele Planungs- und Suchprobleme mit riesigen Suchräumen können verteilt angemessener gelöst werden. Oftmals ist es nicht möglich, diese Suchräume mit globalen Heuristiken zu beschränken, weshalb „traditionelle" Strategien sehr kostenintensiv werden können.
Wir zeigen, daß ein verteilter Zugang in natürlicher Weise Abhilfe schaffen kann. Hierzu wird das Problemlösungswissen und die Informationen über die Problemwelt verteilt auf einzelne Bestandteile des Szenarios. Einfache lokale Heuristiken genügen, um eine gute globale Lösung zu erreichen.
Diese Arbeit gibt zunächst einen Überblick über die grundlegenden Begriffe und Methoden der „verteilten Künstlichen Intelligenz". An der Realisation eines verteilten „Türme von Hanoi"-Spiels sollen die Ideen dann klargemacht werden. Die Ergebnisse untermauern die Eignung dieses Ansatzes.

Thanks to all who helped me with this paper, especially Hans Jürgen Ohlbach for various discussions and good proposals on the topic, and, of course, my girl friend Micha for the "mental" support she gave to me during some hard periods.

Table of Contents

1. Introduction

Distributed Artificial Intelligence (DAI) is a pretty young research area in the wide field of Artificial Intelligence. If offers many appealing viewpoints and methods to tackle old and matured problems in a new and surprisingly successful manner.

A good example is the well known AI discipline planning. The traditional setting is that a distinguished planner must find a sequence of actions which transforms a given start situation into a desired goal situation. Assume, you have to plan a complicate process, e.g. a machine is to be assembled, a puzzle is to be solved, or a schedule of charges in a transport agency is to be found. A centralized planner must know everything about the scenario a priori. The methods ("how to do something"), the desired and possible configurations ("what to do") and the order and the dependencies of the various actions ("when and why to do"). All knowledge and all experience have to be concentrated in the planner.

Alternatively consider the scenario where each involved part has some small amount of knowledge. Each (simulated) part of the machine, for instance, knows its desired position and other positions where it may be stored intermediately. The parts can send messages to each other and thus cause other parts to help them reaching their goals. Then planning is distributed and restricted, all parts of the whole process are planners for their own sake, altogether they may fulfil the common goal. This is a sketch of our proposal which will be discussed in detail.

In this diploma thesis a sample scenario for DAI planning shall be introduced and embedded in the overall terminology.

Therefore in the second chapter the basic termini of DAI are introduced. Especially the subchapter about planning shows the fundamental differences to "traditional" AI planning. Other important aspects are cooperation and communication.

The third chapter provides a foundation for the tools to solve DAI planning problems. We lay our emphasis on "Heuristics" and "Eco-Problem Solving", which both have much influence on the practical implementation.

Due to the limited space, the relevant issues can only be sketched in these two chapters. However, references to the literature are given to encourage further studies.

The main part of this work comprises chapter 4. It introduces the practical implementations to show the capabilities of distributed planning. I implemented extended versions of the well known puzzle of the "Towers of Hanoi", a children game which appears at first glance quite simple. But it is not as it seems; for a general configuration no optimal global algorithm does exist. When the disks, however, are all individual agents which try to reach their individual goals, the scenario comes to a pretty good solution.

Two versions were realized, one with sequential message passing, and one allowing parallel exchange of information and contemporary actions. In chapter 4 first the common aspects and then each version is described in detail. Especially the ideas for the parallel version are discussed thoroughly.
In the end the realized scenarios are embedded in the overall DAI terminology and a short prospectus to future work is given.

This thesis proves the feasibility of the distributed planning approach in general. Even if most decisions depend on heuristics, the tests provide promising results. Furthermore the potential power of parallel acting can be foreseen.
The substance of this work is the investigation, which kind of local information must be exchanged and evaluated between distributed entities to guarantee a coherent global behaviour of the system. There are some general principles for a large class of "distributed problems", independent from special scenarios.

2. Distributed Artificial Intelligence

About 10 or 15 years ago, a new branch of research in Artificial Intelligence (AI) came up, Distributed Artificial Intelligence (DAI). Meanwhile the field has matured enough to reach a certain consensus about its main characteristics and principles [Ca88].
Some authors [Ch81, De87] still distinguish the term Distributed Artificial Intelligence from "distributed problem solving", which is more special in their opinion. We do not make this distinction, but define DAI according to Huhns and Castillo Hern [Hu87, Ca88]:

> DAI is concerned with the cooperative solution of problems by a decentralized group of agents. These agents are loosely coupled, but at least logically independent from one another. Agents display sophistication in the AI sense with a capability for reasoning, planning, and communicating.

This definition reveals all of the central aspects of DAI. We will soon investigate them exhaustively. Different authors put different emphasis on the topics of DAI. Decker [De87] suggests to view the field along the following dimensions:
- the level of decomposition
- the distribution of expertise
- the methods for achieving distributed control
- the process of communication

Castillo Hern [Ca88] suggests as the "main issues":
- Global coherence
- Knowledge representation
- Communication

In the sequel, these items will appear many times. The next subchapters describe them in the context of the basic mechanisms of Distributed Artificial Intelligence. First the term *agent* with respect to the community of agents (i.e. the whole system) are explained and the essentials *cooperation*, *communication*, and *synchronization* are discussed. Next the traditional approaches of *planning* are compared with the new fields of Multi-Agent-Planning and distributed planning. The planning aspect, though naturally inherent in DAI, shall gain a certain emphasis throughout this work. Principles of architectures close this introduction into the methodology of DAI.
But we start with two motivational subchapters: First some good reasons for considering DAI are given. Then several real world problems are introduced which cry for an intelligent distributed solution, and, furthermore, a collection of DAI systems which are currently at the threshold from research to application, is presented.

2.1. Why Distribution ?

The distribution of problem solving as the general paradigm of DAI provides appealing prospects for the solution of various hard nuts. Starting with the hypothesis that "All real problems are distributed" (quoted by [De87]) and the observation how communities of humans solve large tasks, we get a feeling for the demand of research in DAI.
Reasons for work in and study of DAI: [1]

- DAI provides insights and understanding about information processing phenomena occurring in the real world. Especially interactions among human societies are heavily investigated. But also models for the behaviour of a single individual may be found, models for the brain or models for performing unconscious movements, like walking.
- Distribution is a natural approach for large, evolutionary systems. These systems should be "open" for extensions and adoptions. Modularity facilitates the handling of such systems.
- The cooperation among multiple expert systems with different (but possibly overlapping) expertise can provide a solution for problems whose domains are not contained in one expert system.
- Distribution is a useful means to control complexity. "Large" problems can be decomposed and broken down into multiple cooperating subsystems to become feasible.
- DAI is the most appropriate solution when the problem itself is inherently distributed (e.g. distributed sensor nets).
- Due to their parallelism, distributed systems are potentially more efficient. They better exploit resources and solve problems faster.
- Distributed systems normally have a certain amount of redundancy. That is, some agents can solve the same tasks as others or pieces of information are known by several agents. Hence the systems become more robust against exterior influences or breakdowns of own agents. Generally, the prospectus for a graceful degradation ("soft fail") increases and the system can guarantee a higher reliability.
- The distribution of control avoids bottlenecks, which may occur when a central instance must take care for the needs of a multitude of agents.
- A distributed system with intelligent agents has a higher flexibility concerning changing problem classes. The wider the spectrum of all the agents' expertise is, the more different the analyzed problems can be. A further improvement are agents capable of learning.
- A small independent expert system could be a part of many large distributed systems. This reusability facilitates the exploitation of already matured tools for new systems.

[1] The following arguments are collected from the introducing papers [Ch81], [De87], [Hu87] (Foreword), [Ca88], and [Mi86].

Besides saving time for new research this could put forward a desired standardization of certain tools.

- Research in DAI yields a mutual stimulation of research in "traditional AI". Important fields are: knowledge representation, formal specification in general, reasoning about knowledge and belief, and planning.
- Last, not least, DAI is a challenging area for interdisciplinary approaches. For the first time, theories from biology, sociology, psychology, philosophy, linguistics, and other research areas, may provide a direct and fruitful impetus to computer science.

But we are still standing at the beginning of this evolution. Before exploiting the benefits just discussed, the foundations must be settled. Also distribution of problem solving is not gratis, the price to pay is a much more complicated organization and the costs for taming individualism. But the prospects promise it is worth the efforts.

2.2. Real World Applications

In order to get an impression about the kind of problems people think DAI can solve, we present some scenarios which have been mentioned in the literature. Then some real existing systems are briefly described to give a feeling for the capabilities of current DAI research. Maybe some of the systems will become commercial soon.[2]

DAI in working cells

In [ME90] an interesting scenario is outlined (see Figures 2.1. and 2.2). In a car factory a working cell for painting cars looks as follows: The whole working space consists of four limited areas. The areas are connected by corridors. Three painting robots shall paint 80 cars; 50 cars shall get white colour, 18 red, and 12 blue. The order and the place for painting them is not limited, however, there must always be enough space to move. After the painting is finished, all cars must be arranged such that as long as there are some, a car with a certain colour can be accessed.

This is an excellent example for a DAI problem. The goal state is specified and the task is to plan the motions of the cars and robots. Planning must obey rules of coordination. Information about others' intentions must be gathered by communication. Behind all activity an amount of uncertainty remains. Not each robot can always know what every other robot did, does, and will do in future.

Analog examples would be any other working cells in factories, or bureau organizations working towards a common goal.

---

[2]It is a pity that most of the work in DAI seems to be very interesting for military use (and thus it is heavily sponsored). I hope there will never be an opportunity to test such (still hypothetical) systems in "real world applications".

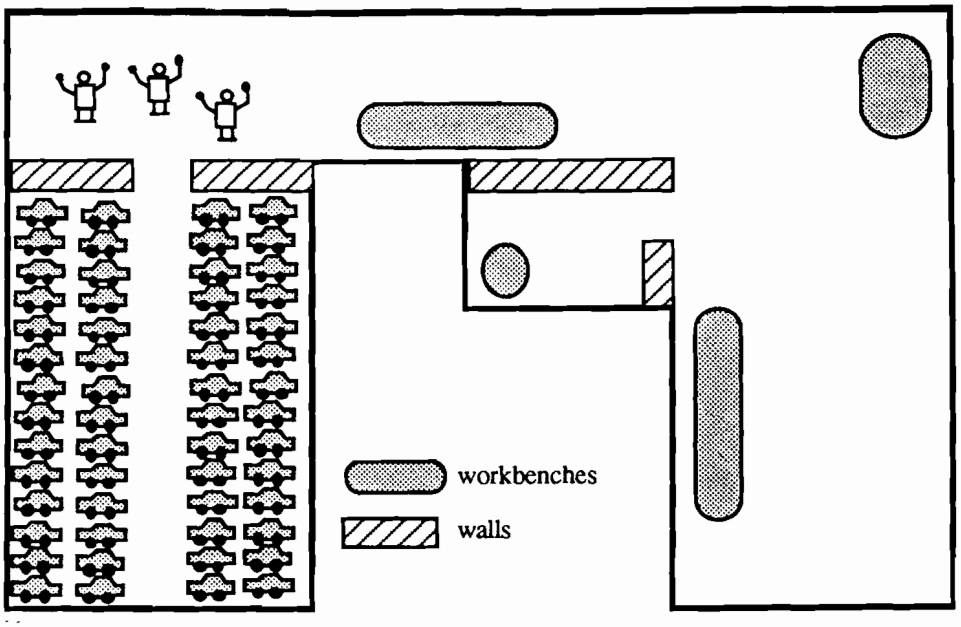

Figure 2.1. : A car painting scenario (according to [ME90])

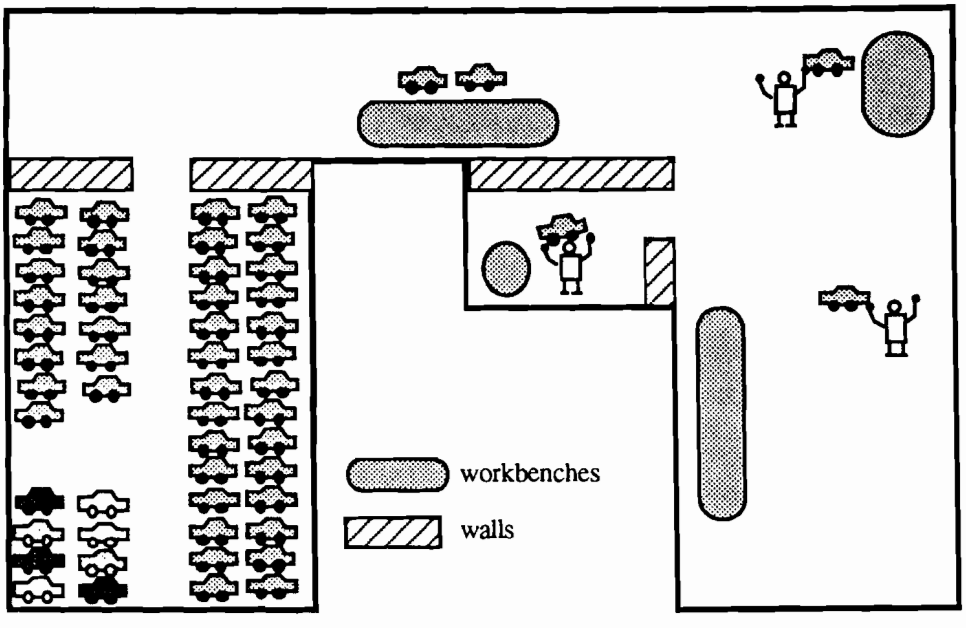

Figure 2.2. : Painting robots at their work

DAI for managing resource conflicts

Common problems of computer science (especially for operating systems, cf.[Ne86]) are those where resource conflicts occur. There are more requests for a resource than can be granted in one moment. For instance, [Ca88] introduces the well-known setting of the "Dining Philosophers" from a DAI perspective.

The distributed approach with its inherent communication metaphor may provide means for elegant solutions. Communication facilitates negotiation, which could solve resource conflicts.

New approaches

Up to now the problems show configurations where it was a priori clear which part is active (the actors which plan and must be coordinated) and which part are passive (the resources which constrain the planning). A new and attractive approach is to make all parts of the scenario potentially active, in particular the resources. These participate in planning and negotiations, arguing from their very own positions.

A planning system organized in this way can generate a possible solution without the help of a supervisor who knows everything. The local knowledge of the "agents" suffices. Especially for problems where large global search spaces but no suitable global heuristics exist, this approach seems to be appropriate. The main phenomenon is that large global search spaces are broken down into relative small individual search spaces of each agent. Each agent can find its solution very quickly, the overall solution of the system must then be attained by negotiation and conflict resolution strategies.

Consider the car painting scenario above. In order to achieve a real distributed scenario we regard not only the robots as autonomous agents (this remains an almost centralistic view), but also the cars, the colours, the workbenches, and so on. A goal of a car looks like "I want to be painted red"; colours and workbenches wish to be used as often as possible. Note that all scenarios are *simulations* to find an optimal plan, no one wants to teach real cars or colours to send messages.

A further example for such scenarios could be a *shunting-station*, where locomotives, wagons, parts of the rail and the track switches are intelligent, active parts of the scenario. Now suppose that the wagons only get their individual goal description and have to take care for themselves how to reach it. A wagon would send a message to any of the locomotives: "Come here, and pull me to position XY". But surely other wagons would also like to "possess" the locomotive, so a negotiation must take place. Also ordering constraints are important.

The same way many other applications can be modelled, for instance a loading yard of a transport agency, or assembly processes. The central issues are to provide the agents with some local knowledge concerning themselves, their nearer environment and their exact individual goals. Then they will pursue these goals alone.

This will be exactly our approach, introduced in a larger example in chapter 4.

Now to some already existing DAI systems.

Hearsay II (revised in [De87])

Developed in the mid 1970's as a *speech understanding system* at Carnegie Mellon University, it became a "classical" foundation for many later DAI systems. Though firstly not distributed, it is obvious that it can be. The ideas of several knowledge sources, a blackboard, a priority based scheduler and a focussing mechanism for meta-level control

were so well accepted, that "Hearsay II" became a metaphor more for the architecture than for the system itself.

Air Traffic Control Systems ([CaMcSt83])

Several researchers at the Rand Corporation, California, USA, are engaged in studies about air traffic control. Their domain is a simulated airspace with planes arriving at fixed entry and exit points. Several different organizational policies are taken into account to handle, for instance, the scheduling of planes and the satisfaction of fuel demands.

Vehicle Monitoring

The Distributed Vehicle Monitoring Testbed (DVMT) (revised by [Ca88, De87]; see also chapter 2.8.) is a simulated environment with moving vehicles making sound that can be picked up by electronic sensors. The goal is to create a dynamic map of the area monitored by the sensors to enable traffic supervision and control. The DVMT is based on the Hearsay II architecture (see above).

YAMS ([Pa87])

Parunak introduces YAMS ("yet another manufacturing system") as a factory control system for discrete manufacturing (opposed to continuous process control). Interesting aspects are the separation between control medium (YAMS) and performance medium (the working cells), the real time constraints for the performance medium and its activeness (compared to passive sensors in "normal" sensor nets).

MINDS ([HuMuStBo87])

Another interesting application is distributed document retrieval to facilitate the expensive but unavoidable search for documents in bureaux. Huhns et al. developed the system MINDS as "a distributed collection of knowledge based systems for efficiently managing and retrieving documents in an office environment of networked workstations".

Especially knowledge and tasks are shared among the workstations. Document retrieval can be customized for each user by learning document distribution patterns as well as user interests during operation. Heuristics are employed for these learning procedures and also for self-initializing the system.

Further application examples

- Context free parsing of formal languages. In [Sr87], parallel versions of the recognition algorithms of Cocke-Kasami-Younger and Earley are introduced. One result was that the implementations revealed more parallelism than was apparent at first glance.
- Real time dialogue systems as examples for participant systems [Ch87]. The simple system *Cantata* allows the exchange of messages between many users in real time. The emphasis lies on communication and synchronization, no cooperative work is regarded directly.
- The management of hospitals [Mc84].

- Knowledge based vision tasks. In [LaChRoMc89] a method for automatic detection and classification of objects and shadows contained in image data is presented. These image data are provided by an active sector scanning sonar system, which itself then is a part of a much larger system for distributed problem solving tasks. The final goal of the overall project is to build an unmanned, free swimming submersible vehicle for offshore energy exploration and production activities.

All these applications show possible utilizations of DAI methods. There will be even more chances, if problems are tackled from new perspectives, far away from old and matured methods. Maybe some nuts can be handled more natural in the light of distribution, remember: All real problems are distributed!

2.3. Agents and the System : The Individual in the Mass

In the DAI terminology, an *agent* has nothing to do with agents from the KGB or 007 James Bond. An agent simply is the smallest entity in a system that causes effects.[3]
Agents are autonomous, when they have their own goals, capabilities and knowledge [KrMa90]. This means that such agents act with their own responsibility. Agents are also known as actors, node processors, knowledge sources, etc.
Important attributes of autonomous agents are therefore a certain degree of freedom and the consciousness for their actions. They are only loosely coupled with others, thus forming a network of relations. But at least they must be logically distributed, i.e. independent. Furthermore, if they are also physically separated from one another, we speak of decentralized agents.
An important characterization of multi agent systems is the *granularity*. This describes the ratio between the capabilities of a single agent with respect to the capabilities of the whole system.
Fine grained systems are constituted from a mass of dumb agents which can perform only simple, uniform tasks. Examples are models of connectionism (cf. [Sh87]) or distributed sensor nets. The intelligence becomes apparent only in the behaviour of the whole system.
At the other side of the spectrum, there are coarse grained systems. Here an agent has many capabilities, can perform various tasks and covers a broad spectrum of abilities. Each agent alone can already be regarded as intelligent. The whole systems then is a

[3] Agents need not have a sex. They may be arbitrary entities in a scenario, e.g. expert-systems, robots, humans, sensors, or blocks in a Blocks World. So they are referred in the following as an "it", which may be unfamiliar at the beginning.

collection of "experts". So the typical example for agents in a coarse grained system are - not surprisingly - expert systems. But also flexible robots are imaginable.
Both extremes have their pros and cons. Fine grained systems may be easier (and cheaper) to build and to maintain, but the task of decomposition for the simple processing units will probably be very complicated. Also the class of problems which can be handled at all, is restricted. Coarse grained systems are expensive. When single agents are complex, relations between them will be even more. Each agent must also have its well-defined and widely disjoint competence areas, otherwise long discussions and negotiations may occur.
Castillo Hern [Ca88] suggests a framework to describe and compare different approaches of DAI systems. He distinguishes between the "System Conceptual Model" to characterize the overall system and the "Agent Conceptual Model" for the individual entities. The following dimensions are proposed:

System Conceptual Model	Agent Conceptual Model
- Structure	- Structure
- Knowledge Organization	- Knowledge Organization
- Coherent Cooperation	- Action
- Communication	- Perception of the Environment
- System Reliability	
- Result Formation	

Some of these points shall be discussed in more detail.

2.4. Cooperation

The paradigm of distributed problem solving is applicable to various problems. But none of the problems discussed above can be solved only by decomposing the problem, allocating subtasks to the distributed problem solvers and at last assembling the partial results. Problems like these (and many other interesting problems for DAI) have some common characteristics: Simple decomposing and isolated solving on distributed nodes is not sufficient. To solve the tasks, a node needs more information about its environment; "which are my neighbours ?", "what information do they have ?"
Consider the Distributed Vehicle Monitoring Testbed (DVMT [De87, DuLeCo87], see also chapter 2.2). A single sensor has to monitor its well defined area. Well, one might say, that is fine. Each sensor has its own subregion, all sensors may assemble the results and, finally, the whole region is known. But this is impractical. A sensor alone cannot

produce sound results. Its measures may be inexact or disturbed. The sensor may be interrupted for some time or break down completely.

Now we could try to use algorithms from image processing to improve the quality of the sensor data, however, this is not appropriate. A vehicle may have an arbitrary trajectory. There are no straight lines, which a computer vision algorithm would detect. So the correct solution must be found in the moment when the image data is created. An outcome of this problem is to let a sensor "get in contact" to its neighbours and thus try to get more information about its own measures. If one or more other sensors confirm the opinion of the first, the probability of the data increases considerably.

In human organizations we call this phenomenon *cooperation.* We regard cooperation as an interaction of at least two co-actors resulting in a benefit for at least one of them. Without this interaction the benefit could not have been achieved.

Cooperation is a natural way to tackle large problems, e.g. the interacting of a human group of experts [DaSm83]. Because of the kind of problems - as in the example - in DAI, it is one of the very principles to find a solution at all. Cammarata et al. [CaMcSt83] see the expertise of a DAI system in its cooperative strategies.

Without cooperation each node would stick on its limited view, see only its limited problems and thus produce limited solutions. An overall result as a synergy effect of the work of all nodes would be almost impossible.

Werner calls goals of agents in distributed settings as introduced in chapter 2.2. *social goals* [We88, We89]. He argues that social goals are not achievable by one agent alone, but only by a group of cooperating agents. These goals cannot be decomposed into separate subgoals that are achievable independently of the other agents activities. One agent cannot simply proceed to perform its action without considering what other agents are doing. So, in this terminology, the monitoring of vehicles with distributed sensors is a social goal.

Because of its importance for DAI we must take a closer look to the principles of cooperation. We said, cooperation is necessary to achieve social goals. But often also the quality of the solution is important, too. A distributed system, which spends such a long time with communication that nobody is interested in the solution anymore, is worthless. Two new aspects become evident. The demand for coherence [DaSm83, Ca88] and the impact of time [KrWo89].

A successful cooperation strategy must ensure a steady convergence towards a solution. The system must not move around and come to no end, a coherent behaviour should be achieved. Since each action takes time, methods are to prefer which involve actions as short as possible. Consider distributed sensor nets which monitor dangerous processes in a factory. Surely time for cooperation is limited in an emergency case. For the sake of simplicity, the aspects of tense are not treated in the further work. In [KrWo89] some

relationships between temporal interval relations and multi agent scenarios are sketched. The principles of temporal relations were presented by Allen [Al84].

Based on the previous work in this area (e.g. [We88, We89, DuLeCo87, CoMiCa90]) we distinguish the following spectrum of *social intentions*:

Co-existence. The agents do not interact, but solve their problem independently. So no cooperation is necessary. Since we consider "social goals", this case is not relevant for DAI.

Malevolence. Agents do not cooperate, they even work actively against each other (preventing them from reaching their goals). This is a typical competition situation, e.g. when a resource is heavily demanded and neither a priority order for the agents is given nor a fair mediator resolves the conflicts. We will not regard such anarchic scenarios in the further work because in total commensurability only stochastic observations are possible.

Self-interest. Cooperation is strived for only if agents regard a cooperative action as beneficial for their own interest. All agents which engage in cooperative acting expect advantages for themselves. Consider an agent which wants another (an obstacle) to go away. The latter will only follow the demand, if it can do something for itself at the same time (e.g. coming nearer to its goal).

Benevolence. Agents also engage in cooperation when no direct use for them is in reach. They would, for instance, even conform to wishes of others, when they do not come nearer towards their own goal nor gain any other advantage. But a delineation to altruism must be made. Agents will only remain cooperative as long as they do not suffer any negative influence.

Altruism. Agents act always cooperatively, despite of the consequences for their own interests. An example is a (human!) parent-child relationship, where the parent does *everything* for his/her child. These relationships hardly occur in the DAI literature. Because we are interested only in those agents which have a certain will to attain their goals, we do not regard altruism any further.

Thus especially the spectrum self-interest - benevolence is relevant for cooperation strategies in DAI. This should be regarded as a continuum, no strict separations can be made. In groups of humans the behaviour may change, e.g. dependent from different tasks or moods. Though we do not intend to provide artificial agents with moods, their behaviour also may not keep static for an observer. We must keep in mind that the last categorization was a descriptive one. Self-interest, benevolence, etc. can only be noticed with respect to the agent's actual intentions, it cannot be an absolute characteristic.

Types of cooperation are adjacent to social interactions as described above. These describe the kind of relation between the (usually two) parties involved in a direct cooperation task. Again we introduce a range of classifications [We88, We89, CoMiCa90]:

Accidental cooperation. The parties act independently and unwittingly in favour of at least one of them. Example: One agent moves to a certain position and provides thus a desired support for another which wants to act. The latter one has not explicitly demanded this support.

Master-slave relationship. An agent *forces* a second one to do something for it (the first). The "master" alone has control of the proceeding.

One-way cooperation. An agent *asks* another agent for cooperation (also known as "bidding"). The other may agree or may not. The task is necessary only for the first agent, but the second keeps its autonomy when serving the request.

Mutual cooperation. The task is beneficial for both parties. By exchange of information they resume their proceeding (for negotiation cf. chapter 3.5.).

Cooperative goal adoption (proposed by [CoMiCa90]). Not only information about the task which shall satisfy the goals of the parties is exchanged. Moreover, information about the goals is communicated and the goals are mutually adopted. A task which formerly would not have met both intentions, eventually satisfies the (new) goals.

Example (from a Blocks World): Agent A's goal is "go to position X,Y". Agent B's goal is "Build a blue tower at position X' Y'". If the goal can be slightly modified to A: "go to position X',Y'" and B: "Build a tower at position X' Y'", a move from A to X' Y' would be the basis for the mutual satisfaction of the new goals. But the problem is to let the agents find a common basis for the goal adoption. This is not trivial.

What we have seen is that some amount of knowledge of one agent must necessarily be known by the other to achieve a successful cooperation. How can this exchange of information be achieved ? The key issue is *communication*. Like Werner states [We88] "No cooperation without communication!" (though it is not overall accepted), we will follow this paradigm.

2.5. Communication

In the previous chapter the importance of shared information to cooperate effectively was outlined. We will now argue that communication is an appropriate means to achieve that. Communication shall be regarded as the exchange of information between at least two agents. Both the partners are aware of the situation, at least one of them wants to gain benefits.

Werner introduces the spectrum of approaches made to communication [We88, We89]:
No communication. Agents rationally infer the other agents' intentions. One example of this approach is the use of decision matrices as proposed in [Gi87, RoBr89] and used in game theory: The agents always have as much information as possible about the next move in a global decision matrix. So they can compute what is best for them with respect to the next move of the other agent. Communication is unnecessary.

Advantages of this approach are:

- Avoidance of communication costs. No communication channels need to be established and to be maintained. No synchronization procedures are necessary.
- Computation may be faster than communication, if otherwise many intermediate results must be exchanged [DaSm83].
- Errors due to unsafe communication are avoided.

But on the other hand a lot of drawbacks appear:

- If each agent computes all results on its own, the whole society works very redundantly. The communication of complete results might be faster in the case when computations are complex.
- In many situations the agents must have information about the other agents' beliefs. Without verifying these beliefs by asking about their status, but only relying on own information, infinite nestings of belief may evolve. Agent A forms its beliefs based on the beliefs of agent B. This in turn relies on the information it has about agent A, and so forth. How should possible changes of belief reach the other agent without communication ?

Because of its costs communication should be restricted to a minimum. For simple or standardized information exchange, rational deduction may be appropriate. A robot $\mathcal{A}$, for instance, gets the information of another robot $\mathfrak{B}$ that $\mathfrak{B}$ carries a certain chest. If $\mathcal{A}$ is sure that $\mathfrak{B}$ has not moved in the meantime nor any supernatural things have happened, it needs not to communicate again about the "carry" fact, but can deduce it rationally from its own information.

But this is not the case for the exchange of high level information, e.g. the information about complex goals.

Primitive communication. Communication is restricted by a finite set of fixed information signals with fixed interpretations. The possible effects of these means are limited, for instance simple coordination tasks between sequential processes can be achieved. Complex commands cannot be represented, let alone arbitrarily syntactic constructs. The number of representable information entities is fixed, so sophisticated cooperative action is virtually impossible.

For fine-grained DAI systems this may be a proper way to allow communication between the nodes. Where only simple information must be exchanged, flexible channels and

sophisticated nodes in coarse grained systems are not exploited adequately. Complex nodes like experts systems should not be limited to a finite set of possible messages. The format may be restricted, but the content must be free to permit the nodes the exchange of flexible and worthwhile chunks of information.

Plan and information passing. The agents mutually exchange their final plans. Whichever plan arrives first is accepted. Despite of the fact that such a proceeding is extremely expensive (costs of communication), there are still more disadvantages: No guarantee can be given that the "winning" plan is optimal (or at least acceptable) for both parties. A further problem is (as we will discuss further in chapter 2.7.) that total plans normally cannot be formulated in advance, especially not in real world applications.

High-level communication. This is a field at the frontier to linguistics; for instance, much has been done in speech act planning. To exploit the work of linguists and cognitive scientists a more formal approach is necessary. An artificial agent cannot understand arbitrary syntactic information, it needs certain *rules* to interpret language, as well as a human does. This is not apparent when we use our native language, but the less experience we have in a foreign language, the more we consciously stick to the rules we have learned. An artificial agent has no knowledge a priori. It must completely rely on rules. Therefore a formalized approach to communication is necessary. The advantages of high-level communication are evident:

- Flexibility of expressions, almost all communication situations are covered.
- It is better to understand by humans (humans may even be directly involved in the communication process).
- Social, cognitive, and linguistic theories and results may become applicable.

High-level communication, which is strictly formalized, shall be called conversation [KrMa90, WoKr89, KrWo88]. The processes which are incorporated, the conversational processes, shall now be investigated further. As we mentioned above, these processes have to be formalized that agents are able to interpret, say, a bitstream of '0' and '1' as a certain message. Protocols [KrMa90] are an appropriate means to this end. We define a protocol rather informally. It determines the message types, the states, and perhaps also the roles of the parties involved. *Message types* outline the allowed formats of the messages, the arguments, etc. *States* partition the whole conversation process into discrete steps. In each state alternative branches to proceed are normally given. *Roles* install social structures [We89] which may further restrict the conversation.

Example: A protocol for robbery.

Message type:	give_money
Format:	"give <requestor: agent> <addressee: agent> <budget: integer>0> $"
States:	Before: requestor has not enough money, addressee has money After: requestor has more money, addressee has less money
Roles:	requestor: master; addressee: slave

Some researchers propose a mediator as a special agent only for conversational processes [KrMa90, WoKr89, KrWo88]. The mediator monitors the conversations and maintains the communication channels. The other agents can save their resources for private tasks, they do not have to cope with conversational problems. A mediator furthermore is suitable to break up the complicate multi-lateral conversation into a set of bilateral conversations.
In [KrWo88], mediators are defined as "fully automated pseudo-users, which supply precisely defined services within conversations of certain types".

How will communication then in fact be performed ? Actually, there are two principles for the realization: Information can be exchanged via regions of shared memory, or through message passing [KrMa90].
A typical example for communication by means of shared memory is the *blackboard mechanism*. This was already used in Hearsay II ([De87, DuLeCo87]; cf. also chapter 2.2.). A blackboard can be conceived as the name suggests, it is a common "structure", where all authorized agents can concurrently write upon or read from [NiAiRi89]. In principle, an information on the blackboard is accessible by all agents (we say, the information is *broadcasted*). This can be restricted by delineating the groups of agents which are explicitly allowed to read (analogous for write). The information on the blackboard normally resides in several levels of abstraction. The advantage is that the agent can decide, how fine (or coarse, respectively) the information must be for it.
The realization of a blackboard is quite simple, just some areas of common access must be provided. These may be common chunks of storage, or (more abstractly, but at last the same) data structures which all involved parties may access.
Message passing [KrMa90] is the other paradigm. The exact realization can vary considerably. The spectrum ranges from collecting arriving messages in mailboxes and processing them when enough time is left, to a direct return of an answer (e.g. as a remote procedure call, see chapter 4.4.1.). Unlike in a blackboard setting where broadcasting is essential, the general principle is the individual addressing of a message. A message normally has exact one recipient (selective communication). It enhances the knowledge of the receiver or causes it something to do. Broadcasting can only be achieved by dull copying the message and sending it to all.

The information communicated has, as mentioned above, influence on the knowledge state of the receiver. But it can be further categorized by three aspects [DuLeCo87]:

- Relevance. Describes the amount of information in the message that is consistent with the solution.

- Timeliness. Measures the extent to which a transmitted message will influence the current activity of the receiver.
- Completeness. Describes the fraction of the complete solution which is represented by the message.

At last, a few words about levels of communication. It may be convenient to exchange information not only about the world states, or the intentions and abilities of the agents, but also information about this information.
How is the information organized, where does it come from, where is it stored ? Also facts about the processes of cooperation and communication can be of interest: What is the actual role of an agent, which states does it aspire next ?
Consider an agent who knows not only the intentions of another, but also how the latter came to these intentions. The first one knows considerably more about the other and may possibly *deduce* the next reactions and thus save the costs for communication.
This form of communication is known as meta-level communication [DuLeCo87]. It concerns primarily planning aspects, where agents have to reflect about future intentions of their co-actors in the scenario (see chapter 2.7.).

2.6. Synchronization

In systems with distributed activity, especially if parallel processes are involved, certain uncomfortable problems may arise: Processes overtake or cross each other, causing inconsistencies or deadlocks [KrMa90]. It is even more difficult to monitor the effects of various processes if the scenario has no global clock and thus no global states. Thus synchronization is crucial for DAI scenarios.
Synchronization shall be regarded as the policy to prevent inconsistencies and deadlocks by detecting mutual dependencies, possible conflicts, and hazardous behaviour of involved parts of the system.
For the multi-agent scenarios, this implies that the acting of the respective agents must be coordinated[4] such that the effects mentioned above cannot occur.
There are three basic ways to realize synchronization:
- The agents alone are responsible for a coordinated behaviour. They must come to an agreement (mostly by negotiation) and furthermore control the execution of their actions. Hence the typology of messages must be augmented by *synchronization messages*.

[4]Though not fully identical, we will assume that synchronization and coordination have the same intention and do not distinguish between them.

- Another agent, which is not a "normal" acting agent, ensures coordinated processing [KrMa90]. Such a mediation through a coordination agent must not be dictated to the other agents. They may bring in their proposals and the mediator may either "have the final word" or go on in mediation until a compromise is reached.
  We prefer this issue, because a coordination agent can release itself from any local agent's view and come to a more comprehensive anschauung of the world. Furthermore expensive communication can be reduced to information of the coordination agent.
  A drawback is the problem of a possible bottleneck [DaSm83], which the coordination agent as a unique node may produce.
- A total order of the agents with respect to attached priority values is possible in advance. In each conflict situation synchronization can automatically be managed by using the priority order. An improvement of this inflexible solution would be an effectively computable priority function depending on the situation the agents are involved in at the moment.

Abstractly spoken, a coordination agent has always to detect *negative relationships* between the planned actions of different agents [KrMa90]. These relations disturb or prevent the actions of at least one of the agents. Another word for negative relationships is *conflicts*.
Two basic types of conflicts may occur:
- Conflicts through incompatible wishes.
  Example: An agent A wants to go to a certain place X. This is not possible because agent B is located at X and cannot go away.
- Conflicts through overlapping wishes (including resource conflicts).
  Example: Both agents A and B want to go to place X. Which of them will win ?

We have introduced the *principles* of cooperation, communication, and synchronization. All these aspects mentioned in the last three chapters shall be exemplified in chapter 4, where a sample scenario is introduced.

2.7. Planning

Planning is an essential human problem solving method. Whenever a problem is worth the effort, i.e. it is complex and we are interested how to come to the solution, we develop a plan. A plan describes a way from the actual state to a future situation, where the desired solution is attained. Developing a plan in "normal life" often occurs informal and by no means spectacular. But because of its use for humans it has been heavily

investigated in AI. Since the beginning of AI research, the automated generation of plans is a "classic" goal. Thus it is also desirable to formalize the intuitive understanding of that, what planning is or may be.
One of the first approaches was the General Problem Solver (GPS) of Newell, Shaw and Simon which was presented in the early 1960's. This was an attempt to model universal human problem solving, but due to its generality [5] the solution of complex planning tasks was infeasible [Ni80, Ri83, He86].
The tendency from general approaches to domain dependent systems after these experience is an evidence for the complexity of real world applications. Today it is well accepted that planners must comprise a certain amount of domain specific knowledge to cope with interesting problems [He86]. The DAI metaphor provides new ways for planning: Multi-agent planning and distributed planning. These proposals try to avoid some of the shortcomings of "traditional approaches".

2.7.1. Traditional Planning

In the traditional approach, one planner creates the whole plan for the entire environment to be planned. We assume the following: The planner has collected all the knowledge he/she/it needs to be able to take all possibilities into account and to ensure that the plan is appropriate. Furthermore the environment is good-natured, i.e. only the effects caused explicitly by the plan will occur. In particular, there are no interferences which could endanger the plan's execution.
Planning itself is regarded as finding a sequence of actions that transform a given start state into a desired goal state. *States* are (partial) descriptions of the world, *actions* perform changes in this world. Actions are usually represented with three lists: PRECOND, the list of all conditions in the world which must be true to enable the execution; ADD, the list of all what becomes true after the execution; DELETE, list of all facts which are no longer true and thus must be deleted. In the rest of this paragraph we will briefly sketch the main aspects of traditional planning (the material is mainly taken from [He86, He89, He90, St87, Ri83], where the traditional approaches are exhaustively described).
A plan is formed for a *single actor*, e.g. a robot. Thus the sequence of actions must be linear. The technique of *non-linear planning* tries to delay this linearization as long as possible to avoid unnecessary restrictions. *One-level planning* resides at the level of elementary operations. For instance, in the Blocks World a plan is built up with actions like: ...STACK(B,A), PICKUP (C), STACK(C,B), PICKUP(D),...etc. In large

[5] And, of course, because of the lack of sufficient computer power in these days.

planning problems this view is too limited. It will be more convenient to plan first "larger steps" and then refine the plan more and more by reducing the stepwidth (replacing of "abstract actions" by more "concrete" ones). Finally the level of elementary operations is reached. This stepwise decrease of abstraction is known as *hierarchical planning*.
Another approach is *interval-based planning*, where aspects of time are modelled more naturally. *Meta-planning* comprises "normal" actions, which change the world, as well as plan-changing actions, which are used to plan the planning process. This abstraction allows more flexibility in plan generating.
After all, two issues hold for traditional planning:

- There is a strict separation between plan generation and plan execution. Before executed, the plan is created explicitly and handed out to the executor.
- Strictly spoken, planning is nothing but search. How else should the planner find a sequence of actions ? Therefore it is interesting to consider some basic aspects of search separately (cf. chapter 3.1.).

STRIPS [Ni80] is the prototype of all traditional planners. It plans sequences of actions for manipulating, for instance, objects in a Blocks World. It has the current state of the world in its database and a goal description on a goal stack. Using search methods, the entries of the goal stack are successively removed by the preconditions of appropriate actions, until all entries of the goal stack match entries in the database. This implies, the sequence of actions (the plan) has achieved the transformation of the start state to the goal state.
Several problems arise when real world scenarios are to be planned.
The *frame problem*, for instance, asks for the constant frame which stands "behind" the actual effects of an action. On transforming a state description into a description of the successor state, one has to specify not only the effects which are caused by an action (ADD-, DELETE-lists), but also the facts which are not altered. STRIPS uses a simplifying assumption (known as "STRIPS assumption"), which generally expects that actions modify only the things explicitly mentioned in their ADD- and DELETE-lists. In complex applications where each modification may cause arbitrary side effects, this assumption cannot be uphold any longer.
Furthermore, there is the problem of subgoal interaction. If a certain subgoal if reached, a solution for another subproblem may undo the first solution. This is not a trivial problem because of the requirement for linearization. STRIPS uses a simple model to overcome it. After reaching the final goal, it tries to attain possible undone subgoals anew. This is a rather coarse and unfounded way and may result in endless loops.
The strict separation between planning and execution must be weakened. Real world problems cannot be planned fully in advance. An usual human way to plan is : Create

some steps, look what you have caused (acquisition of new information!) and then go on with planning. This interweaving of planning and executing facilitates the reaction upon hazards occurring during the execution. They may be caused by environmental factors, by faults in planning, or by inexact execution.
The frame problem is inherent in the specification of actions. So it will not vanish in new approaches, except a specification of all consequences of an action is attempted. But no appropriate formalism has yet been found. The last mentioned problems, however, are tackled in the approaches of distributed planning, whereas multi-agent planning merely avoids problems of linearization.

2.7.2. Multi-Agent Planning

Multi-agent planning tries to avoid the demand of a strict linearization of non-linear plans. If there are many agents in the scene which may carry out the planned actions, there is no need for a total order of actions anymore. However, the principles of traditional planning are mostly preserved. There is still only one planner, which must have all the knowledge that is necessary. Non-linear, complete plans are created and distributed adequately among the agents.
The big advantage is the potential increase in performance through parallel execution. Also the use of resources can be optimized (agents receive just the tasks where they are specialists for). Linearization needs only be performed, when there are more parallel subtasks than available agents at one instant of time. However, the problem of an adequate task decomposition gains importance. If it is too complicated, the advantages of multi-agent planning may be compensated.
There are only occasional approaches to multi-agent planning in the literature. Katz and Rosenschein propose the following - roughly sketched - proceeding [KaRo89]: First of all, they represent "traditional" non-linear plans graphically in special directed acyclic graphs. The relation induced by this partial order is called a "plan-like relation". This relation must then be tested to be "valid", which means informally that the plan is executable and can be distributed to the agents.

2.7.3. Distributed Planning

In distributed planning not only the execution is partitioned over the agents, but also the planning process. Usually, the acting agents are a subset of the agents which plan. Thus the agent plan for themselves. Also the strict separation between planning and acting is weakened, normally a turn in turn change between these phases takes place.

The main advantages for distributed planning overlap with the general advantages for DAI mentioned in chapter 2.1. Especially for the planning aspect, the handling of control and data must be cited: No central instance must cope anymore with all the complex interdependencies and mutual influences of the subtasks or of the respective agents. The control is distributed; via cooperation strategies and communication facilities the agents themselves are responsible for their acting. The complex knowledge can also reside appropriately distributed among the agents. Each individual agent merely has a small portion of the big knowledge cake. This way, special features like expertise can be modelled, but also uncertainty has its natural place in the scenario: All the facts, which the agent does not know (not in its knowledge base) and also cannot guess (inference fails), must be asked from other agents.

As mentioned above, planning and execution alternate normally. The proceeding can be sketched as follows:

① Each agent creates a limited plan from its very own point of view. This may comprise one or more of the next actions to perform, actions either of its own or also of other agents.

② In order to maximize parallelism and performance, the individual plans must be coordinated. Conflicts have to be detected and to be resolved. Thus some agents probably must perform step ① several times.

③ After all conflicts are resolved, the agents act according to their plans. For the next actions they continue with step ①.

Durfree and Lesser call this proceeding *partial global planning* [DuLe89], because different parts of the scenario plan to achieve more global goals (in the end the overall goal of the planning process).

Current research [CoMePo89] tries to find out methods to determine the non-local impact of local decisions in distributed planning. This is a very important issue to avoid subgoal interactions and ensure global coherence of the plan.

The price to pay for the flexibility and the potential performance increase in distributed planning as well as in parallel computing anyway is an effective problem decomposition and the costs for communication and coordination. Hence we promote the distributed planning paradigm especially in problem domains where a distribution of control and knowledge is inherent and a centralized approach would be complicated or even infeasible (cf. examples in chapter 2.1. and 2.2.). Later, in chapter 4., we will introduce a special scenario to discuss the aspects of distributed planning in detail.

2.8. Architectures

In this chapter some principle architectures for DAI systems shall be presented and compared. These systems are general in the sense that they need not be bound to a certain application domain a priori. However, there are tendencies insofar, that a system which, say, allows only rather simple nodes cannot represent a network of expert systems. The survey comprises "classical" DAI systems like DVMT, Contract Net and MACE, and a new approach, RATMAN. We begin with an overview of common principles for all DAI architectures.

Generally, a system which models a distributed society of agents should provide the following issues:

- parallel behaviour of agents is possible
- the agents can improve their information status via communication or inference
- tasks are decomposed, distributed to agents, solved, and the results are synthesized again (this process may need many cycles)

Depending on the active influence onto its environment, a multi-agent system is characterized either as behaviour-based or as knowledge-based system. We say behaviour-based, when the agents merely react upon their environment. They adapt their behaviour to a changing world like ants in an ant-heap. On the other hand, knowledge based agents have own goals, which they actively pursue. They plan in advance, hence they can be regarded as more "intelligent" than reactive agents. A group of experts would fit in with this category.

The Distributed Vehicle Monitoring Testbed DVMT [Ca88] is used in distributed sensor net domains. The DVMT is a collection of complex problem-solving agents, where each of them resembles a Hearsay II (cf. chapter 2.2.) blackboard architecture system. The agents cooperate to interpret signals from a sensor array. Therefore they can communicate hypotheses, goals and meta-level information to other agents.
Each agent has then an individual blackboard, i.e. a common data structure where tentative hypotheses and subgoals are written upon. The blackboard is common to several knowledge sources (KS) which generate new hypotheses, and a goal processor which tries to match the hypotheses against goals, thereby building actual knowledge source instances (KSI).
The KSI's together with information about long-term and medium-term strategies of the agents are given to a planning component, which builds a plan queue in order to modify the old knowledge sources. This process is repeated many times, until all goals are satisfied.

The Contract Net [DaSm83, Ca88] is a collection of agents that cooperate through communication to perform a given task. The negotiation (see also chapter 3.5.) is guided by a protocol, it is mainly used for task distribution.
The principle is as follows: One agent can broadcast the availability of a task to other agents. This agent is then the "manager" of the task. Other agents which want to perform the tasks may send "bids" to the manager and the manager will eventually award the task to the agent with the favourite bid. The chosen agent becomes the "contractor". The manager is responsible for the task and monitors the execution, which is performed by the contractor alone.
The contract holds as long as the tasks lasts. Afterwards the two agents can engage in new negotiations. Each agent can send several task announcements and also several bids to different managers. The decision whether a contract is built up, lies both by manager and contractor. There is no force to find an agreement.
One central use of the Contract Net is to ensure the equal distribution of workload among the agents.

While the two architectures (systems) introduced so far predetermine the kind of joint problem solving, i.e. the principles of cooperation and coordination are built in, the next two systems provide a large degree of freedom to specify how a society of agents should work together in order to solve a common problem. Also the granularity of the systems can be individually selected. Such systems which have many parameters that may be adjusted to desired configurations, and provide furthermore instruments to measure the behaviour of agents, are called *testbeds*.

The Multi-Agent Computing System MACE [Ga88] is such a testbed. It is designed to support experimentation with different styles of distributed AI systems, at different levels of complexity. The dominant metaphor of MACE is a collection of intelligent, semi-autonomous agents interacting in organized ways and communicating via messages. Each agent has a model of other agents it knows in the world, called acquaintances. Agents can build organizations (groups or coalitions), they behave social in nature. An organization is a structure of expectations and commitments to behaviour; it exists only directly through the commitments and expectations of its members. The term organization refers to an abstraction which allows agents to treat a collection of activities as being part of a known concerted effort.
MACE must be instantiated to a concrete multi-agent scenario by exactly defining the agents (attributes, skills), their acquaintances, their goals and the message types allowed.

Another approach to a universal multi-agent testbed is RATMAN [BüMü90]. The new and interesting idea is to model an agent as a hierarchical knowledge base completely in terms of logic. The whole system consists of four modules: the *specification kit* is the user interface to define the agents and the strategies desired; the *agent toolbox* provides the hierarchical knowledge base as mentioned above for each agent; a special application domain is defined in the *current world scenario*, which has a blackboard as a communication platform for the agents and the user; and at last, the *status sequence and statistics box* to analyze effectively what goes on in the scenario.

A few words more to the agents in RATMAN. The hierarchy of their knowledge bases is constituted as follows: At the lowest level, the sensoric knowledge is represented (e.g. the information about an agent's hand or foot etc. (if it has some at all...). On the second level, a knowledge base in the usual sense is located. Four modules contain knowledge about space, time, common sense and special expertise, respectively. The next level defines the actions an agent can carry out, the communication level follows which uses grammatical principles to facilitate an almost natural exchange of information. The highest levels are a planner component and two meta-knowledge bases *introspection and partner modelling and learner*, where models about knowledge of others are and ways to achieve this are stored.

After all, this is an approach which may show a way to the desirable formalization of multi-agent scenarios.

3. Problem Solving Techniques Revised for DAI

The investigation of problem solving techniques is the central issue of Artificial Intelligence. Several paradigms have evolved and some of them have matured enough to be regarded as *foundations* of AI. This tendency now begins also for DAI. However, it is more natural to exploit the basic techniques of AI than to discover the wheel anew.
We first introduce four essential principles of AI, *search*, the use of *constraints*, the need for *heuristics*, and how to deal with *uncertainty*. Each will be discussed first from the general point of view, and then in the light of the DAI setting.
After this two paradigms will be presented which are of interest for distributed approaches. *Negotiation* is already a well accepted issue, whereas *eco-problem solving* is a pretty new concept.

3.1. Search

Search is one of the essential techniques for solving problems of any kind. A basic principle herein is the antagonism between the efforts for pure search and the amount of knowledge about the problem under investigation. It is obvious that more knowledge means less search.
An example. Suppose you come to the city of Kaiserslautern and you want to meet a friend in a certain street. It depends on the knowledge you have about the streets, places, buildings, etc. how long the search will take (assumed there is nobody whom you could ask for the way and you have no road map, too). If you are a stranger in the town, you will probably have to try many streets and ways until your reach your goal. In the worst case you run through the whole town - your friend may become nervous...
But assume, you have been here some years ago, say, as a student. You have lost the details of the town map, but you still remember some landmarks: salient buildings or characteristic corners. You know, the street you look for, is near by the city hall. You may go straight there (the building is high enough to capture permanently) and thus restrict your "search space" considerably. Even if you have to check the whole region near by the city hall, this is much less work than running through the whole town.
The other extreme is you have perfect knowledge about Kaiserslautern. In this case you do not need any search at all, because you already know the solution of your problem.
This example reveals some of the issues of search problems. We will only sketch them here. Good introductions are given in [Ni80, Ri83, Ri89].
The focus of interest shall comprise "large" problems with incomplete knowledge. For the other cases, e.g. when all possible outcomings can easily be enumerated, we do not need AI techniques.

This implies, AI techniques are suited for search problems, where the search space is infeasible large, too large for an exhaustive testing of all alternatives. Without any information about the domain, also AI techniques will fail. Thus even very little knowledge must be well exploited.

We distinguish between irrevocable techniques that do not permit a correction of a once selected way and those which do, known as tentative or backtracking strategies. All these strategies need at least two possible measures: One to assess the status of the current (intermediate) solution and one to get information about the next step to take. Whereas the value of the intermediate solution often can be determined (almost) exactly, this is not the case for the values to rank the alternatives for the next step. Since normally only estimations are possible here, heuristic functions have to be used. These should be simple enough to ensure efficiency, but on the other hand preserve an acceptable solution. Especially when the optimal solution is not explicitly required, this may be a possible strategy. It reveals again the tradeoff between quality demand for the solution and efficiency threads for the search process (for a closer look at heuristics cf. chapter 3.3.).

So far the short excursus about search in the traditional AI view. Where are the relations to DAI ?

Well, as mentioned above, knowledge is important to shorten the search process and (or) prune the search space. We argue, the distributed approach has the potential power to facilitate the unavoidable search. Provided an adequate distribution of knowledge both about the domain and about individual skills among the agents, search strategies may better be fine-tuned than it were possible from a central perspective.

Each agent may be equipped with individual search strategies, heuristics, and rating functions about the status of the scenario. The next steps of the search can be discussed and estimated from various points of view.

A general phenomenon seems to be that the individual search spaces of the agents are considerably smaller than the global search space. The local knowledge about own preferences, but also information about wishes of others (constraints for own decisions, see the next chapter) reduces the possible alternatives for the next step. A global planner cannot respect all these local dependencies, he/she/it must concentrate on the entities of the scenario which are able to act at all.

In this context, a criterion for distributed planning systems arises. If the search space is represented only in one central structure, planning is not really distributed, even in a multi-agent setting. Distributed planning takes place when also the representation of the search space is distributed, i.e. there are many "small" structures, where search can be performed much faster and cheaper.

3.2. Constraints

Formally, constraints in AI applications are nothing but n-ary relations $P(x_1,\ldots,x_n)$ with a special interpretation in the actual domain [Ri89]. They may be represented in a network in order to symbolize their mutual dependencies.

Constraints are very important in search processes (and thus in planning) because they are a useful means to keep the search space small. Only those values, which satisfy all constraints, are potential candidates for search. There are two ways of constraint propagation (in the following shortened CP) throughout the network, constructive CP and destructive CP. Constructive CP starts by calculating the values in some starting nodes. From these nodes the adjacent nodes are investigated. Thus, turn in turn, all nodes get appropriate sets of values. Sometimes assumptions about possible values must be made. These can be "backtracked" and replaced by new assumptions.

Destructive CP proceeds from the opposite direction. Here it is preliminary supposed that all values of the respective domains hold for all constraints. After successive checking of dependencies, more and more values are excluded; in the end only the valid values remain in the set. The results of constructive and destructive CP should be identical.

A multi-agent scenario with distributed control can also be regarded as a constraint net. But no rigid distinction between constructive and destructive CP can be made, because the nodes of the net are active. From a more abstract level we may say, the constraints themselves propagate through the net, no separate propagation instance is necessary. Expressed in the common vocabulary, we identify the agents (exactly: the knowledge of the agents) as constraints and the sending of information between them as constraint propagation. An agent itself knows what it may do and what it may not do for any single instant. An incoming message can restrict this possibilities, we have a destructive CP. On the other hand, the sending agent is active in telling its wishes, demands, etc. to the other agents. In this light we have a constructive CP.

An example [Ri83]. Consider the puzzle SEND + MORE = MONEY where each different letter represents a different digit between 0 and 9. The rules of arithmetic (e.g. overflows) must be obeyed. Let each different letter be an individual agent, which is informed about these rules. The agents may communicate in order to reach a solution, i.e. a consistent mapping from the eight letters to eight digits. What the agents will actually do is to propagate constraints.

Both ways mentioned above are possible. A constructive CP may begin with agent M's perception "I am surely the number 1", which it broadcasts to the other agents. These others need not take the '1' into consideration any longer. Sometimes tentative mappings are necessary, for instance, when agent S sends "I may be the number 8" after M's message above. If later the agents stuck in contradictions, some of the tentative mappings must be retracted, and S may say "I may be the number 9". In a destructive setting S

broadcasts "I am in the set {8, 9}" and other agents can rule out the alternatives which are contradictory to this configuration.
We can extend the constraint paradigm of multi-agent scenarios even more. Messages, which must obligatory be obeyed, (e.g. if the sender has a higher priority) represent *hard constraints*. Hard constraints immediately prune the decision spectrum of the receiver. Otherwise, it is possible that agents send merely advice or hints to other agents. These messages increase the information status of the addressees, helping them in more reflected acting. The addressees decide, after all, if they want to follow their given advice. This type of message can be regarded as *soft constraint.*
In our sample scenario in chapter 4. we will employ both types of constraints.

3.3. Heuristics

We have exact algorithms for almost all planning problems.[6] These guarantee to find the optimal solution. But as mentioned in chapter 3.1. about search, this often results in giant search spaces. We should be aware that solving the problem is not the crucial point, provided the solutions can be effectively enumerated. The simplest method is to produce one solution after the other and to compare each of them whether it is the desired one. This proceeding is known as the *British Museum Method*, because even a crowd of monkeys equipped with typewriters would have once produced all books of the British Museum, provided they had time enough to try [Ri83].
Of course this is absurd in larger search spaces. So we introduce heuristics in order to reach at least a pretty good solution. Heuristics are not exact algorithms, because exact algorithms must be infallible. Intuitively, heuristics are like rules of thumb: rough, generic principles that are not perfectly accurate, but are still accurate enough to be convenient and useful [Ha85]. It is important to be aware that heuristics may fail to attain the best solution. Sometimes they may find only poor results, but one characteristic of a good heuristic should be: the average result is well acceptable, the worst cases should never be reached.
Humans widely use heuristic procedures in normal life: Suppose you are driving to a supermarket and therefore you need to find a parking lot. You see one, pass it (there are surely some more nearer to the supermarket), pass the next, and so on, but once you choose one. Even there could be still more, you decide to stop the search. You unconsciously have calculated the tradeoff between the advantage of being nearer to the supermarket and the risk of finding no more parking lots. This is based on heuristics, not on exact algorithms [Ri83].

---

[6] Breadth search finds always the optimal solution if there is one.

Heuristics can be coarsely classified: On the one hand there are common sense heuristics for problem solving, on the other hand we need special heuristics, tailored to certain domains. Common sense heuristics, like rank all alternatives and choose the best one, are not sufficient. Mostly not the large steps in problem solving are the hard nuts, but the fine grained procedures. In the example above, how shall the alternatives be ranked, what is the criterion for the best one ?

In Distributed Artificial Intelligence there is a third class of heuristics. We call them *social heuristics*, because their intention is to manipulate somehow the social behaviour of the distributed agents working towards a common goal. Social heuristics do not mainly concern any specific domains. They model, for instance, generic ways to find a consensus in negotiations, principles of cooperation, and society rules.

These processes are inherently of heuristical nature, no exact algorithm can be found. Only past processes may be empirically evaluated and statistics may be formed, which provide a basis for a general trend prediction or an estimation of the future development. But this is merely a conjecture with a certain likelihood attached.

Social heuristics in multi-agent systems thus should help to simulate human social behaviour in order to realize certain general principles (benevolence, self-interest,...see chapter 2.4.). Since humans are flexible in their decisions, also these heuristics should be. Therefore we claim as an essential feature the ability of an agent to reflect about its own decisions. There should be a permanent checking process in the core of each agent: "What has changed in the environment ? - Are my decisions still up to date ? - Shall I correct my strategy ?".

For an operationalization of the social heuristics, especially for the aspects like negotiations, conflict resolutions, or mutual support, an extensive interdisciplinary approach seems to be necessary. Besides Artificial Intelligence, researchers coming from psychology, biology, linguistics, social science and other areas should engage in interchanging their results. Thus a concerted effort could fine-tune and revise the heuristics.

After all, a major problem seems to linger on: How shall the human's intuition be modelled ? Often we really cannot say why we did just this, and, that is amazing, often this step is essential to reach the solution. Minsky calls this phenomenon "thinking without thinking" [Mi86]. He states that before humans do not really know what is going on in their brains (or in their minds, respectively), they cannot build a machine with these capabilities. But he does not claim that this will ever be the case.

3.4. Uncertainty

The assumption of traditional planning approaches, that the planner knows everything about the "world", is appropriate only in mini-worlds (e.g. the Blocks World). In real scenarios plans must be elaborated without complete knowledge. In order to model uncertainty explicitly, AI employs several traditional methods from mathematics and has also proposed new ones. For instance, there are the method of the coarse sets, the Bayesian conditional likelihoods, the "Mathematical Theory of Evidence" of Dempster and Shafer, the theory of fuzzy sets, or the uncertainty factors of Buchannon and Shortliffe, utilized in the expert system MYCIN (A description of these methods can be found in [Ri89]).

To say it again, the purpose of these formalisms is an explicit modelling of uncertainty, which mostly results in a numerical value. A dangerous trap lies in the unreflected use of this value for further computations, e.g. the addition of two "certainty values" to a new one. For each special case it must be investigated what these calculations really mean.

All these issues can surely be utilized also in multi-agent scenarios. But the distributed setting provides more features to model uncertainty:

- An agent wants to know a specific fact. If it either knows that it does not know this fact, or if it cannot infer the fact from its own knowledge base, it has two possibilities:
  a.) If it knows, which other agents might know the fact, it may ask directly.
  b.) Otherwise it may broadcast the request for the fact to all the agents.
- An agent has a fact with an attached "certainty value" (see above). It may ask the other agents about the fact and thus correct its information about the fact.

We see, two aspects of distribution are important for managing uncertainty. First, there are the possibilities of exchanging the grade of uncertainty by communication and negotiation. Several sources of information about identical facts may considerably improve the overall certainty even if these sources are very uncertain from their individual point of view. The natural human exchange of uncertain knowledge in a problem solving group is modelled appropriately. Each one knows a "little piece", together the group can solve the whole puzzle.

On the other hand the knowledge about knowledge is of essential interest (epistemic knowledge). When it is missing, an agent must deduce a certain fact over and over from its knowledge base. Furthermore, the agent would not be informed about its capabilities, i.e. it could not decide if it is a proper "expert" for the query at all. While this could still be acceptable, some information about the other agents is absolute obligatory. This knowledge can be obtained by information exchange or it is given a priori. Other information which an agent cannot collect must be assumed. These assumptions (or beliefs) are valid as long as they do not contradict new facts. There are special reasoning

components which guard and update the assumptions and the corresponding dependencies, called assumption based truth maintenance systems (ATMS). There is ongoing research interest to distribute general truth maintenance systems (TMS). One approach is described in [HuBrAr90].

But what is more important are well-founded theories of ability, knowledge and belief of agents. Werner's recent work tends to this area [We90]. Especially his formalization of the principles of "can" (e.g. agent X can do the job Y) is interesting. Werner reduces "can" to "having a strategy for" and defines a strategy as a certain mapping of information sets $\mathfrak{I}$ to alternatives for $\mathfrak{I}$. Information sets are sets of several relations which describe the current status of the world. Thus the abstract *can* becomes a provable property. This could be a way to make abilities effectively testable so that the agents can improve their knowledge about their own qualifications and that of the others.

For the special issue *knowledge about knowledge* epistemic logics have evolved from modal logics. A new operator KNOWS is employed to model knowledge structures in the world. KNOWS (A,P) means that agent A knows the fact P; KNOWS (A, KNOWS (A,P)) means : A knows that it knows the fact P; KNOWS (A, KNOWS (B,P)) says that A knows that agent B knows the fact P, and so on.

3.5. Negotiation

When a group of humans is engaged in solving a problem together, normally more or less different views about the best way to proceed collide. Each human regards the problem from his/her individual point of view, each may have experience in different methods and strategies to tackle tasks like that. These are by far no bad conditions, just the opposite holds; such a setting is desired. The broader the spectrum of knowledge, experience, and opinions is, the more aspects and varieties of the problem can be foreseen. If all experts would have the same attitudes, the group would produce no better results than one expert alone, the process would merely last longer. An expert group lives through a rich spectrum of expertise. But at last a coherent solution should be presented. This demands at least two issues:

- the will of each concerned agent to reach a compromise
- the ability to reach a compromise (despite of all different points of view, a common basis must exist; e.g. it is problematic for a biologist and a historian to find a constructive agreement about a theme concerning computer science)

The process to reach an agreement or to attain a compromise is *negotiation*. Because it is pretty natural for human beings, it should be nearer investigated in general autonomous agent settings. The term negotiation already occurred several times in this paper. It was used in the context when two (or more) agents had to find a common way how to

proceed in favor of both of them. We will now encounter a more formal definition of negotiation.

A kind of "classical work" about this theme is an early paper of Davis and Smith introducing the Contract Net ([DaSm83], see also chapter 2.8.). The authors define negotiation as "discussion in which the interested parties exchange information and come to an agreement". This is a very general definition and gives no hints to special aspects of (artificial) multi-agent scenarios. Kreifelts and von Martial [KrMa90] come nearer to this end: They regard negotiation as "a structured interaction by message exchange between the partners involved in the negotiation". Here we get a first feeling for the main topics concerned in a formal account of negotiation: messages are exchanged between the agents in a structured, i.e. formalized way. We are back again in the terms of communication (cf. chapter 2.5.), but with one crucial addition: Negotiation must come to an end, a reasonable tradeoff between discussion of different opinions and the costs for communications is desired. It is important that even if no agreement was found after a certain amount of time, the proceeding can effectively go on.

There are different opinions where and how negotiations shou'd exactly be used in multi-agent scenarios. Davis and Smith [DaSm83] claim it mainly for the distribution of tasks which have already been decomposed. Negotiation is performed as a form of "market", where contracts between agents are temporarily built up to allocate subtasks. The emphasis lies on parallel execution of tasks as an organizing principle and on the transfer of control. A central goal is the uniform distribution of workload among the agents. [KrMa90] take another point of view. They point out the need for solving conflicts in preformed plans, that is on ensuring cooperative behaviour and to avoid interferences by asynchronous influences, like message delays etc.

The realization of the negotiation process offers a wide potential. Negotiation is principally a multi-lateral affair, i.e. a n:m-relationship is constructed. The Contract Net adopts this directly by allowing a "manager" to offer a task to several potential "contractors" and, on the opposite, allowing a potential contractor to send "bids" to several managers (see chapter 2.8.). For the next task, managers may be contractors and vice versa. Manager and contractor cannot force each other to a compromise, if no contract is reached, the manager must try it anew (may be with a changed announce) or quit.

[KrMa90] also do not use any force to reach an agreement. But here the negotiations are broken down to bilateral relations. This is achieved by a coordination agent. Each agent which wants to negotiate with others must get in contact with the coordination agent. The coordination agent then will converse one by one with the agents the first wanted to address. As mentioned above, the coordination agent cannot force any agreement, but only make proposals to the concerned agents.

One general demand to make negotiations possible in multi-agent scenarios is a well-defined and formal exchange of information. A negotiation protocol could achieve that. Certain requirements to the protocol are [KrMa90]:

- it should keep communication to a minimum
- the autonomy of the agents should be respected
  the resolution of conflicts must be facilitated
- it is desirable to make the protocol comprehensive for both human and artificial agents

After all, it should be obvious that negotiations cannot guarantee the optimal solution of a given problem. On the one hand, they are normally based on heuristics, on the other hand they manifest decisions, which may be non-optimal from a more global point of view. But as mentioned for heuristics (cf. chapter 3.3.) it suffices to find an appropriate consensus with reasonable effort.

3.6. Eco-Problem Solving

A challenging new perspective for multi-agent systems is to conceive them as eco-systems. An eco-system is an *ecological community considered together with the non-living factors of its environment as a unit.* The term "ecological" means: *of or having to do with the environments of living things or with the pattern of relations between living things and their environments.* [7]

But we do not commit ourselves to biological systems, also the association with economics shall be allowed. Abstractly we define an eco-system as a system of individuals with certain intentions and competing demands for resources. When the intentions can be almost overall satisfied, an equilibrium of activity will be reached after a while. By identifying intention with (sub)goals and equilibrium with stable goal state we get the direct connection to multi-agent systems.

Note that even if the aspects of the social behaviour and the common goals are somewhat relaxed, we can still assert an overall system goal for the system: All individuals shall be satisfied, i.e. the personal intentions must be fulfilled. When we now assume that in real eco-systems (where resources are limited) normally an absolute selfish acting of the individuals seems to be impossible, we are back again to (at least some) social aspects.

But what did we gain by the "eco" metaphor ? First of all, we get a well suited model to explain the behaviour of distributed systems without a centralized control. Movements which appear at the first time chaotic, become explainable.

Then, and that is a hope, a new metaphor arises for a special kind of distributed problem solving with a special class of corresponding problems. The classical distinction of DAI

[7] Both definitions were taken from Webster's Third New International Dictionary.

systems between behaviour based (agent which only react upon influences from their environment) and knowledge-based systems (intelligent agents, which actively plan their actions) may be exposed as inappropriate. We will now try to substantiate both the theses just mentioned.

To explain multi-agent scenarios without centralized control, Kephart et al. performed exhaustive simulations [KeHoHu89]. The authors also use the eco-systems paradigm. The central characteristics of "computational eco-systems", as they call their settings, are in their opinion:

- distributed control
- asynchronism
- resource contention
- extensive communication among the agents
- incomplete or delayed information

All these presumptions match almost perfectly general conditions for multi-agent systems we have mentioned in this work up to now.

The intentions of the agents in eco-systems represent the desire to attain a specific amount of limited resources. This is nothing but an abstract description of conflicting goals. Seen in this light, all DAI scenarios without central control are eco-systems in any way. But this is the second step before the first is completed. As a matter of fact, Kephart et al. found a behaviour similar to that of a living population in a real eco-system. They distinguish three classes:

① non-oscillatory relaxation to the equilibrium
② damped oscillations about the equilibrium
③ persistent oscillations

The equilibrium can be identified with a stable state, where all agent are sufficiently satisfied. If the eco-system has an overall social goal, a tendency as in case ③ above should be avoided. A stable solution presupposes an equilibrium in resource consuming.

Kephart et al. [KeHoHu89] extracted two interesting conclusions from their various measures of computational eco-systems:

- They detected long living meta-stable states for certain combinations of parameters. A system, which appears to be stable, may rapidly jump into a real stable state. This implies, an eco-system does not guarantee a smooth and steady behaviour, but in most cases it conducts a such.
- Kephart et al. also provided the individual agents with more information about the world (procedural knowledge about time sharing, etc). In contrast to their assumptions that the performance of the system increases with the increase of the local knowledge,

just the opposite happened from a certain point on: The overall results became worse, sometimes even worse than in a system where agents have no special information at all.

The authors conclude that a distributed system without central control must not have too intelligent individuals. A chaos may be induced by overly-clever local decision making algorithms. On the other hand the authors found that imperfect knowledge (i.e. a higher amount of randomness) suppresses oscillatory behaviour of the system at the expense of reducing the average performance.

We must keep in mind that all the results are based on statistics. They hold for the average case. Each test run, however, may exhibit completely different results.

Now to the second thesis mentioned above. Ferber [Fe90] proposes a new paradigm "eco-problem solving" for a certain kind of DAI scenarios and problem configurations, especially for settings where agents have to arrange themselves in a desired manner (e.g. Blocks World problems).

Ferber does not formally define eco-systems, but he describes the attributes of the agents ("eco-agents") incorporated, i.e. their knowledge, their principles of cooperation, and their behaviour.

A strict distinction between domain-dependent and domain-independent knowledge is made. The first must be adapted anew for each scenario, whereas the other knowledge holds for all possible applications. It comprises:

- an agent knows, when it has reached its goal ("satisfaction state").
- an agent knows, when other agents prevent it from acting (its "jailers").
- an agents is informed about dependencies to other agents, i.e. if it must wait with acting until another agent has done anything.

The latter two issues are in permanent change while the scenario runs.

The behaviour of the agents is characterized by three attributes:

- The will to reach its goal. Each agent actively works towards its satisfaction state. Thus an eco-system is an active, not a re-active system.
- The will to be able to act. The agents permanently try to remove obstacles, which also may be other agents.
- The obligation to flee. If an agent is an obstacle for another, and this other agent wants to act, it sends the second one off. This is performed by message passing. The second agent must go away, it may decide to go anywhere the first agent does not want to go. Here we have a behaviour like in reactive systems.

Provided with these means, knowledge about the initial configuration and informed about the desired goal, each agent tries to reach satisfaction. It is characteristic that through the messages used to send other agents away, dependencies are constructed. These dependencies prevent the agents sent off (slaves) from reaching their final goals before the agent, which announced the message (their master) has reached its goal. So, if the final situation can be characterized by a partial order of dependencies (the order, in which the agents may reach satisfaction), there is good hope that the overall solution will be found and eco-problem solving will terminate.

Ferber characterizes eco-problems (problems which can be adequately solved by eco-agents) as those which can be described through a triple $(\mathcal{A},\mathcal{S},\mathcal{G})$, where $\mathcal{A}$ is the set of agents, with both domain specific and "basic" knowledge, $\mathcal{S}$ is the start configuration, $\mathcal{G}$ is the goal configuration, and $\mathcal{S}$ and $\mathcal{G}$ are individually ascribed to each agent.

As a conclusion we state that the paradigm of eco-problem solving seems to be well suited for problems, where the goal situation can be a priori decomposed to all individual parts of the problem. If these parts then get the capability mentioned above, they will find the way to the solution alone, if possible at all. Hence the eco agents perform also distributed planning and acting alternatively. The planning part lies mainly in the decision where to go, when sent off by another agent. There could be some intelligence in this step (but maybe not too much, see the results of [KeHoHu89] above !).

We introduced eco-problem solving broadly,because it is just the way how we realized our sample scenario in chapter 4. Amazingly, we used almost the same "framework" like Ferber, without knowing his paper or having ever heard the term *eco-problem solving*. But, alas, Ferber named it as such and we consider it useful.

At last, an interesting prospectus from [KeHoHu89] shall be mentioned: Biological eco-systems can adopt to a changing environment through feedback mechanisms. It would be a challenge to transform these abilities also to computational eco-systems. Feedback must therefore induce an adaptive behaviour, like learning. Thus it may be possible to evoke an "evolutionary process" and to come to better problem solving techniques for multi-agent systems.

4. The Towers of Hanoi - A Distributed Planning Scenario

The scenario we have experimented with is a variation of the well-known puzzle of the Towers of Hanoi. At a first glance this appears rather simple. Every computer scientist has to learn the short recursive "Towers of Hanoi" algorithm in one of his first lessons.

```
algorithm Towers_of_Hanoi (no_of_disks, start, goal, aux)
        {if no_of_disks > 0
         then  {Towers_of_Hanoi (no_of_disks – 1, start, aux, goal)
                Print ("Agent from ", start, " to ", goal)
                Towers_of_Hanoi (no_of_disks – 1, aux, goal, start)} }
```

But this is not our approach. On the one hand we extend the scenario by increasing the number of available sticks which makes the algorithm given above no longer optimal, on the other hand we leave the global view and distribute control and information. Decisions are made decentral and their impact on the global situation is partly unknown. We are directly confronted with a DAI problem.

The Towers of Hanoi scenario (in the following shortened ToH scenario) covers several of the aspects mentioned in the last two chapters. An emphasis will be laid on heuristics, negotiation, and eco-problem solving, besides the more technical details of the realization. The scenario is simply enough to overlook but nevertheless should prove sufficiently complex to reveal clearer insight into important concepts of DAI.

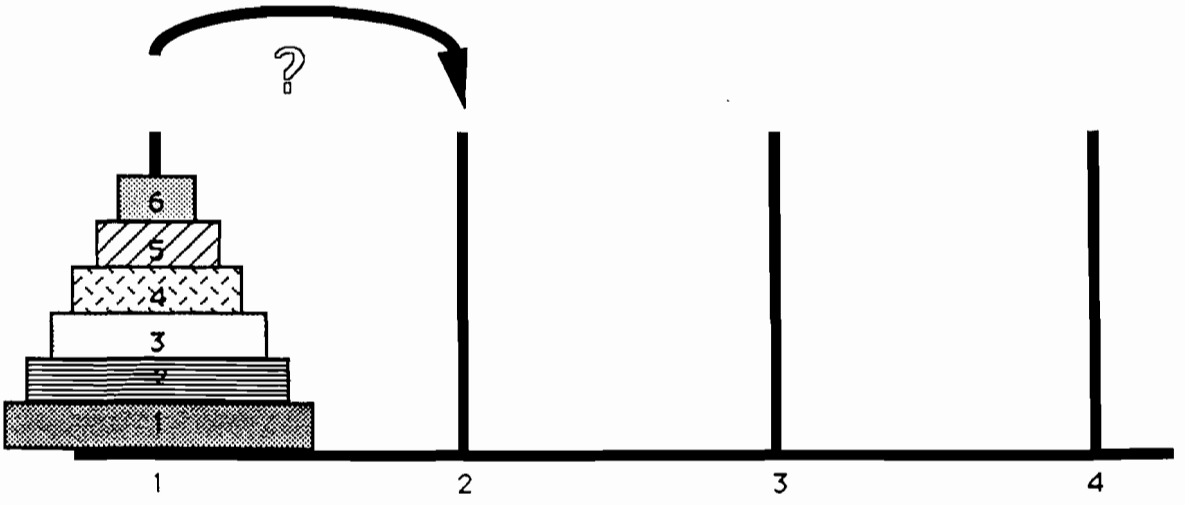

Figure 4.1 : Start situation of a possible ToH scenario

4.1. Description of the Scenario

Essentially we have to explain two topics. First of all, how the ToH scenario can be regarded as a multi-agent scenario, and secondly, why it is an example for planning. Figure 4.1 shows a sketch of the world we are going to talk about.

Each of the moveable disks shall be seen as an *individual agent* with own knowledge, own abilities and own tasks (resp. goals). The knowledge comprises very local knowledge (who is directly over me ? - where am I now ? - what do I have to do next ? ...) and knowledge about the access to more global information (do other agents interfere with me ? - can I reach my desired goal ? - what are the best alternatives for the next move ?...). The abilities of the agents are mainly sending messages to other agents and acting if all obstacles are removed. Note: The agents are able to move themselves, no other agent (e.g. like the "hand" in the Blocks World) is necessary.

The principles of the problem solving process resemble the aspects of eco-problem solving (cf. chapter 3.6.). The agents are egocentric, they try to make their way straight to their desired goals. On this way they are confronted with two types of obstacles:

- An "upper neighbour" $\mathcal{U}$ might lie upon a waiting-to-act agent $\mathcal{A}$.
  In this case $\mathcal{A}$ (whose next goal is place **X**) will send the message "Leave your place, but do not go to place **X**" to $\mathcal{U}$ .This simply avoids a further confrontation between $\mathcal{A}$ and $\mathcal{U}$ at place **X**.
- Another agent $\mathcal{B}$ with a lower priority than $\mathcal{A}$ might be situated at $\mathcal{A}$'s next attempted place, say **X**, and thus prevent $\mathcal{A}$ from going there.
  $\mathcal{A}$ (who now resides at place **Y**) sends the following message to $\mathcal{B}$: "Leave your place, but do not go to place **Y**". $\mathcal{B}$ shall not block $\mathcal{A}$ as an upper neighbour.

The message "Leave your place, but do not go to place..." is the central concept of this approach. The agents force others to step aside to pave the way for their very own goals. But we should be aware that the execution of such an order is not deterministic at all.[8] This is the key issue to intelligence. The possibility of free choices offers chances to bring in decision procedures and other auxiliary facilities, like heuristics, which will speed up the problem solving process.

We consider the ToH world as a sample for distributed planning. The agents have to coordinate their next moves, clashing subgoals must be resolved. As we have just heard some sophistication can also be put into decisions where better not to go. All this strongly touches aspects of planning. The identity of planner and actor is typical for distributed planning (cf. chapter 2.7.3.). Each single agent plans its very own next steps within its

[8]This is true for a ToH world with at least four places. It is clear that in the case of three places (the "standard scenario") no option remains. At one place the agent resides momentarily, one other place is forbidden by the orderer of the "leave-message", finally, the third place *must* be chosen for the next step. Hence the three-place ToH problem has a deterministic solution and does not require intelligence.

partial world view. After planning (which can only be partial) the agents try to fulfil their restricted plans. Later we will see more detailed how such a plan could look like.
But also social capabilities are demanded, the willingness to help each other in the sense of a social system (cf. chapter 2.4.). The ToH scenario provides facilities for help requests and gives priority to attempt a satisfaction of this requests before trying other alternatives.
These are the aspects of the ToH scenario under investigation:
Agents: Each disk is an intelligent, active agent. The agents are unique and identifiable by their names. Their number is unlimited, but fixed. All agents have got a different priority value (their "size"). Furthermore they have knowledge about their goals and also partial knowledge about the world. If necessary, they may get further, more global information.
Places: The number of places (i.e. sticks) is unlimited but fixed. The smallest possible number, however, is three, otherwise no reasonable moves can be made. The places are not an active part of the scenario, the agents must take care for correct moves and update the places, too.
Positions: The term "position" shall denote the position of an agent in the stack at a certain place. The agent on the bottom of the stack has got position 1, the next upon it position 2 and so forth. Thus the maximum number is the total number of agents in the scenario.
Goals: At the start of the problem solving process all agents who are involved in the final configuration must have received their final goals. Only the individual agent knows his special goal.
An important presumption is also the demand of compatibility and consistence of the subgoals with respect to the global goal. The global goal (generally more than the sum of its subgoals, cf. chapter 2.) cannot be achieved unless all subgoals can be reached. This implies a full check for consistence through an external instance before the detailed subgoals are given to the agents.
Parallelism: The ToH problem is in a class of problems, which can inherently be solved in parallel. The agents may pursue their (sub)goals contemporary and may try to perform actions always when they think it is possible.
Constraints: No agent can "hang in the air" except in the moment when it is acting. We have to obey the simple physical laws like gravity, uniqueness in the parts of space which are covered by the agents, etc. The main aspect is that the agents need permanent support. This is achieved either by other agents which tolerate them or by the "ground" which supports everyone.
Constraints are also important to restrict the search space for the next decision of each agent. All information an agent has about the intentions of others, constrains its own choice.

Capabilities of agents: The agents send messages to themselves and to other agents. The primary purpose is to inform about their own intentions, but also to remove obstacles and to negotiate about conflicts emerged. Last not least, the abilities to perform the planned moves and to do the bookkeeping shall not be forgotten.

Behaviour: The agents send off those agents, who prevent them from reaching their goals. The latter have certain degrees of freedom to choose a place where they could flee. Into this choice as much intelligence as possible must be brought in. The behaviour is always motivated. Either the agents pursue their own goals or they act in order to obey other agents' wishes. Though especially big ToH scenarios (many agents and places) often resemble somewhat the behaviour of ants in an ant heap, every single decision is well-founded and correct from the individual's point of view.

Priority: Each agent has got a unique priority value, so all agents are in a linear priority order. Especially all agents are comparable. In fact, the "bigger" the agent is, the larger is its priority value.

Toleration: An agent which wants to go onto another agent, must be sure that the second one tolerates itself. Another term would be "capable to carry". In the ToH scenario agents tolerate others if and only if they have a higher priority.

Heuristics: Because the choice of the agents sent off is non-deterministic, heuristics are a convenient way to find a good solution. We will make an exhaustive use of heuristics and try to prove the appropriateness in benchmark tests.

Social aspects: The final goal to build a new tower on a different place is a social goal [We88] in that the agents with lower priority depend on the foundation of the "higher" agents and, on the other hand, the bigger agents must hope that the smaller agents really go away if they are obstacles.

The foundation of benevolence is built in the ToH scenario. *Orders* of other agents ("go away, but avoid...") are always obeyed and *wishes* and *petitions* as much as possible. Help requests are considered privileged by ranking the alternatives (in TOHPAR). But the agents are not altruistic at all. In doubtful situations, if they cannot combine benevolence with their very own intentions, they act selfish. This is a key issue to force the aspiration to reach the goal.

4.2. Why Not Conventional Planning ?

We could try to plan the way from the start situation to the goal situation in the Tower of Hanoi scenario with conventional methods of planning (traditional planning , cf. [St87, He89]). It seems to be straightforward to construct first a graph of all possible states (see Figure 4.2) with the start configuration as start node and the respective transitions as edges, and then look for the node representing the final state. Dijkstra's algorithm returns the shortest path between the states. But taking a second glance onto the extension of the graph reveals problems. The absolute number of states **S** is finite, however, it grows exponential. Let n be the number of disks and m be the number of sticks, then we have **S** = m^n. For n = 10, m = 4: **S** = $4^{10} = 2^{10}*2^{10} = 1024*1024 > 1{,}000{,}000$! And this is a very small configuration. Higher parameter values soon exceed the storage capacity of any computer.

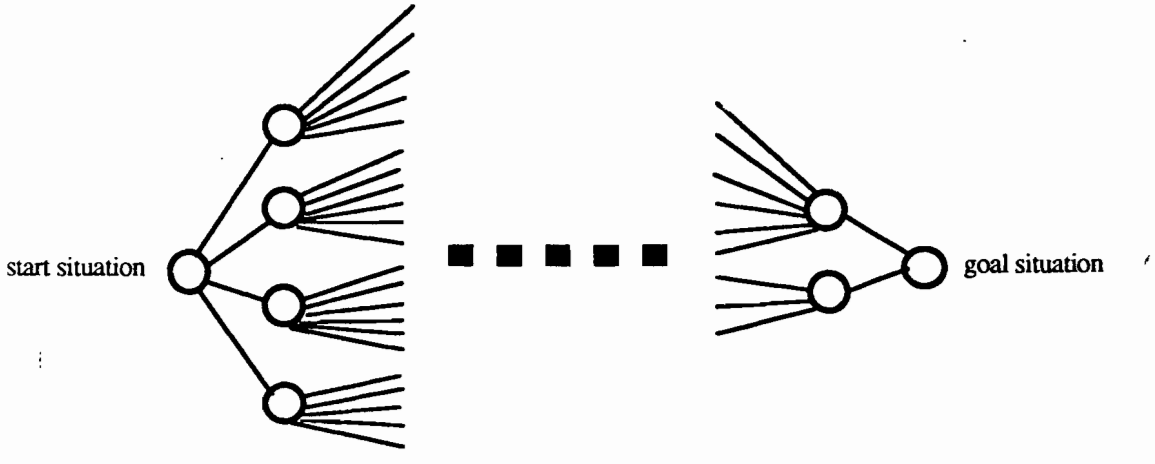

Figure 4.2 : The search graph of the ToH problem is infeasibly large

Another method is search [Ni80]. If we use simple hill-climbing methods, we lack of a pretty good heuristic function, which estimates the difference between the actual state and the goal state. Some very special rules of thumb are possible, like "one disk at its goal stick in the correct position is better than none" or "try to remove the obstacles from the largest disk which is not yet at its goal". But most of all adjacent states cannot be compared, let alone be ranked by a function using more abstract principles.

So, what still remains is graph-search (see Figure 4.3). For the same reasons as mentioned above for hill-climbing, the branching rate can hardly be restricted. Also best-first strategies are infeasible therefore. Let us investigate the possibilities for depth-first and breadth-first search. What is the average branching factor ? If the towers are high, it tends to m–1, otherwise, when all disks are distributed among the sticks it is n*(m–1)

without parallel moves (n : no. of agents, m : no. of sticks, like above). A coarse estimation for the average factor is (n*m)/2, which is rather too low because of the various combinations of parallel moves.

If we now try breadth-first search, we must take into account the total number of expanded nodes $\mathcal{N}$ (breadth-first will surely find the solution, the question is for what costs ?). If we assume that the solution is found in d steps in the best case, the search tree comprises about $\mathcal{N} = (n*m/2)^d$ nodes.

An example: Let n = 10 and m = 4 as in the example above. We take d = 50 as an empirical value from test runs. Then we get $\mathcal{N} = (10*4/2)^{50} = 20^{50}$ which is absolute infeasible.

Depth-first search as an alternative is also impossible, because the depth of each branch is not finite. Testing whether some configuration has already appeared before, would cut the branches. But this is expensive both in time and storage. The whole branch must be stored and one by one compared with a new node.

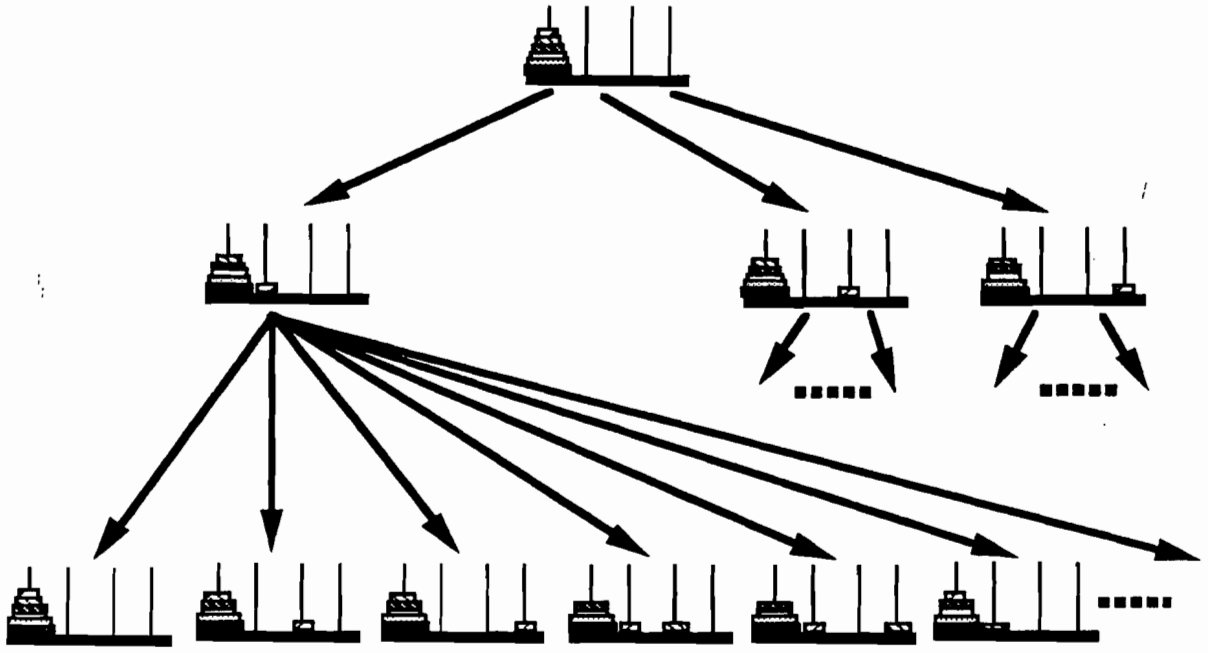

Figure 4.3 : Branching of the search-tree can hardly be restricted. It grows enormously.

After all, the search problem of the Towers of Hanoi cannot be tackled with conventional methods. We must find better solutions.

If we distribute control among the disks themselves and let them make decisions which are optimal from their local point of view (ongoing with negotiations and conflict-resolution strategies) we rid ourselves from the burden of finding a global ranking function for the states.

Important properties which guarantee the feasibility of multi-agent planning in the ToH scenario are:

- The search space in confluent, i.e. each two branches from an arbitrary node can be joint again. This is because each configuration can be constructed from each other and everything may be reversed again.

  A consequence is that backtracking is obsolete. We do not need to go back in the search graph, but may attain the same result by going forward and simply performing just the opposite moves as before. As in "real life", wrong moves ("deeds", "behaviours",...) must be actively cured or improved, they cannot simply be "withdrawn", as if they had never occurred.

  This implies the possibility of interweaving search and moving, in other words of planning and execution. It is characteristic for multi-agent planning, that a plan normally is not completely developed and afterwards executed. In the ToH scenario the extreme of planning only *one* (i.e. the next) step with following execution is promoted.
- The common goal can be efficiently decomposed into individual subgoals, that if each agent (disk) has reached its personal goal, the overall goal is satisfied.
- Even if the disks directly perform their planned moves towards the goal, this may be regarded as a proposal. If these steps are recorded, post-optimizing becomes possible, which may delete unnecessary cycles or enhance the parallelism.

For these reasons we model the ToH problem as a multi-agent planning approach. Essentially, it is an example for classic heuristical planning. What is new and challenging is that the heuristics (and the rest of knowledge) is not any longer concentrated in one central instance, but distributed among the entities, which are in the scenario themselves. Thus we could regard our agents as "experts" for their very own limited domain.

The ToH scenario is realized in two versions which succumb distinct paradigms: One uses sequential message passing whereas the other allows for parallel sending of messages. In the sequel both versions shall be considered in some detail. After introducing common aspects, the sequential version TOHSEQ and the parallel version TOHPAR are described. Next the versions are compared with respect to their test results, their costs and their adequacy for modelling the scenario. At last some implementation details are presented.

We will lay emphasis on the parallel version, after all it is the aim of all our investigations. So all prior results may prepare the way for the parallel scenario.

4.3. Common Aspects of the Realization

First of all, some general principles of DAI problems like the ToH scenario shall be outlined. In each state of the problem solving process, each of the components (i.e. agents) has got an individual goal, but only a subset of the components is really able to act. The elements of this subset can mostly decide what to do next by choosing one next goal out of a set of possible alternatives. These decisions should be as clever as possible; not only the individual situation should be taken into consideration, but also the goals and the intentions of other agents.
In the ToH world we have (and presume, respectively) some issues, which provide simplifications for otherwise very complicated settings:

- If a component finally has reached its goal, it surely has not to leave again [9].
- Actions and planning is performed synchronized, i.e. all agents need exactly the same amount of time for these two phases, no agent can overtake another. This implies we have discrete states and discrete transitions between the states, too.
- The agents behave cooperatively and try to help others as much as possible.

The sequential and the parallel version of the ToH scenario are based on a uniform implementation environment, i.e. the same language and the same machine. Hence we have a solid base for developments from one into the other and also for comparisons. For the user both versions appear almost identical except that some parallel moves now and then occur, which are not possible in the sequential version. Anyhow, often the parallel moves are only perceived in slow-motion. But under the surface the realizations differ considerably. Well, what is common ?
Programming paradigm: We use an object-oriented programming paradigm. That means that entities in the scenario are regarded as instances of object classes. Each object class has certain methods attached, which serve as the procedural knowledge of the members of the class. The declarative knowledge is stored in slots of the instances, the slot names are defined with the objects class. More mathematically, an object instance can be seen as a n-tuple, where each component has a specified range of possible values.
Modelling of the agents: Each agent is considered as an instantiated n-tuple (A_1:v_1,..., A_n:v_n). The names A_i of the components shall be called slots. Each slot can be instantiated with a certain value v_i, which is either an element of the range of A_i, or NIL (i.e. no actual specific value given). The slots represent the entries to the knowledge base. Strictly spoken, each agent is a knowledge base with procedural attachment, the methods. An individual agent is created by making a new tuple "agent" and filling the

[9] Assumed we have a "traditional ToH problem" like "Move the whole tower from A to B". In a setting where the goal configuration comprises more towers, the disks probably have to leave sometimes their final position again.

slots properly. The number of slots is fixed, hence the aspects and the amount of knowledge are inherently restricted. As mentioned above, not each slot must have a specific value, NIL-entries indicate either ignorance or a lack of interest for certain aspects.[10] An agent owns at least these aspects of world knowledge:

tuple AGENT:	(name, place, position, blocker)
name	: unique identification
place	: actual place
position	: exact position at place
blocker	: actual agent which is located on the instance itself
Example (ToH):	(name: agent-4, place: 3, position: 2, blocker: agent-7)

The procedural knowledge (hidden in the methods) comprises information about when and how to get more global information. This will be discussed separately for both versions later.

User dialogue: The user is asked to start the scenario. He must declare the number of agents, the number of places, whether he wants a statistic evaluation, the starting place, the place to finish and the degree of slow motion. Changing the direction of the output (information about status and action resp. statistic information) is also possible. Further parameters depend on the versions.

Graphic output: Principally the graphic output is identical in both versions. The agents appear as black rectangles and perform moves in three steps (up - to new place - down). The places are perpendicular lines to sketch the "sticks".

[10]These kinds of uncertainty do not yet emerge in the current model of the ToH scenario. But a missing of a goal, for instance, is already integrated. Agent without goals would only passively react to orders of others.

4.4. The Sequential Version : TOHSEQ

4.4.1. General Phenomena

In the sequential ToH scenario TOHSEQ there are some inherent simplifications with respect to a parallel version.

At each instant of time only one agent can be active, the rest is "frozen" and does nothing. The common goal is built up stepwise; the correct order of the subgoals is implicitly given by the priority of the respective agents. It is impossible that Agent "x" is able to go to its final goal unless Agent "x-1" has already reached its.

The priority of the agents corresponds to their size (a total order). First the biggest agent (Agent "1") receives its goal and pursues it by sending blocking agents off. Agent "1" owns the highest priority. After it has reached its goal position, the next agent (Agent "2") starts active working. This principle is also used recursively when sent-off agents pursue any subgoals (where they want to flee) and therefore send other agents off, which are obstacles for them. Thus in the moment one agent $\mathcal{A}$ succeeds in reaching a goal/subgoal just the one which previously caused $\mathcal{A}$ to act, can go on in working towards its next goal.

For the aspect of control, at each moment at most one agent, say $\mathcal{A}$, really pursues its original goal, all others stick on subgoals to pave the way for $\mathcal{A}$. When $\mathcal{A}$ reaches its goal, the next agent - the one who waited for $\mathcal{A}$'s completion - may act. Control passes over if one agent invokes a method from one other. Thus we have an implicit control flow in TOHSEQ, guided by the priority of the agents.

Sending of messages (evoking of methods from another agent) actually works like a remote procedure call: The addressee will carry out the ordered "service" at once but the sender must always wait for completion. Figure 4.4 shall clear this up (according to [Ne86]).

If an agent has decided where to go next (new subgoal), it may wait until all obstacles are removed. This "removing of obstacles" does not concern the agent in any way; especially it does not interfere with the agent's planned move.

As a consequence, there is only one task to perform at each instant of time. No internal conflicts about goals occur, because an agent pursues maximal one goal/subgoal.

If an agent is able to act from its local view, this holds also for the global point of view because no parallel action can interfere with the agent. This implies that no external conflicts (conflicts with other agents) are possible. Hence no external component to manage these conflicts is necessary.

Conflicts thus do not appear obviously. The strategy of conflict resolution is built into the heuristics employed. By exploiting the maxim of "most intelligent decisions possible from each agents point of view" the heuristics are founded on preventing conflicts.

Agents are sent off only in the case if no other alternative has remained, and moreover, only those agents which really are obstacles. Thus silly conflicts are avoided. Conflicts concerning the order of the goals or the priority of the agents are excluded from the first because of the reasons mentioned above.
To draw a conclusion, the main facility for control and conflicts in TOHSEQ is the principle of sequentiability. We do not have to take care, for instance, for coordination and synchronization aspects. This issue will dramatically change in the parallel ToH scenario.

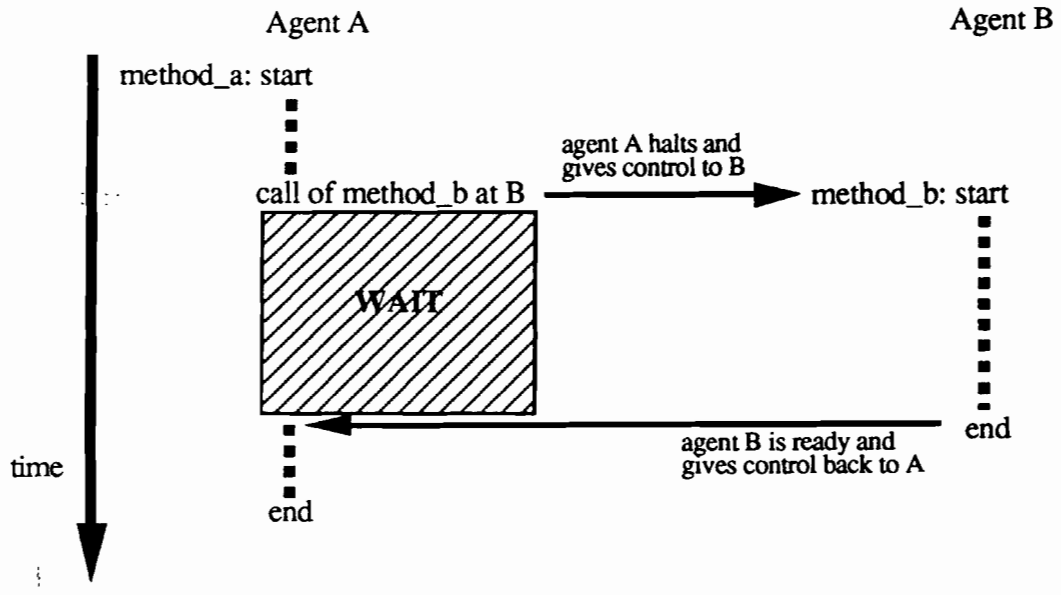

Figure 4.4 : Message Passing as a Remote Procedure Call

4.4.2. Architecture

We consider *architecture* as "the structure and team-work of the particular components of the entire system". This concerns primarily the structure of the procedural knowledge of the agents besides the declarative aspects like the knowledge bases mentioned above. It is important to know how the agents' methods interact, what they cause and how they are invoked. The scenario is started from outside, however, once it runs, the agents rely completely on their procedural knowledge to solve the problem. So it is absolutely necessary to get a thorough understanding of the system architecture to be able to follow the agents' decisions. There are two points of view regarding methods and sending of messages, an abstract one and a concrete one (Figure 4.5).

	abstract view	concrete realization
methods	Abilities of an agent to solve a certain well-defined problem	procedures which cause side-effects (mostly in changing slot-values)
sending of messages	Sending of information to another agent, make use of the abilities of whom (use a service like a client)	Remote procedure call (interrupt of the sending procedure)

Figure 4.5 : The abstract and the concrete view of messages

4.4.2.1. Abstract View

Each agent merely needs a very simple control structure to execute a given task. It consists of four steps, which are iteratively performed until the agent has reached its goal:

① Determine the next (sub)goal

② Try to become able to act (remove obstacles)

③ Reflect about your decision and perhaps modify your subgoal

④ Perform the action

Constraints are the basis for the decisions in step ① and ③. An agent knows its own goal, but has also information about the wishes of other agents. This can be regarded as a transmission of the constraints of the other agents to the first one. Because of its cooperative behaviour, the agent tries to obey the others' intentions as much as possible.
In step ② an agent sends orders to go away to agents, which are obstacles for it. The agent hopes that it can act when these obstacles have left their place. But because of its limited view, this cannot be guaranteed. Possible side effects must be restricted, this is another duty of constraints. An order in step ② has informally a format like "go away, but do not go to place X, because I want to go there". Thus the new constraint is "place X is forbidden".
Step ③ only occurs in the most sophisticated heuristic. In this case, the steps ②,③ build a loop, which lasts until the agent really can act. For the other heuristics, step ③ is omitted and the agents wait in step ② until their obstacles have gone away.

Heuristics are used in all steps except step ④, where no decision has to be made anymore.
The result of the whole proceeding (the simulation of the agents) is a sequential plan. This plan can either be used directly, or be post-optimized in order to find possible parallel moves.

4.4.2.2. Concrete Realization

TOHSEQ utilizes directly the facilities of sequential message passing. Sending of messages works like a remote procedure call: The addressee will carry out the ordered "service" at once but the sender must always wait for completion.
The format of the messages is a list consisting of *addressee*, *message type*, and n parameters $par_1 .. par_n$ with $n \geq 0$. The actual parameter of *addressee* must be an instance of the class of agents where the message type is attached to.
We use message passing with the objective to implement the central idea of the agents' desire to cause blocking agents to flee: "Leave, but do not go to the place where you would block me again".

The sequential scenario TOHSEQ consists of one main function "hanoi" which starts the scenario and several methods which are called from the main function and from other methods or agents, respectively.
The call structure of TOHSEQ is depicted in Figure 4.6 (note: *hanoi* is a function, all other identifiers denote methods).

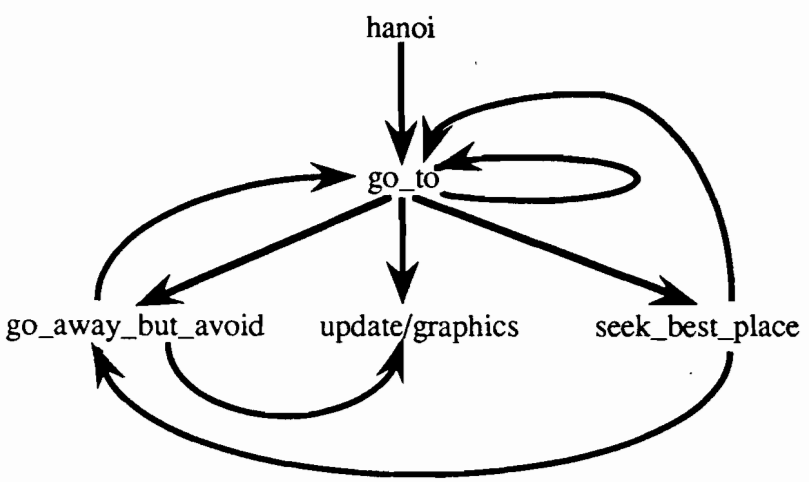

Figure 4.6 : Call Structure of TOHSEQ (especially *hanoi*)

In the following we present the sequential TOHSEQ system by means of flow diagrams (Figures 4.7 - 4.10). The arrows stand for control flow and the bold rectangles symbolize the sending of a message (The control passes over to the called method, it comes out again from the bold rectangle and goes further into the next arrow when the stop-label of the called method is reached). The technical methods *update* and *graphics* are not decomposed further.

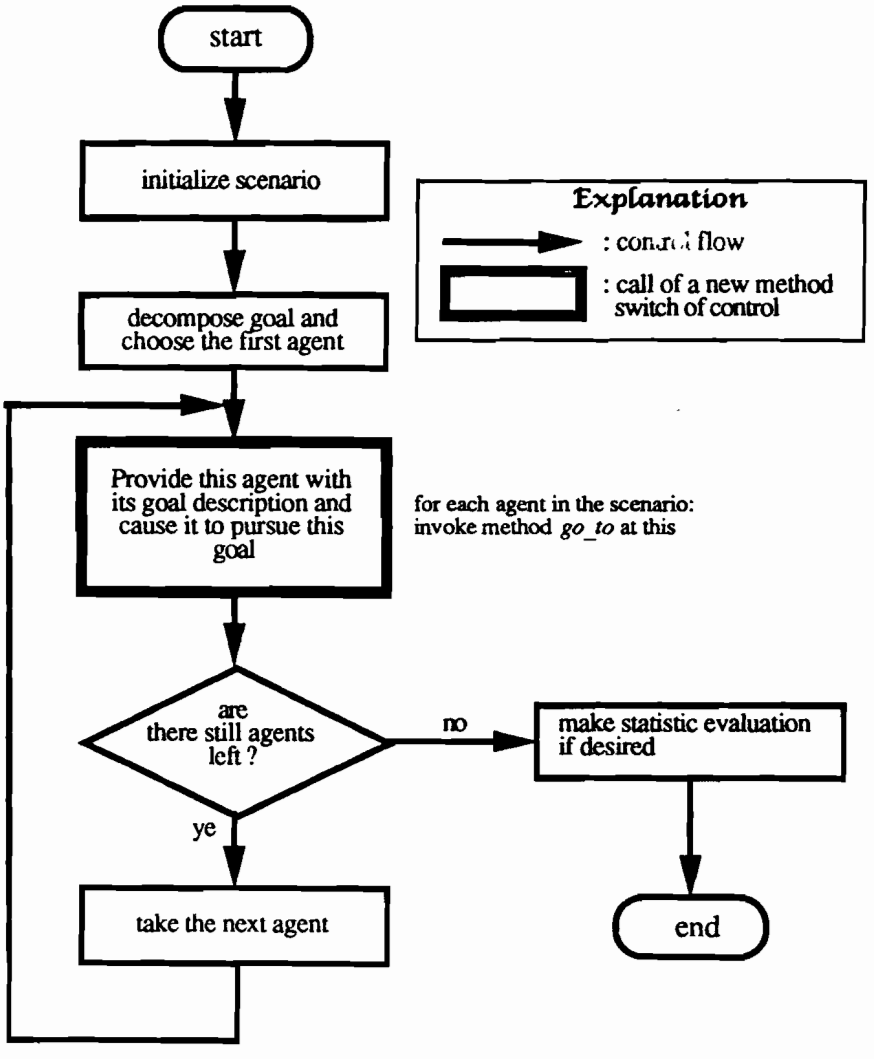

Figure 4.7 : Top Level of TOHSEQ

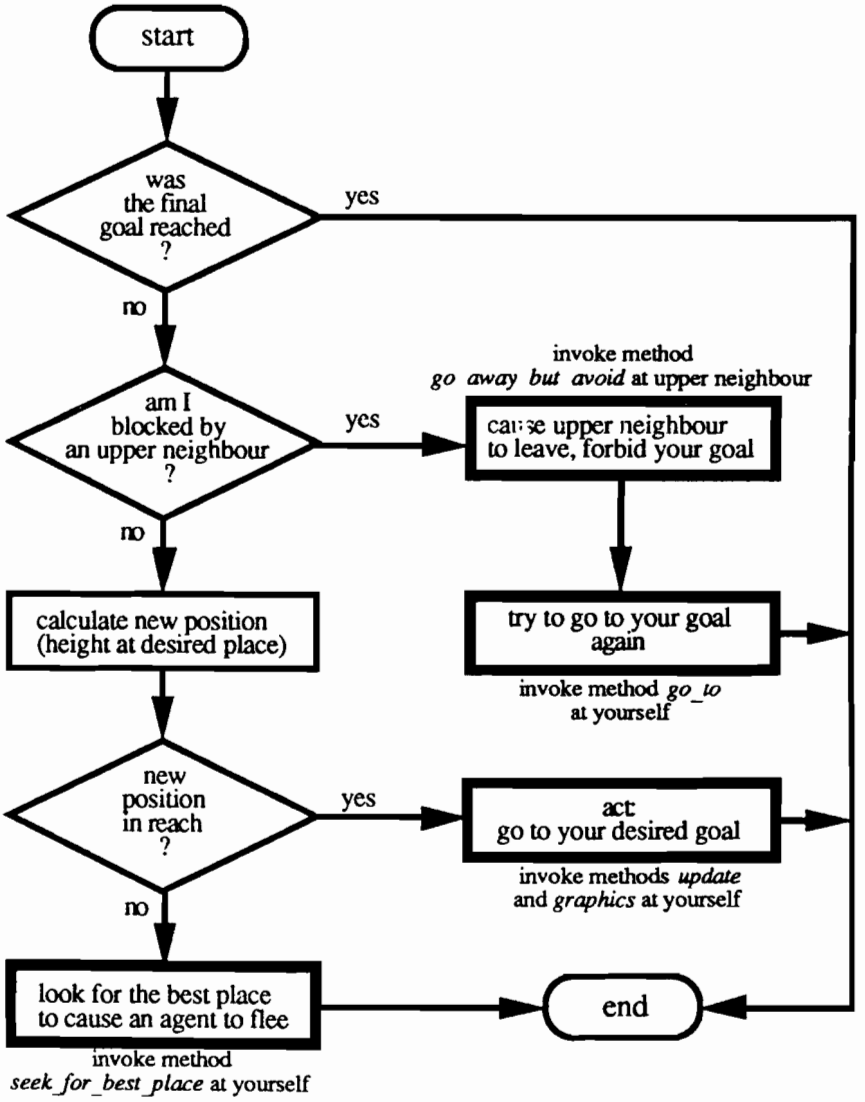

Figure 4.8 : Method *go_to*

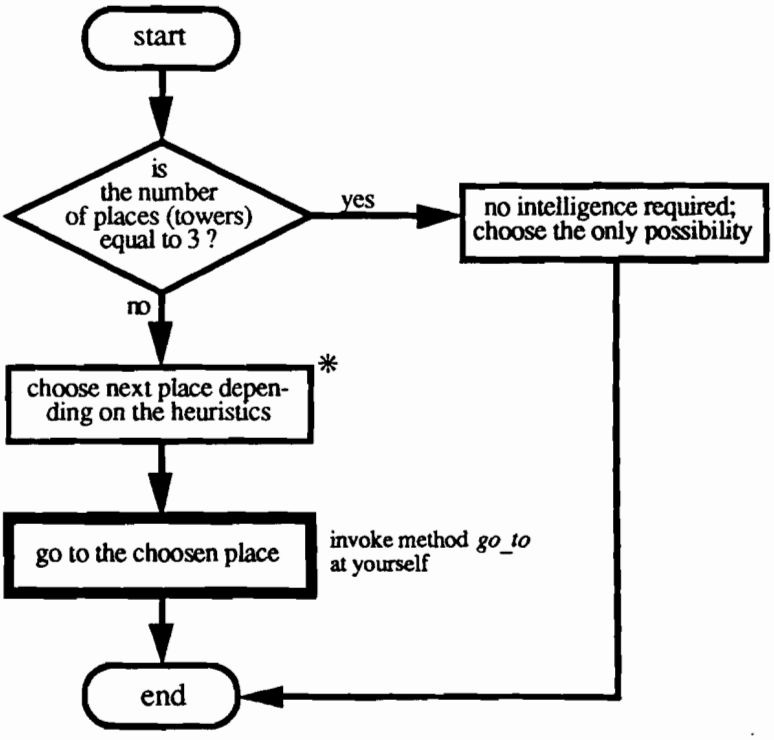

Figure 4.9 : Method *go_away_but_avoid*

*Eventually a call of the methods *update* and *display* (i.e. to perform a move) might happen. In this case the method *go_away_but_avoid* ends after the performed move.

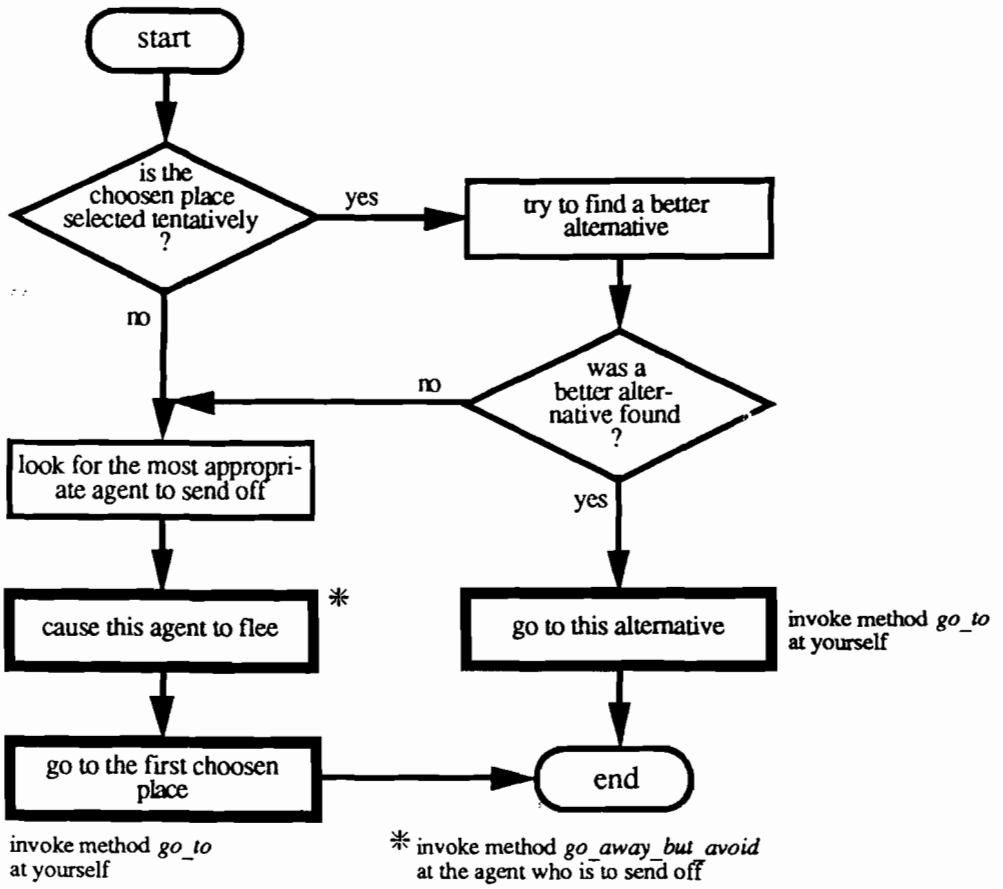

Figure 4.10 : Method *seek_best_place*

4.4.3. Heuristics

All heuristics described in this chapter manage the decision where to go, when an agent receives the message "Leave your place, but do not go to...". There is the opportunity to bring in some kind of intelligence, the agents normally have certain degrees of freedom to make their new choice.

The heuristics used in the sequential version TOHSEQ are structured hierarchically. The user selects one out of seven heuristics, which are identified by an associated integer (1,...,7). As a principle, heuristic i is founded on heuristic i–1, i.e. it has all the capabilities the former heuristic has. This implies, all tests and considerations which are provided by heuristic i–1 are also employed by heuristic i, even if not explicitly mentioned in the description below. Thus if we present the features of a heuristic, it is a presumption that all knowledge of the poorer heuristics is at hand. The "higher heuristics" should therefore be more sophisticated than the "lower" ones. Generally this should result in steadily improved benchmark results for increasing numbers of heuristics. The actual results are given in the next chapter. We will now describe the heuristics in some detail.

Heuristic 0 (in method *go_away_but_avoid*)[11].

As mentioned above, the user can choose a heuristic between 1 and 7. Heuristic 0, however, is switched on automatically if TOHSEQ is started merely with three places (towers). Then all sophisticated considerations are in vain, there is no free hand for decisions at all. Every heuristic would behave identically and thus all would show the same performance.

To save computation time, heuristic 0 is added. In the method *go_away_but_avoid* it directly calculates the only place which is left to go for each agent which is sent off. This reduces the decision where to go into a quick computation. The results must be equal to the recursive ToH algorithm which surely cannot be improved.

Heuristic 0 "do not use heuristics, if not necessary"
In a deterministic scenario (no alternatives), heuristics for decisions are unnecessary. Do not think deeper and take the only possibility left to go on.
If there are cheap exact algorithms, exploit them as much as possible.

[11] In the following we will annotate in brackets in which methods the heuristics operate.

Heuristic 1 (in method *go_away_but_avoid*).
Comprises the fundamental principle of all higher heuristics. For each agent and each decision situation the set $\mathcal{P}$ of all places (towers) is divided into disjunct subsets, the set $\mathcal{A}$ of "allowed places" and the set $\mathcal{F}$ of "forbidden places" with respect to the next move of each agent. $\mathcal{F}$ consists of at most two places; the next subgoal of the agent's "lower neighbour" just below itself, and the actual place of another agent, which causes the agent to leave.
From the set of allowed places one element is selected at random. To flee from the momentary place to the selected place becomes the new subgoal.

Heuristic 1 "the selected alternative must be allowed"
Partition the set of all alternatives into a set of allowed alternatives $\mathcal{A}$ and a set of forbidden alternatives $\mathcal{F}$, according to the actual constraints. Select one element out of $\mathcal{A}$ at random and make it your new subgoal.

Heuristic 2 (in method *go_away_but_avoid*).
Forces the willingness and straightness to reach the final goal and avoids needless stepping around. Before selecting any alternative for the next action the agent checks whether it already can reach its final goal within the next move. In this case no other alternatives have to be regarded any longer. Three conditions must be fulfilled :

- The goal place is an allowed place.
- The goal place is attainable.
- The agent is not blocked by other agents.

Heuristic 2 "preserve some self-interest"
Form the sets $\mathcal{A}$ and $\mathcal{F}$ as in heuristic 1. Check whether your final goal is in $\mathcal{A}$ and if you can reach it within your next action. In this case go directly to the goal place, otherwise proceed like in heuristic 1.

Heuristic 3 (in method *go_away_but_avoid*).
If the test in heuristic 2 does not return true (i.e. the goal is not attainable in one step), consider all allowed places where you could go ($\mathcal{G} \subset \mathcal{A}$). These are places which are either unoccupied or where an agent with a higher priority is situated on top of the stack. Select the place with the flattest stack out of $\mathcal{G}$. If there is more than one, take the first.
The basic idea is to cause as few blockings as possible. Again, this will be a foundation for the following heuristics.

Heuristic 3 "look for proper alternatives"
Build the set $\mathcal{G}$ of possible alternatives for the next action as a subset of $\mathcal{A}$. Find a subset $\mathcal{M}$ of $\mathcal{G}$ where each element promises minimal blockage for later actions (domain specific). Choose the first element of $\mathcal{M}$ as the next subgoal.

Heuristic 4 (in method *go_away_but_avoid*).
Is similar to heuristic 3 except the selection criterion. The next subgoal is selected by accident.

Heuristic 4
Proceed the same way as in heuristic 3 and build the set $\mathcal{M}$. Use a random selection function which guarantees equal likelihoods for all alternatives in order to determine the next subgoal.

Heuristic 5 (in method *go_away_but_avoid*).
The local knowledge of the agents shall be augmented in the way that more information about the wishes and intentions of the other agents will be at their disposal.
Actually an agent which is sent off will get the additional advice where other agents want to go after it has left. Thus the agent can try to obey other agents' wishes when making the decision where to go next. In detail this may sound as follows: "Leave your place now, but avoid to go to place Z, because I want to go there. Furthermore I heard that other agents want to go to the places $Y_1,\ldots,Y_n$. Try to stay away from these, too." The receiver of this message will do its best to keep the advice and selects the first place from set $\mathcal{M}$ (see heuristic 3) where no agent wants to go. If there is no such place, it uses the selection criterion of heuristic 3.
This proceeding can be regarded as the combination of constraints given by other agents, which are still blocked.

Heuristic 5 "combine constraints"
Try to take as much intentions of co-actors as possible into consideration. If you cannot go along with their wishes, i.e. each alternative in the set $\mathcal{M}$ (see heuristic 3) is objected by some agent, use the selection criterion of heuristic 3: Choose the first element of $\mathcal{M}$ as the next subgoal. Otherwise take the first detected alternative out of $\mathcal{M}$ which does not interfere with any agent's intention.

Heuristic 6 (in method *go_away_but_avoid*).
Likewise heuristic 5, but with a selection function analogous to heuristic 4.

Heuristic 6
Proceed like in heuristic 5. If the known intentions of the agents cannot be protected, choose an alternative in $\mathfrak{M}$ at random.

Heuristic 7 (in methods *go_to, go_away_but_avoid , seek_best_place*).
The agents decide where to go next often long before they really can act. All the agents lying on them and blocking them first have to go away. In the heuristics up to now the agents' decision was fixed. Once made, it will be carried out, no matter what happened in the meantime. But this time interval can be considerably long, the world may have changed remarkable. The decision made a long time ago is hardly up to date. Perhaps better alternatives have emerged.
Unlike in the prior heuristics where the agents stick tightly to their decisions, heuristic 7 makes reflections and corrections possible. On the basis of the situation in the moment when the agents really can act, they try to find a better alternative for the decision made. If they cannot find any, they will pursue the prior subgoals.
A few technical difficulties arise by realizing the idea. The heuristic not only concerns a decision at the time when the agent is sent off (in the method *go_away_but_avoid*), moreover it has influence on the method *go_to* which is involved in the moment the agents can act. Therefore also the selection function *seek_best_place* must be reviewed.
Some details: The next subgoal is chosen like in heuristic 5 but gets a mark "tentative". Additionally, the place where the agent shall go by no means is memorized. This is done in the method *go_away_but_avoid* . Later, when the agents really can move, only in the case the decided goal is occupied the agents behaviour changes with respect to heuristic 5. Otherwise the goal will normally be pursued. These tests are made in the method *go_to*. If the place is occupied, however, the agents try to find a better alternative, namely an allowed place where they can go (method *seek_best_place*). If there are more such places the one with the minimum number of places is chosen (see arguments from heuristic 3) and, if there is more than one, just the first one.

Heuristic 7 "reflect about your decision"
Select one alternative for the next action like in heuristic 5. At the moment you can really act (all obstacles are removed) make a test: Is the chosen alternative still appropriate ? If yes, go on as usual. Otherwise reflect your decision and try to find a better alternative. If you find one, pursue it. If not, proceed like in heuristic 5 (stick to your prior choice).

The next chapter will prove the capabilities of the heuristics. We performed some benchmark tests concerning the number of moves of the agents (corresponding to the number of nodes expanded in graph search!) and measuring the absolute run time.

4.4.4. Benchmarks

The sequential version TOHSEQ allows for the choice of seven heuristics with increasing power, as we have seen in the prior chapter. For heuristics with non-deterministic decisions [12] we give three test results. An upper and an lower bound for the number of actions performed by the agents until they all reached their final goal is surely interesting to rank the heuristics.

The maximum number cannot be determined. Like stupidity is (almost) infinite, a silly heuristic could produce an arbitrary number of moves. The minimum number of moves as an upper bound for the quality of a heuristic is much more interesting. It depends on the number of places in the scenario.

Let a be the number of agents in the scenario, p the number of places. There exist two extreme conditions:

① $p = 3$

Proposition

In a sequential ToH scenario with p=3 places, starting place ≠ goal place, and a agents, the minimal total number of moves is $2^a - 1$.

Proof: (based in the principle of the recursive ToH algorithm) By induction over a.

a=0, a=1 ok.

$a \rightarrow a+1$: Let 1 be the starting place, 2 the goal place. We proceed the following way :

1.) a agents move from place 1 to place 3: $2^a - 1$ moves (induction assumption).

2.) 1 agent moves from place 1 to place 2: 1 move.

3.) a agents move from place 3 to place 2: $2^a - 1$ moves (induction assumption).

Thus for $a+1$ agents we have $2^a - 1 + 1 + 2^a - 1 = 2^{a+1} - 1$ moves.

② $p \geq a+1$

Proposition

In a sequential ToH scenario with a agents and at least $a+1$ places the minimal total number of moves is $2a - 1$.

[12] These are the heuristics 1,2,4, and 6.

Proof: Let 1 be the starting place, 2 the goal place. The biggest agent must be free to go from 1 to 2.

1.) $a-1$ agents move away from 1 to free places between 3,...,p. (There are at least $a-1$ places for free choice): $a-1$ moves.

2.) 1 agent moves from place 1 to place 2: 1 move.

3.) $a-1$ agents move to place 1: $a-1$ moves.

Thus for a agents we have $a-1 + 1 + a-1 = 2a-1$ moves.

The proof for ② can be directly translated into a simple algorithm based on the principle of "first unstack each agent on places different from the goal place, then move the biggest agent to the goal and stack the others again on the goal".

Now we have manifested the spectrum of upper bounds for TOHSEQ. As a quality criterion for heuristics we demand the $2^a - 1$ result as mandatory for a scenario with $p=3$ places (towers) and the $2a+1$ result for a scenario with at least $a+1$ places.[13] Furthermore a good and stable heuristic must not produce worse results if the number of places is increased.

Thus the results of the tests should be in the interval 19 ($2a+1$),...,1023 ($2^a - 1$), where $a=10$ agents are in the scenario. The results are depicted in diagrams (Figures 4.11 - 4.15). On the left diagrams the total number of moves needed by each heuristic is shown. On the right diagrams corresponding time measures are given, for non-deterministic heuristics a minimum time and a maximum time. Each pair of diagrams represents the results for a certain number of places. The number of agents is always 10.

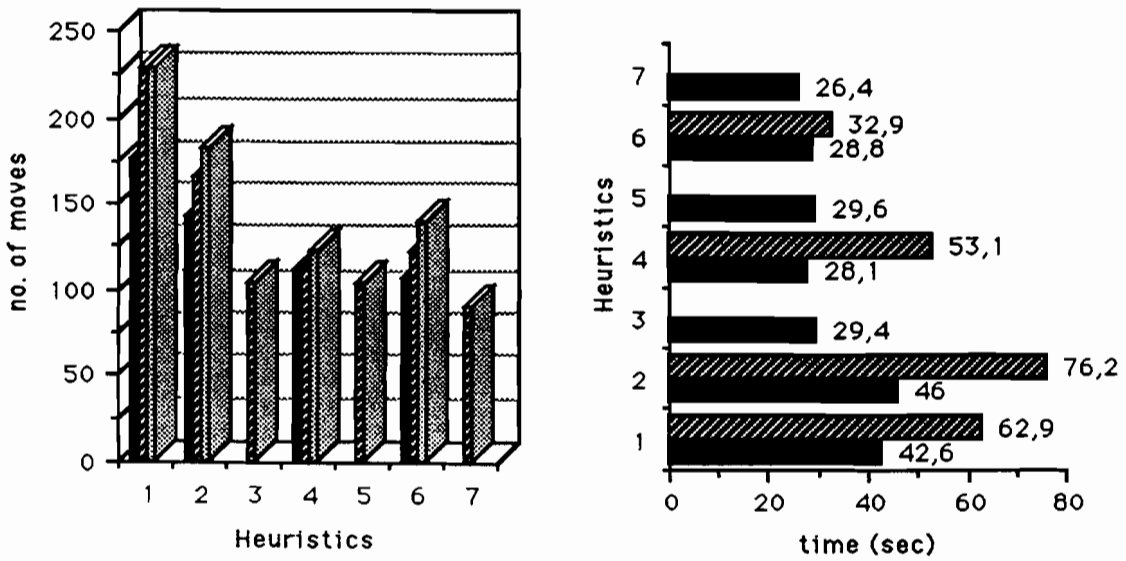

Figure 4.11 : Results for 10 agents and 4 places

[13]For a number of places between 4 and n see the considerations in the appendix of this chapter.

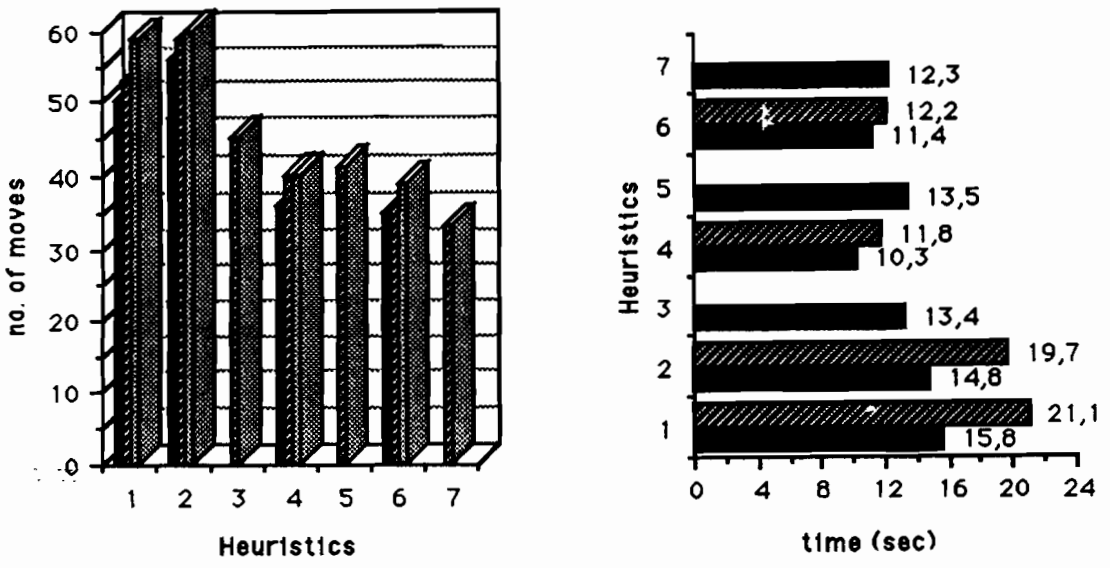

Figure 4.12 : Results for 10 agents and 6 places

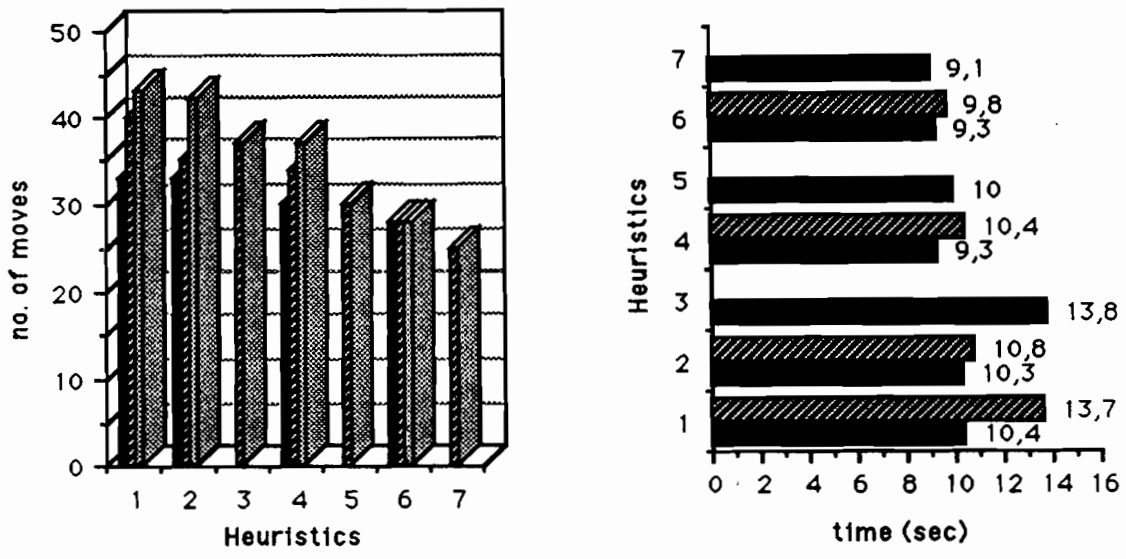

Figure 4.13 : Results for 10 agents and 8 places

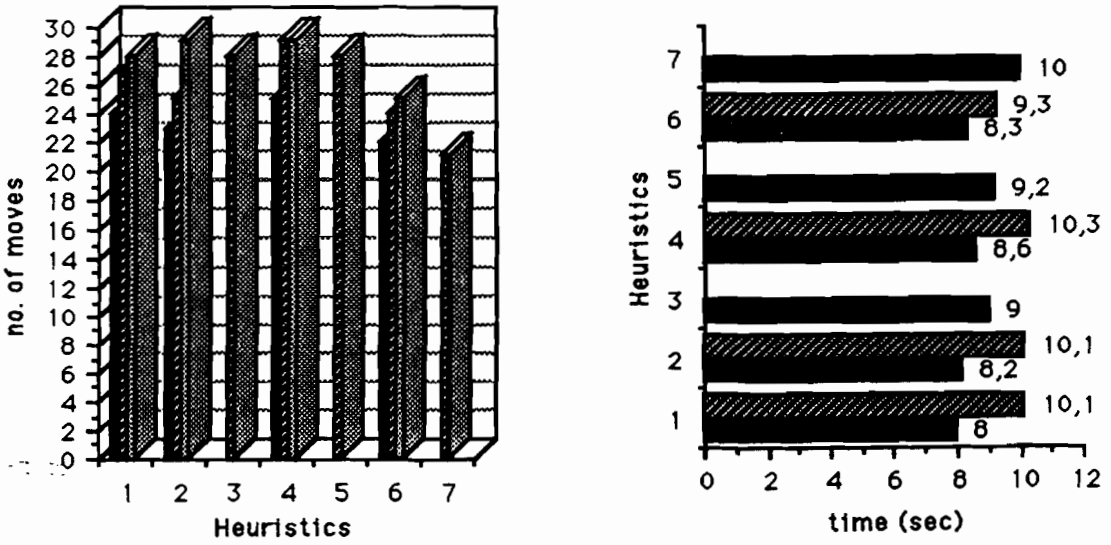

Figure 4.14 : Results for 10 agents and 10 places

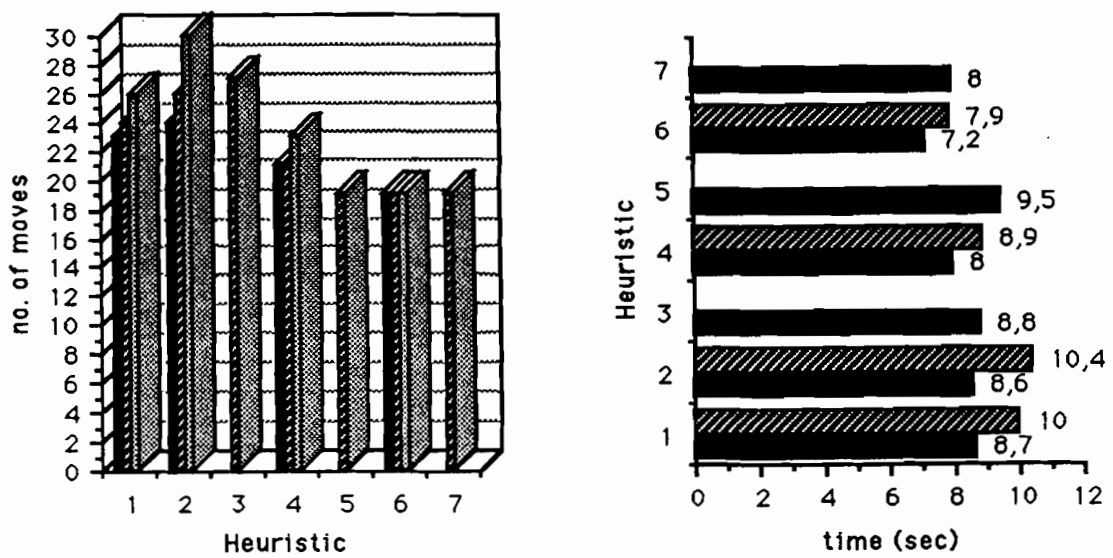

Figure 4.15 : Results for 10 agents and 11 places

Critique of the heuristics with respect to their test results

All heuristics produce optimal results for p=3 places (not given in diagrams). That is not surprising since the scenario is deterministic. Heuristic 0 prevents each chosen heuristic from unnecessary computations. The agent must produce the same result as the recursive ToH algorithm. For $p = a+1$ (i.e. 11) places, the heuristics 1 to 4 fail to satisfy the quality criterion mentioned above. Just the augmented knowledge about the agents' intentions (beginning with heuristic 5) guarantees the optimal solution. In general we found the intended improvement of the heuristics with refined considerations.

① only Heuristic 1:

- + + simple computation
- + for "many" places acceptable results
- – – in other cases catastrophic results
- – – wide variance for calls with identical parameters

② Heuristics 1 and 2:

- + – variation narrower than in ①
- – – in relation to ① and the results inadequate expensive

Reasons for the poor results: Heuristic 2 covers only a very rare condition. But this condition is a useful basis for the further heuristics to avoid very stupid moves.

③ Heuristics 1 through 3:

- + + for p=4 places remarkable good results
- + – balanced ratio between effort and results
- – the more places in the scenario the worse become the results, relatively
- – – for $p=a+1$ (11) places an unacceptable bad result

④ Heuristics 1 through 4:

- + + only a bit more expensive than ③ but significant better results
- + narrow variance
- – not optimal for $p=a+1$ (11) places

⑤ Heuristics 1 through 5:

- + for $p=a+1$ (11) places optimal results, otherwise not really better than ④
- + – solid results, but increasing costs

⑥ Heuristics 1 through 6:

- + almost each result is better than ⑤, but not much more expensive
- + very narrow variation
- – – inexplicable break-in for p=4 places

⑦ Heuristics 1 through 7 (all heuristics) :

- + + the very best results
- + + seems to be near to the optimal results in a certain range of p, cf. appendix
- – the price to pay for the quality is the much more complicated control flow

Appendix

We present a sequential ToH algorithm that provides an upper bound for the number of moves in a restricted range of the number of places, p. This range shall be called "linear range".

Proposition

Let a be the number of agents and p the number of places in a sequential ToH scenario. If a lies in the range between p and $2p-3$ (linear range), the total number of moves is not worse than $4a - 2p + 1$.

Proof: An algorithm with just this complexity[14] is introduced.

Algorithm for the linear range

Preconditions

- a : number of agents, p : number of places, $p \leq a \leq 2p - 3$
- the agents are numbered with running integers 1,... where agent 1 is the biggest.
- the order of the moves in lists in the algorithm is mandatory !

Procedure

① Unstack $p-1$ agents arbitrarily onto the free places. Do not put agent $p-1$ on the goal place (agent $p-1$ will be moved, because at most $2p - 3 - (p-1) = p - 2$ agents are still at the starting place).
☞ $p - 1$ moves

② Stack $a - p + 1$ agents from the ones which were unstacked in step ①.
Agent p onto agent $p-1$, agent $p+1$ onto agent p,..., agent a onto agent $a-1$
☞ $a - p + 1$ moves

③ There are $a - p + 1$ agents not yet moved and still on the starting place (follows from ①: $a - (p-1) = a - p + 1$). Furthermore $a - p + 1$ places are free (follows from ②).
Unstack $a - p$ agents ($a - p + 1$,...,2) from the start place onto arbitrary places, but not onto the goal place.
☞ $a - p$ moves

④ Move agent 1 from the starting place to the goal place.
☞ 1 move

⑤ Stack the agents 2,...,$a - p + 1$ onto the goal place.
☞ $a - p$ moves

[14]The number of moves shall be the only factor for complexity.

⑥ Stack the agents $a - p + 2, a - p + 3, \ldots, p - 2$ onto the goal place (These were unstacked in step ① and not stacked again in step ②).

☞ $2p - a - 3$ moves

⑦ There are still $a-p + 2$ agents stacked on one place, namely $p-1, \ldots, a$ (follows from ②). Unstack $a-p + 1$ agents on arbitrary places (there are $p-2$ free places and at most $(2p - 3) - p + 1 = p - 2$ agents to unstack).

☞ $a - p + 1$ moves

⑧ Stack the agents $p-1, p, \ldots, a-1, a$ onto the goal stack. READY.

☞ $a - p + 2$ moves

The total sum of moves is:

$p-1 + a-p+1 + a-p + 1 + a-p + 2p-a-3 + a-p+1 + a-p+2 = 4a - 2p + 1$ moves.

4.5. The Parallel Version : TOHPAR

4.5.1. General Phenomena

Nearly everything changes in the parallel realization of the ToH scenario. The foundations and basic ideas, however, remain the same (cf. chapters 4.1., 4.2.).
The agents pursue their goals in parallel, i.e. they decide about their next subgoals as well as execute the planned actions contemporary. In such a context conflicts inherently occur. At each instant of time, an agent has to find the right next action; it must decide either to attempt its own goal, or to act in favour of someone else. On the other hand, two or more agents may try to reach the same or overlapping goals. In this case an agreement must be found in order to avoid deadlock situations. Later we will call the first mentioned class of conflicts *internal conflicts* and the latter class *external conflicts*. Heuristics are widely used to resolve both types of conflicts.

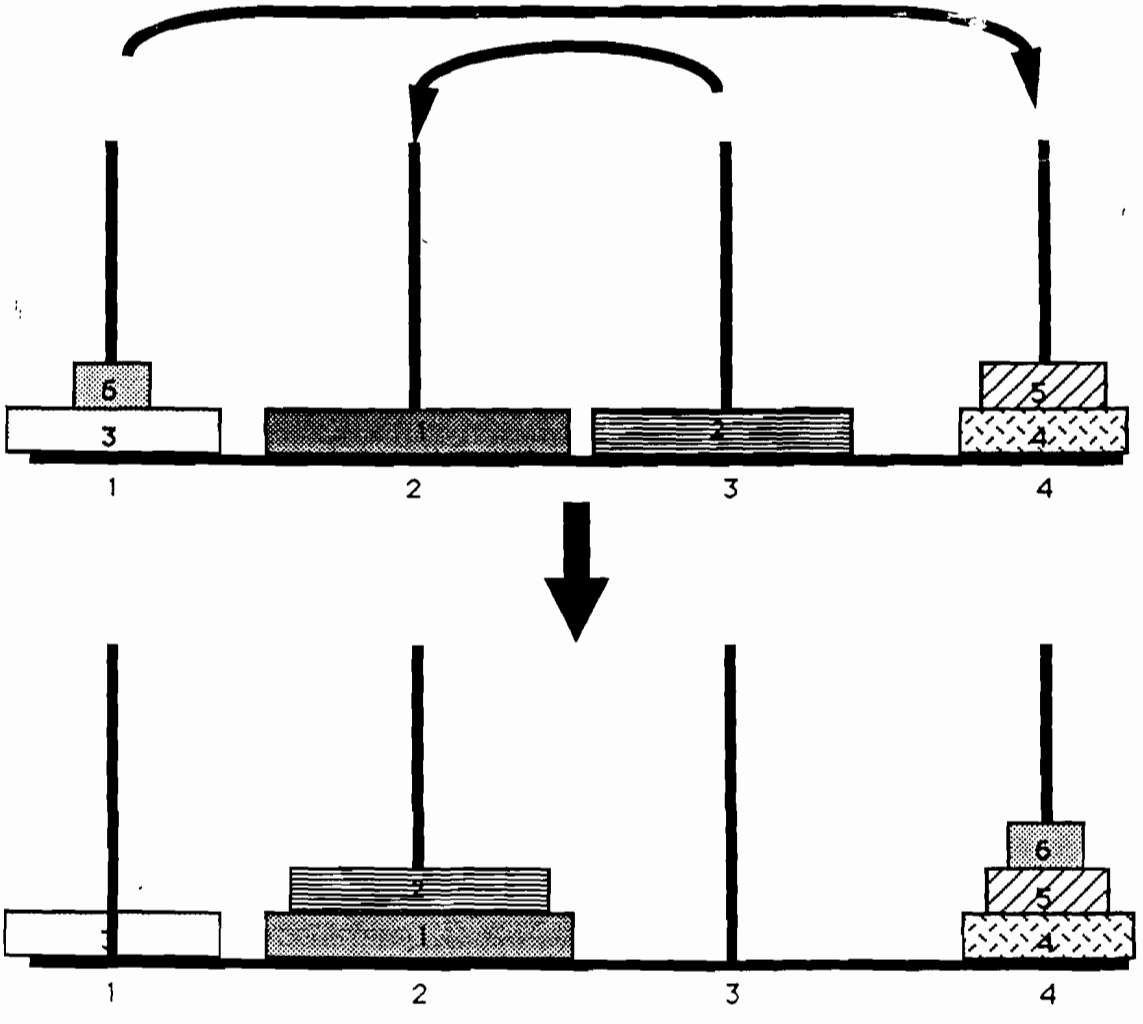

Figure 4.16 : Parallel acting may speed up the process of problem solving

In a parallel scenario certain uncomfortable phenomena may arise which are naturally inhibited by sequentiality. These are, for instance, synchronization needs, resource conflicts, priority clashes, unpredictable influences of the parallel acting for the individual subplans and subgoals of the agents, and the need for negotiation.
The reward received for the attainment of parallelism is twofold. First, the simulation becomes more natural. We can really speak of eco-problem solving and transfer the employed techniques to real world applications. Second, parallelism accommodates the chances of speeding up the problem solving process (cf. Figure 4.16). But these chances must be exploited, careless decisions and heuristics may also cause a degradation with respect to the sequential proceeding.
In the following we will start analogously to the sequential version with presenting the architecture and a chapter about the heuristics employed. Additionally, a theoretical model of the agents behaviour in form of a finite automaton is presented and, after all, open problems are discussed.

4.5.2. Architecture

The agents themselves may still be identified with mathematical tuples as in chapter 4.3. These tuples must be augmented with components for storing decisions or intentions in order to detect and resolve possible conflicts. Furthermore information about the inner status must be at hand, i.e. an agent has to signal itself or others whether it is just waiting, in a conflict situation, perhaps able to act, or ordered to leave its place. In the sequel the agents shall also be regarded as knowledge bases with attached procedural knowledge.

4.5.2.1. Abstract View

The abstract view of TOHPAR comprises the *behaviour* of the agents which is modelled in the control flow of the concrete realization. The central concept still comprises the urge of the agents to reach their goal and thus the demand to cause those agents to step aside, which hinder them: "Leave your place, but do not go to the place where you would block me again."
But now the agents receive their goals all at the same time. All agents have their final goal in mind, not only just one, all agents must permanently rank their personal interest of pursuing their very own goal and the interest of others that they go away.

We distinguish between internal and external conflicts.
Internal conflicts arise when the decision about the next step is to make, obeying and weighting all known intentions of other agents. Decision rules supply means for solving these conflicts.
External conflicts, however, are caused by other agents who e.g. have selected the same goal (remember: all decisions run in parallel) or have just forbidden the selected goal. These conflicts are resolved by negotiation. The agents exchange public information via a blackboard where also suggestions for solutions are written upon. If no solution for all involved agents can be found, some of them must retract their wish to act. Decision rules and the negotiation protocol heavily depend on good heuristics.
The knowledge of the agents can be distinguished analogously.
The individual knowledge is written on each agent's private knowledge base, which contains, besides information about current position and upper neighbour etc. each agent's agenda. The agenda serves as a kind of mailbox, or better, as a working desk. New messages permanently arrive like new notes fluttering through an open window. These notes must be evaluated, ordered, or sometimes thrown into the waste paper basket. They may comprise information about the agent's personal goal, about a subgoal which is next to encounter, alternatives for the next subgoal, and orders from the other agents to flee. At least the personal goal must be steadily kept on the agenda, all other entries are permanently in exchange. Finally, when the agent has reached its desired goal, it deletes the whole agenda which indicates the new status "ready".
Public knowledge is information concerning all agents. For the sake of simplicity, efficiency and consistency it is not sent to each individual agent, but "broadcasted" to all other agents via a blackboard. All agents must permanently have a look onto the blackboard to check whether they still conform with other agents' intentions. Each agent may perform only two operations with blackboard entries, assert and retract, and moreover, these operations are only allowed on entries they wrote themselves. Agents write messages on the blackboard to inform the others about their intentions. Two message types are promoted: Wish-messages : "I tell other agents what is my wish for my own next step" and help-messages : "I wonder, if anyone can give me support under the position I want to go next".
The whole scenario and especially the subpart where the agents really act is mediated and moderated by a special agent, which does not belong to the class of "normal" agents. The mediator is an instance of the class INFO-AGENT and shall be called "Big-Brother". Big-Brother subsumes all global knowledge about the parallel hanoi world in its knowledge base. If the agents want to know any global fact about the momentary situation which is not in their individual knowledge base, they may ask Big-Brother for information.

In Figure 4.17 a very coarse presentation of the overall TOHPAR system from the abstract point of view is given (in each box of the flow diagram the agents may act in parallel).

This diagram is the foundation of the system. It reveals the abstract architecture (abstract system design) and will be refined and made more concrete in the following chapters. Also a sketch of the scheduling lies herein.

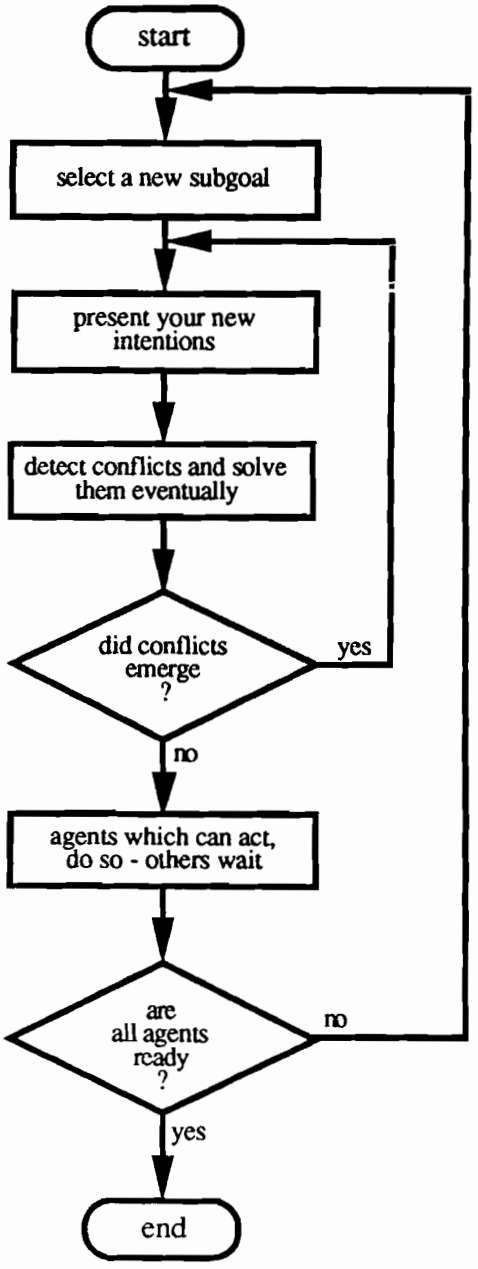

Figure 4.17 : Abstract system design of TOHPAR

The abstract world (i.e. the parallel ToH scenario, Figure 4.18) which is only *modelled* in TOHPAR must be depicted in quite another manner than the system design. Actually, the blackboard is also integrated in Big-Brother's knowledge base. But this does not matter.

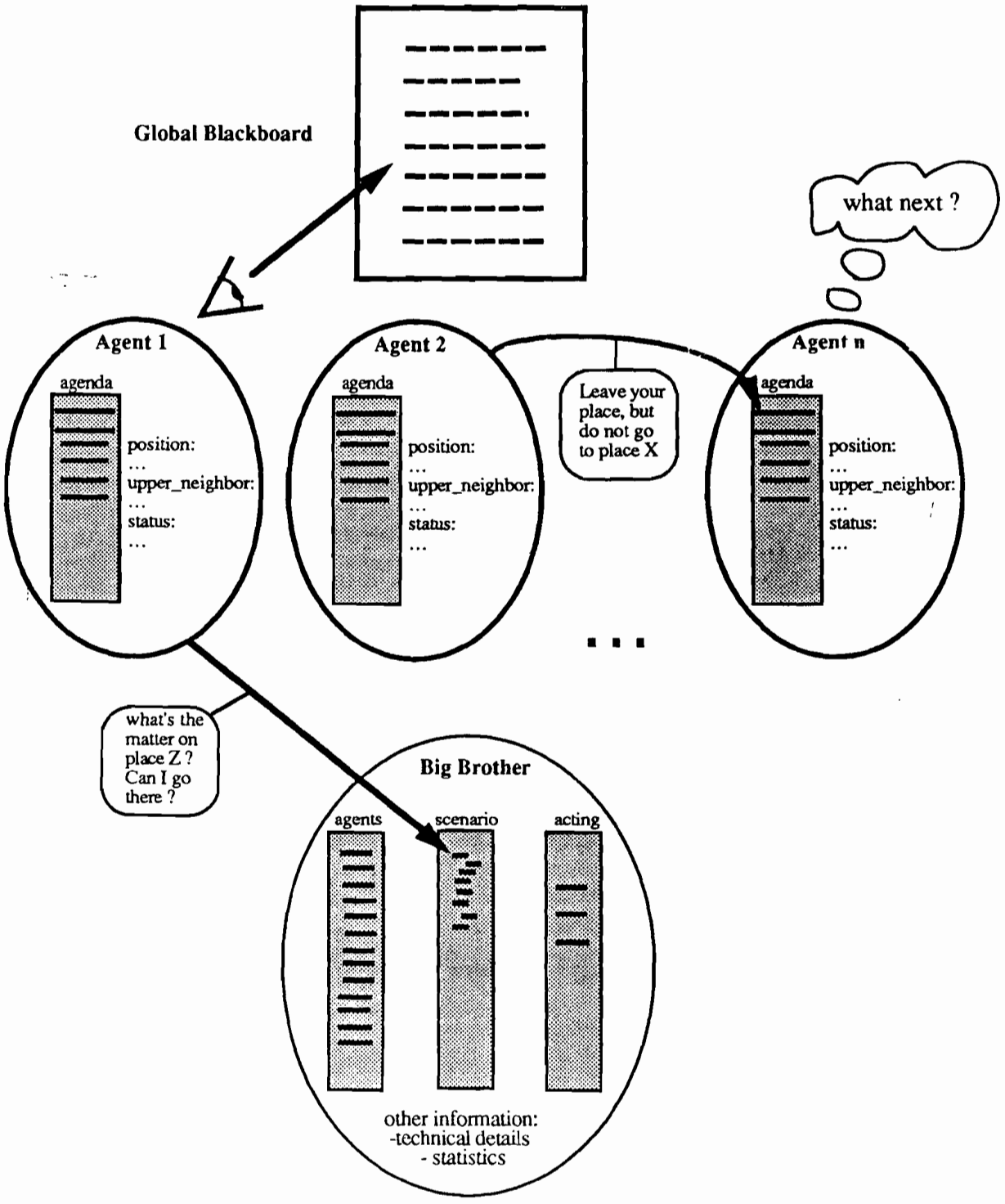

Figure 4.18 : The abstract world of parallel acting agents in a ToH scenario

4.5.2.2. Concrete Realization

The next two subchapters present the general system design starting from the top-level function *hanoi-parallel* and a detailed discussion of the "modules" of the system, where the aspects decision planning, conflict resolution, and negotiation will be emphasized.

General System Design

The top-level function *hanoi-parallel* calls methods of the agents or methods of the information agent. It provides the core for the simulation of the parallel scenario, and, on the other hand reveals the central ideas of tackling the problems which arise when information can be exchanged in parallel. We will give the flow-diagram in Figure 4.19, which is just a refinement of the abstract system design in the prior chapter. The names of the methods are almost self-explaining, at least they evoke a feeling what is desired. The methods represent a kind of modules of the system. They will be described in some detail below.
Heuristics are employed in all methods except the first. Strictly spoken, the whole approach of modelling the parallel ToH scenario is heuristical and does not claim to be the only one.

As we now have got a first insight in the framework of TOHPAR we present the types of agents in the scenario. The normal agents, which really act in the scenario belong to the type AGENT of tuples. The other type is INFO-AGENT with only one instance, Big-Brother, the mediator and source of information for the other agents.
Well, how are the tuples defined ?

tuple AGENT: (name, place, position, blocker, agenda, act-flag, conflict-flag, wait-flag, go_away-flag)

name	: unique identification	
place	: actual place	
position	: exact position at place	
blocker	: actual agent which is located on the instance itself	
agenda	: list of next goals, alternatives, orders to flee,...	
act-flag	: indicates whether	agent thinks that it can act
conflict-flag	: ...	agent was in a conflict situation
wait-flag	: ...	agent waits for the other agents
go_away-flag	: ...	agent is sent off by another agent

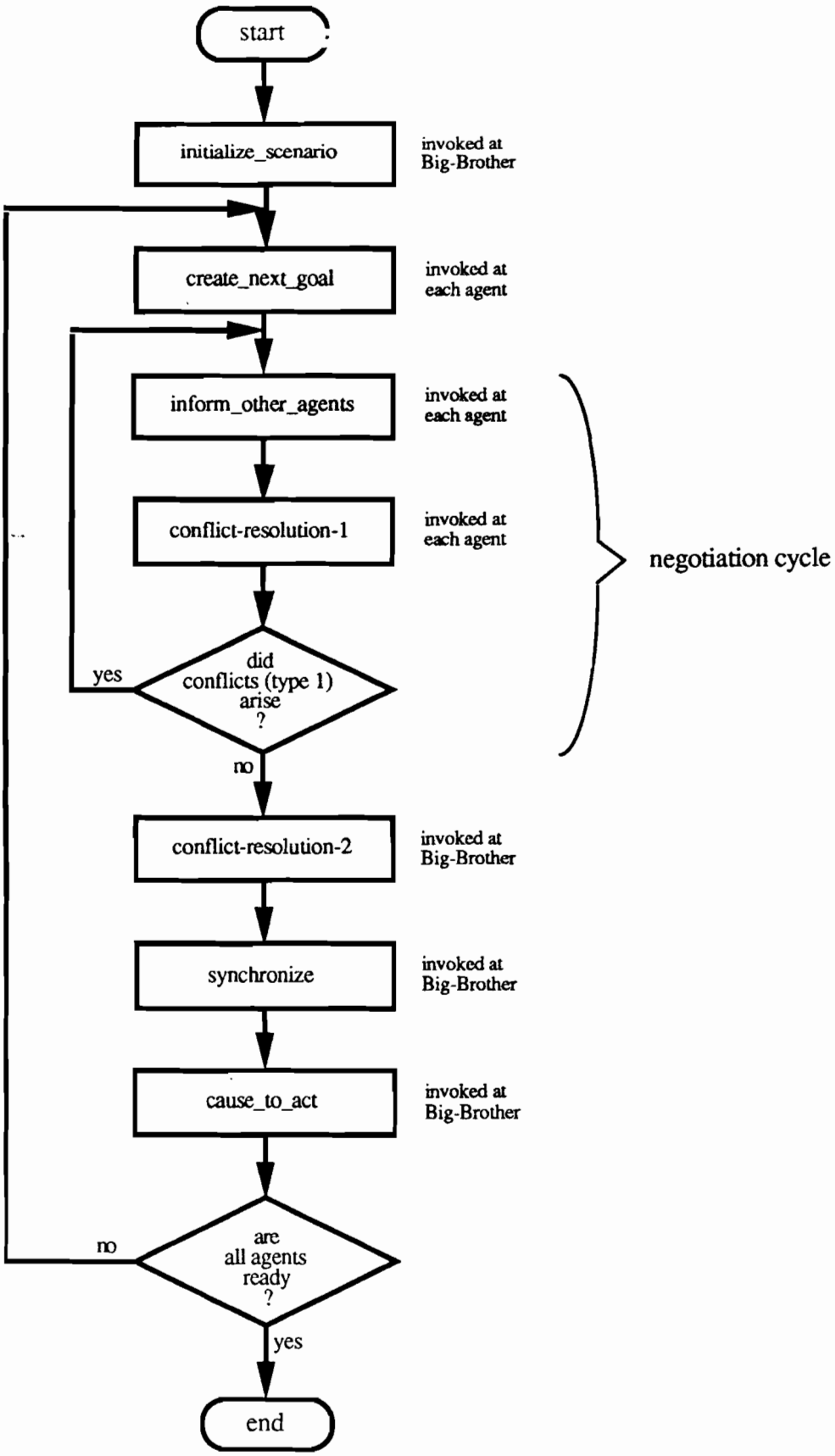

Figure 4.19 : The top level function *hanoi-parallel* and the called methods

tuple INFO-AGENT:	(blackboard, agentarray, placearray, actarray)
blackboard	: actual agent which is located on the instance itself
agentarray	: array of all agents in the scenario
placearray	: actual configuration in the ToH world
actarray	: agents, which actually want to act

Thus the instances of the tuples AGENT and INFO-AGENT comprise all of the declarative knowledge of TOHPAR. The control and data flow (i.e. the procedural knowledge how to tackle certain situations) lies inherent in the order of called methods each with a subtree of called auxiliary functions (see "The Modules of the System").

It is of central interest that all the agents can try to get more information by asking Big-Brother. Up to now no authentication and access control is built in. Four slots of Big-Brother are of predominant importance:

- blackboard : the only structure for the agents to exchange public information
- agentarray : makes access to all other agents feasible
- placearray : an inner model of the outer real configuration of the ToH scenario.
- actarray : information about agents willing to act will be gathered for later synchronization

For the tuple AGENT some attributes (slots) from the sequential version TOHSEQ have been adopted. New slots are the entries for the agenda and four flags which reveal the actual status the agent is in. *agenda* is merely a list of messages managed by certain priority and generation rules (cf. "Creating New Goals"). The three status-flags indicate prior, contemporary, and future urges for the agent:

act-flag

Shows if the agent still believes it can act, but gives no guarantee whether it is really so. The act-flag must be reset, when the agent retracts his wish and finds no alternative.

conflict-flag

Has the agent been involved in conflicts[15] ? If it has, it must inform the others about its possibly changed new decisions.

wait-flag

The agent shows that it does not contribute in negotiations any longer. There are two possibilities to do this. First, the agent may have realized that it can act by no means. Either it is blocked by an "upper neighbour" or the goal place is not attainable, even if the agent there on the top would flee. Second, the agent has elected a (sub)goal where it does

[15] In conflicts of "type 1" as we will see later.

not interfere with others. It waits for the others to finish their negotiation and to start the joint acting. If one of the agents in negotiations selects a new subgoal which then interferes with the waiting agent, however, the wait-flag must be reset again, the former waiting agent is torn back to negotiations.

go_away-flag

It has the duty to memorize that the agent once was send off. It ensures that such orders will not be forgotten in later considerations and forces the concerned agents to keep busy.

The messages which are written on any agenda or the blackboard, respectively, must have a predefined format that all addressees can interpret them. Two remarks hold for all types of messages: *status* can take a value REQUEST(ED) or RETRACT(ED), depending on whether the message is new or unrivalized, or whether the message had to be withdrawn caused by negotiations. *position* often gets the value NIL, mostly only the place is interesting, the exact position on that place is not important.

The messages for the blackboard are:

wish-messages

Agents which believe that they can act, inform the others about their plans. The next desired subgoal is brought into discussion in this manner.

Format: `(wish <place> <pos> <name> <status>)`, which means:

"I wish to go to place <place> (exact position <pos>). My name is <name>. The message is <status>."

help-messages

Agents which are not blocked, but with a still unattainable subgoal[16] may send help-requests to the society of agents. Later, agents selecting alternatives will favor such which satisfy help requests of others.

Format: `(help` <support place> <support pos[17]> <name> <status[18]>`)`, which means:

"I need help. Please support me at place <support place> (exact position <support pos>). My name is <name>. The message is <status>."

Messages written on the agendas of individual agents:

goal-messages

Information about the final goal given to the agents at the start of the scenario. This message is the only one which must permanently reside at the agenda until its intention is fulfilled.

Format: `(goal` <goal place> <goal position>`)`, which means:

"My final goal is place <goal place> with exact position <goal position>."

[16] In the ToH scenario: the existing "tower" is not yet high enough.
[17] One below the position really attempted by the sender.
[18] The status-flag is not yet employed for help-messages. It is permanently set to "REQUEST".

leave-messages

Implementation of the central idea "go away, but avoid..." as mentioned above several times. This order is sent from the agent which is direct below the addressee or from an agent which wants to go just there where the addressee is situated.

Format: `(leave <place> <sender> <status>)`, which means:

"Go away, but avoid <place>. My name is <sender>. The message is <status>."

messages about the next goal

This is rather an information than a message because an agent writes this entry on its own agenda. It is the conclusion of the application of a set of rules to the entries of the agenda and the considerations about the actual situation. The decision what to do next has the highest priority on the so modified agenda.

Format: `(next_goal <place> <position>)`, which means:

"My next subgoal is place <place> with exact position <position>."

Messages about alternatives

Also this is an important information for situations where agents must reflect about their decisions and try to find new (better) issues. The list of alternatives contains those places which are not forbidden for the agent and still potential candidates for the next move. It is by no means ensured that the agents can really go there in the next move.

Format: `(alternatives <list of places>)`, which means:

"My actual alternatives are: <list of places>."

Now after a first insight into the structure and principles of the system, the principal way of discourse of the agents shall be enlightened. But before doing that, we introduce a few convenient definitions for making the further aspects more precise and shorten the terminology .

Definition

An agent *believes that it can act* iff (i) it is not blocked and (ii) either the place of the next goal is attainable or would be attainable if another agent there would flee at the same time.

Definition

Conflicts are of *type 1* iff they are caused by "leave-messages", which collide with own intentions (i.e. if new forbidden goals are in the list of alternatives or especially the next desired goal is suddenly forbidden).

Definition

Conflicts are of *type 2* iff they are caused by incompatible wishes of other agents with respect to the own intentions.

Both types of conflicts are external conflicts in the terminology mentioned above.

Which strategy do the agents pursue ? I will sketch it here in natural language. The more technical aspects can be found in the chapter "The Modules of the System".

Proceeding of the agents

① (the information agent)

Big-Brother initializes the scenario, creates the agents, fills the slots properly and hands over the orders where to go.

② (internal evaluation of all agents)

The agents modify their agendas and create new subgoals.

③ (external message broadcasting)

The agents present their intentions or post help-requests. Only the agents still potentially capable to act can do the first, the others resign and wait until the acting is finished.

④ (internal evaluation of the agents which believe they can act)

The agents which believe they can act, try to detect conflicts of type 1. If there is one, they set the conflict-flag and go back to ③. Other agents wait until all conflicts are resolved.

⑤ (the information agent)

Big-Brother synchronizes and mediates the acting of the agents left. The maximum number of parallel moves is performed. Acting agents delete their agenda except the final goal. After each acting cycle the blackboard is completely cleared.

⑥

If all agents are now on their final goal, the scenario will stop. Otherwise step ② is performed again.

The genuine negotiation cycle lies hidden between the steps ③ and ④. It shall be pointed out further in the following.

Negotiation cycle

The negotiation has the duty to solve conflicts of type 1. Thus when we say "conflicts" in this paragraph, we always refer to conflicts of type 1.

The negotiation cycle starts with publishing the wishes of those agents which believe they can act. This provides a basis for the agreement. It is important that the other agents, which by no means can act, do not take part in negotiations. They are not concerned, so they remain idle until the next parallel acting is performed.

Then each negotiating agent tests whether conflicts occur with respect to its individually planned next action. If there are still alternatives left, an agent involved in a conflict will select one and present it as a new suggestion. If no alternative remains and the conflicts are not resolved, the agent says "good bye" and leaves the negotiation cycle.

For each agent in negotiation the cycle of retracting and making new suggestions lasts until it either must cancel the negotiations because of a lack of alternatives or it is not in conflicts any longer. The whole cycle ends when all conflicts are solved.
One more issue is interesting: Agents, which solved their conflicts by detecting a suitable alternative can eventually be torn back into negotiations by others whose new decisions cause conflicts again.

At last few annotations to the blackboard architecture. The main advantages of a blackboard as a medium for exchanging public information between all agents are:

- Efficiency: Redundant copying and distributing messages to all agents is expensive concerning both temporal and storage aspects.
- Consistency: Once written on the blackboard, all agents have access to the same (and newest) information. Complicate validating and updating procedures are unnecessary.

The principle of broadcasting can be adequately realized with a blackboard architecture.

The Modules of The System

The methods called from the top level function *hanoi-parallel* shall be called modules. We will present these modules in detail here, laying our emphasis on the conditions which hold before and after a method (or a subfunction) is performed. For each method the call structure is given, but not all called functions are considered particularly further.[19]

Initialization

Method: `initialize_scenario` attached to type INFO_AGENT

Parameters: number of agents, number of places, starting place, goal place

Call structure:
```
initialize_scenario
        initialize_agents
        initialize_window
```

Preconditions: none [20]

Postconditions:

- The output window is correctly initialized.
- The configuration of all agents on their start position is visible.
- Instances of the class AGENT are created with properly filled slots.

[19]For a thorough understanding of the implementation it is recommended to consult at least partly the listing of the programs. The descriptions provide a more abstract overview about the "what" and "why", the listing will evolve the "how".
[20]A formal precondition is "true" or T.

- The message about the final goal is the only entry on each agent's agenda.
- The slots of Big-Brother are updated with actual parameters. The blackboard is empty, especially.

Description of the method:

The dynamic window parameters (see Appendix (8.)) are computed. Big-Brother's information is updated. New agents are created and their knowledge bases are filled with information about both start and goal position. The output window is created and the start configuration is displayed: agents, places (sticks), the headline.

Creating New Goals

Method: `create_next_goal` attached to type AGENT

Parameters: none

Call structure:

```
create_next_goal
        rule-1
        rule-2
        rule-3
        rule-4
        modify_agenda_of
                plan_*leave*
        make_new_decision
                plan_*alternatives*
                        find_heuristically_best_alternative
                                select_heuristically_best_alternative
                                        update_agents
                plan_*next_goal*
                plan_*goal*
```

Preconditions:

- Blackboard is empty
- State of agenda[21] : ([(goal x y)] [(leave...)]* [(next_goal...)] [(alternatives (...))]), arbitrary order of entries
- The current place is not in the list of alternatives, nor the prior chosen next goal
- The list of alternatives is not NIL

Postconditions:

- State of agenda : ([(next_goal a b)] [(goal x y)]) or ([(next_goal a b)] [(goal x y)] [(alternatives list)])
- not both a=x and b=y hold
- a is no element of list
- the order of goals at the agenda is mandatory

[21] The meta-notation is like follows: [..] means 0 or 1 times; [..]* means 0 or more times; [..]^n means exactly n times and | separates alternatives to choose.

Description of the method:

The method can be divided into three parts. First, four *rules* are applied to the agenda to eliminate redundancies and inconsistencies on the agenda. These rules are based on heuristics. The effect is a generalization in such a way that all decisions about next goals are retracted. Each element in the set of alternatives has potentially the identical likelihood to become a candidate for the next choice.

Second the leave-entries are investigated and evaluated. This is the duty of the function *modify_agenda_of*.

At last the decision must be performed. The function *make_new_decision* copes with that.

Function: `rule-1` of method create_next_goal

Parameter: agenda

Value: modified agenda

Preconditions: see method *create_next_goal*

Postcondition: no pairs of leave-orders with identical forbidden place exist on the agenda

Description of the function:

Performs the application of the rule

(leave x agent-i ...), (leave x agent-j ...), priority (agent-i) > priority (agent-j) →

(leave x agent-i ...) to the agenda.

Heuristic: The higher the priority of an agent, the higher is the priority of its leave-orders.

Function: `rule-2` of method create_next_goal

Parameter: agenda

Value: modified agenda

Precondition: rule-1 was applied

Postcondition: no forbidden place is a desired next_goal

Description of the function:

Performs the application of the rule

(leave x agent-i ...), (next_goal x y) → (leave x agent-i ...) to the agenda

Heuristic: A forbidden subgoal must not be selected for the next move

Function: `rule-3` of method create_next_goal

Parameter: agenda

Value: modified agenda

Precondition: rule-2 was applied

Postconditions:

- No forbidden place is an element of the list of alternatives
- No list of alternatives is an empty list

Description of the function:

Performs the following rules in the denoted order:

1.) (leave x a ...), (alternatives (x)) → (leave x a ...)

2.) (leave x a ...), (alternatives list) → (leave x a ...), (alternatives list–{x})

Heuristic: A forbidden place cannot be an alternative for the next move

Function: `rule-4` of method create_next_goal

Parameter: agenda

Value: modified agenda

Precondition: rule-3 was applied

Postconditions:

- no entry (next_goal...) will be on the agenda, if there are alternatives
- State of agenda : ([(goal x y)] [(alternatives list) | (next_goal a b)] [(leave ...)]*)
- information on agenda is consistent, all preconditions of rule-1,...,rule-4 hold

Description of the function:

Performs the following rules in the denoted order:

1.) (next_goal x ...), (alternatives list), x element of list → (alternatives list)

2.) (next_goal x...), (alternatives list), x not element of list → (alternatives list ∪ {x})

Heuristic: If you have other alternatives, reflect about your former decision. Therefore reduce your prior chosen next goal to a normal alternative

Remarks to `rule-1,...rule-4`

The functions do not modify the agendas destructively. They are all based on certain heuristics and hence disputable. Each rule fires as long as its precondition can be satisfied. The order of the application must not be changed, otherwise inconsistencies may occur.

Example:

Let an agenda consist of entries (leave x a s), (alternatives (x y)), (next_goal x z). The correct result returned after application of the four rules should be: ((leave x a s), (alternatives (y))) which means: I have the order to leave my place, but I must not go to place x. The only alternative is place y.

But an application of rule-3 first and the rule-4 would return ((leave x a s), (alternatives (x y))). The forbidden goal, however, cannot be a permitted alternative. rule-3 has to be applied a second time. To avoid these inconsistencies, it is essential that first the powerful heuristic of rule-2 is employed to remove former decisions which cannot be uphold.

Function: `modify_agenda_of` of method create_next_goal
Parameter: agent (actual parameter SELF)
Value: only side-effects are of interest
Precondition: rule-1,...,rule-4 applied
Postconditions:
- No leave-entries on the agenda
- State of agenda : ([(goal x y)] [(alternatives list) | (next_goal a b)])

Description of the function:
modify_agenda_of evaluates the leave-entries of the agenda. These entries are consistent with respect to possible next-goals or alternative entries, ensured by rule-1,...rule-4. So if there are other entries than *(goal ...)* or *(leave ...)*, the leave-entries may be deleted. Their intentional restrictions are obeyed.
In the other case, a new entry *(next_goal ...)* must be generated. There will be no problem, if some place where the agents could flee is not forbidden. These places are collected in the list of alternatives.
A harder task is a restriction for all places. The agent cannot obey all orders, it must flee from its place and therefore violate at least one leave-order. An agent uses the heuristic function *plan_*leave** to decide whose leave-order is to be ignored.

Function: `make_new_decision` of method create_next_goal
Parameters: agent (actual parameter SELF)
Value: only side-effects are of interest
Preconditions: modify_agenda_of applied to agent
Postconditions: see method create_next_goal
Description of the function:
After modifying and generalizing the entries on the agenda, the new decision is to produce. An agent has the highest degree of freedom in the case it has a list of alternatives as a basis for its free choice. The functions *plan_*alternatives** covers this case.
If there is an entry *(next_goal...)* on the agenda, the agent will employ the function *plan_*next_goal**, and in the case that *(goal...)* is the only entry on the agenda, the function *plan_*goal** is invoked.

Function: `plan_*leave*` of function modify_agenda_of
Parameters: agent (actual parameter SELF)
Value: only side-effects are of interest
Preconditions:
- rule-1,...,rule-4 were applied
- all potential places to flee are forbidden

- no *(alternatives ...)* and no *(next_goal ...)* entry is on the agenda

Postconditions:

A new entry (next_goal x NIL) is generated and written on the agenda, where x is a place which is actually forbidden, but the order is disobeyed.

Description of the function: Proceeding:

1. Build the set $\mathfrak{G}$ of "good places"; these are places where the agent could go (they are not occupied by an agent with lower priority). If $\mathfrak{G}$ is empty, nothing can be done for the moment. The agent becomes idle until the next cycle of the top level function.
2. Find the agent with the least priority of all on the top of the places in $\mathfrak{G}$. This agent has a priority just higher than the one of SELF.
3. Create the entry (next_goal x NIL) and write it on the agenda. x is the place of that agent found in 2.

Heuristic: If you cannot help but block any agent, choose the one with the least priority of all potential candidates to flee. The disobeyed order of an agent, not to go there, can be possibly corrected later.

Function: `plan_*alternatives*` of function make_new_decision

Parameters: agent (actual parameter SELF)

Value: only side-effects are of interest

Preconditions:

- no leave entries on the agenda
- entry *(alternatives list)* on the agenda, where list ≠ NIL

Postconditions: see method create_next_goal, especially:

- Either a new entry (next_goal x NIL) is on the agenda, where x is the place decided to go, or an attainable final goal

Description of the function:

If the final goal was reached, the agent is ready and it may stop. If the final goal is attainable, it becomes the next subgoal. The agents attempt to act.

In all other cases the function *find_heuristically_best_alternative* is called, which is the central planning function of the whole system.

Function: `plan_*next_goal*` of function make_new_decision

Parameter: agent (actual parameter SELF)

Value: only side-effects are of interest

Preconditions:

- *modify_agenda_of* was applied to agent.
- no alternatives-entry is on the agenda
- entry *(next_goal...)* is on the agenda

Postconditions: see method create_next_goal, especially:

- No alternatives-entry is on the agenda

Description of the function:

*plan_*next_goal** is called if and only if there is an entry (next_goal...), but no alternative-entry on the agenda. This implies the agent cannot find a better alternative for its prior choice, it must pursue the same one. Before doing that, the agent checks whether it can perhaps reach its final goal within the next step, or whether it is already on this goal. In case two the agents clear their agendas and stop to do anything. In the other case, the subgoal *(next_goal...)* is forgotten and thus deleted.

If the final goal is neither already reached nor attainable within the next step, the entry *(next_goal...)* will be checked similarly: Already reached ? ok. - Attainable ? - show that you want to act - Not attainable ? change to status "waiting" until the next cycle.

Function: `plan_*goal*` of function make_new_decision

Parameters: agent (actual parameter SELF), place of final goal, position of final goal

Value: only side-effects are of interest

Preconditions:

- *modify_agenda_of* was applied to parameter agent
- exact one entry on agenda: (goal...)

Postconditions: see method create_next_goal, especially:

- agenda unchanged, agent believes it can act

Description of the function:

A very simple function. The agents test whether they are already on the goal specified in the call. If so, the agendas are cleared and the agents become idle. Otherwise the agents show they can act if their goal places are perhaps attainable.

Function: `find_heuristically_best_alternative` of function plan_*alternatives*

Parameters: agent (actual parameter SELF), list of alternatives

Value: only side-effects are of interest

Preconditions: see function plan_*alternatives*, especially:

- the final goal is not yet reached and also not attainable within the next step

Postconditions: see function plan_*alternatives*, plus:

- a new entry *(next_goal...)* is on the agenda

Description of the function (domain specific features) :

From the list of alternatives (which is merely a list of places) several *sets* are formed which have got a different priority. In descending order priority:

1. empty_places+help	unoccupied places with corresponding help-request
2. good_places+help	occupied, surely attainable places with corr. help-requests

3. empty_places	unoccupied places without corresponding help-requests
4. good_places	like 2., but without corresponding help-requests
5. may_be_good_places	places which were attainable, if the "top agent" would leave in parallel
6. flattest_places	places, which are by no means attainable within the next move. In this set the places with the momentary minimum number of agents are gathered.

The sets are filled up with elements in exactly this order. If a place does not fit in the set of level i., it is tested whether it fits in i+1. Then the function select_heuristically_best_-alternative is called to decide what to do next. It is the heuristic core of the whole system.
Heuristic: The main idea is to collect candidates for the next decision in sets of different priority. First help-requests shall be obeyed. Then the amount of blockings must be restricted as much as possible. The choice shall be flexible.

Function: `select_heuristically_best_alternative` of function find_heuristically...
Parameters agent (actual parameter SELF), the six sets built in function find_heuristically_best_alternative
Value: only side-effects are of interest
Preconditions: see function find_heuristically_best_alternative, plus:
- the six sets are built correctly

Postconditions: see function find_heuristically_best_alternative
Description of the function:
To guarantee a flexible behaviour and a maximal prevention from deadlocks, the sets are not always proceeded in the same order. First the two sets with the highest priority which are not empty are selected. Then one of the two is chosen at random. At last one element of the chosen set is picked out. This picking out may be performed also at random (*empty_places+help*, *empty_places*) or directed (the other sets). Directed simply means that the set is chosen where an agent with minimal priority would be blocked.
Heuristics: The priority of sets must be obeyed sufficiently. The two sets with the highest priority are chosen a priori. But the decision for one of them shall be flexible in that way, that, for instance, in a deadlock situation not always the same fatal decision will be made. So the set with higher priority gets the weight 5 and the other the weight 1. This 5:1 ratio implies that the likelihood to select the set with higher priority is five times as high as the other.
Example: Assume we have elements in the sets *good_places+help*, *may_be_good_places* and *flattest_places*. The selection of the set of candidates takes place between *good_places+help* and *may_be_good_places* with the weight 2:1. Let *may_be_good_places* be selected despite the lower likelihood. The very place is chosen

now, where the agent with the lowest priority is under the interesting position. This ensures the prevention of blocking important agents. The lastly found element becomes the new subgoal of the seeking agent.

Information

Method: `inform_other_agents` attached to type AGENT

Parameters: none

Call structure: inform_other_agents
inspect_place

Preconditions:

- Each agent has found the best next goal from its individual point of view
- State of agenda: ([(next_goal a b)] [(goal x y)]) or ((next_goal a b) [(goal x y)]) (alternatives list)), where not both a=x and b=y hold and a is not in list
- just this order of goal on the agenda
- blackboard is either empty or has the newest entries from the actual negotiation cycle
- either method *create_next_goal* or method *conflict-resolution-1* executed
- if a conflict did occur in the prior negotiation cycle, the agent had surely believed it could act.

Postconditions:

- Agents which are obstacles for others have received individual messages *(leave...)* on their agendas
- Agents which are not blocked and believe they can act, wrote their wish (i.e. the desired next move) on the global blackboard
- Agents which are not blocked, but cannot reach the goal because it is too high, wrote help-requests on the blackboard.
- The agent thinks that no conflict has emerged.

Description of the method:

If the agent is not idle and has still (sub)goals on its agenda, the method inform_other_agents must do something.

If a conflict did occur in the last negotiation cycle, the agent has already written a new decision on its agenda. The former wish at the blackboard is now retracted, and if the agent believes it can reach the new goal, the new wish will be claimed on the agenda. In the other case, when the agents realizes the desired places is attainable by no means within the next move, the agents starts a finer survey of the constellation at the place of the goal. This is performed by the function *inspect_place*.

Mostly an agent will not be be involved into conflicts. The first it must check then is: Am I blocked by an "upper neighbour" ? If it is so, that obstacle is sent off and the agent itself shows that it cannot act yet. In the case the agent is free and nothing is known that prevents it from acting, it writes its wish on the agenda.
In the case the agent is free, however, and does not want to act, the reason for that must be investigated closer. This happens in the function *inspect_place* again.
Let us summarize some important facts for the method *inform_other_agents*. Here the negotiation cycle starts with the active sending of information about the next step planned. Remember that only agents, which believe they can act, may take part in negotiations and also only these are permitted to write wish-entries on the agenda. All other agents may - at best - ask for help one time and send leave-messages to each hindering agent.

Function: `inspect_place` of method inform_other_agents
Parameters: agent (actual parameter SELF), place to inspect, position to inspect
Preconditions: see method inform_other_agents, plus:
- the agent knows for sure that it cannot act within the current cycle

Postconditions: see method inform_other_agents, especially:
- blocking agents are sent off
- a help-request to reach goals which are yet too high is written on the blackboard

Description of the function (domain specific features) :
The exact goal may be not in reach, if the precise position is specified. In this situation a help-request is written on the agenda which orders an agent as support just below the position desired. In all other cases just the agent which is an obstacle is sent off (if there were none, the function would not have been called).
In particular, if no position is mentioned, not the agent on top of the stack is directly ordered to flee, but the agent which is upon the first agent (seen from the top) having a higher priority than the one one who called *inspect_place*. This implies, if this detected agent has left its place, all the other above it naturally must have left their place earlier. The way is then paved for the calling agent.
It is important to realize the consequences of changing decisions, when others may have tuned their behaviour to the former subgoals. Agents, which believe they can act and have already changed to the status "idle" to wait for the acting cycle, must be torn back into negotiations again.

Conflict resolution

Method: `conflict-resolution-1` attached to type AGENT

Parameters: none

Call structure: conflict-resolution-1
detect_leave_conflicts
plan_*alternatives*
find_heuristically_best_alternative
select_heuristically_best_alternative
update_agents

Preconditions:

- The agents are completely informed about the others intentions after their last reflection about the current situation
- State of agenda: ([(next_goal a b)] [(goal x y)] [(leave...)]*) or ((next_goal a b) [(goal x y)]) (alternatives list) [(leave...)]*), where not both a=x and b=y hold and a is not in list
- State of blackboard: ([(wish...)]* [(help...)]*), arbitrary order of entries
- All agents which believe they can act are still in the negotiation cycle, all others are idle

Postconditions:

- The agents in negotiations checked whether their desired goals are compatible with the prohibitions of the other agents
- The status of the agents which were in negotiations may be info.mally described as follows:

  idle: this implies that no conflicts occurred, the agents just wait to act in parallel with the others

  conflict: the agents' former decisions collides with the orders to flee of other agents (conflicts of type 1, see above). The agent has already tried to find a new decision. It indicates whether it has found one. Anyhow, agents in conflicts must later inform the other agents once again.
- The leave-entries are not removed from the agendas (this does not care...)

Description of the method:

If the agent is not idle and believes it can act and furthermore has at least one order to leave its place on its agenda, the method *conflict-resolution-1* must actually do something.

First the leave-orders are sorted with respect to the priority of the sender of the leave-messages.

Then the subfunction *detect_leave_conflicts* evaluates these ordered lists and returns a possibly modified set of alternatives. These function produces a very important side-effect: It marks whether a conflict has occurred, i.e. whether the personal next goal of the agent under consideration cannot be arranged with the orders to leave of other agents (for

more details, see *detect_leave_conflicts*). The modified set of alternatives represents exact these places where the agents still could go. It must be a subset of the former set of alternatives in the entry *(alternatives...)*.
Now there are two possibilities: Alternatives are still left: The agents calls the function *plan_*alternatives** to select heuristically the most appropriate one (*plan_*alternatives** is also a subfunction of *create_next_goal*, see above for the description). Otherwise, if no alternative remains, the agent knows that it cannot act for sure, it will indicate this and then leave the negotiation.
In the case that no conflict was detected, the agent needs not to change its decision. However, some alternatives may have become void because other agents with higher priority insist on going there. The agent must thus update the entry *(alternatives...)* on its agenda according to this new situation.

Method: `conflict-resolution-2` attached to type INFO-AGENT (Big-Brother)

Parameters: none
Call structure: conflict-resolution-2
Preconditions:
- agents have finished the negotiations. i.e. all conflicts of type 1 have been resolved
- agents either believe they can act (these are interesting for the method) or they surely cannot
- State of agenda: ([(next_goal a b)] [(goal x y)] [(leave...)]*) or ((next_goal a b) [(goal x y)]) (alternatives list) [(leave...)]*), where not both a=x and b=y hold and a is not in list
- State of blackboard: ([(wish...)]* [(help...)]*)

Postconditions:
- conflicts of type 2 are resolved, i.e. no contradictions between the agents' wishes exist anymore
- retracted wishes are marked "canceled" on the blackboard
- the data structure *actarray* (slot of INFO-AGENT) is preliminary instantiated. This structure later coordinates and synchronizes the parallel moves.

Description of the method (domain specific features) :
There are two main loops performed by the method: Loop-1 tests the agents whether they believe they can act. If so, all agents which want to go to place i are stored in the actarray in a list at place i. Especially not the agent itself is stored in the list, merely information about it in the form (X Y). Y is the agent's identifier (name) and X is the exact desired position. In most times X will first be NIL, but will be specified later to guarantee the scheduled acting. Loop-2 must coordinate the wishes of the agents. It is obvious that

conflicts of type 2 only occur if at least one agent insists on its exactly specified position at the place of the next goal it must reach. Otherwise, if no agent which wants, say, to go to place i, must obey an exact objective for the position, no problem arises: The agents can simply be rearranged in each list of *actarray* according to their priority.

How does Loop-2 proceed ? For each entry in *actarray*, the list is ordered by descending priority of the agents. If the first entry of each pair in the lists is NIL (no position specified) nothing remains to do. The agents could act in the new order. Otherwise the first pair in a list with a specification of the position is selected. The denoted agent can act (at least it believes so), so all agents with a higher priority must be inhibited to act. We use the heuristic rule, that a wish to act with specified height at the desired place has a higher priority that a wish without such a specification. So all the agents which have a higher priority but no specified position are stopped and will not perform their planned action. If they would, the agent with the exact specification surely misses its goal. All stopped agents are deleted from the actarray then and furthermore, their wish at the blackboard is marked as "canceled".

The next agents in the queue are checked then. All of them with a specified position must also be stopped, because they do not reach their desired position always. This is because all agents which want to go to the same place must have almost the same position specification[22], if they have got one at all. This follows from the fact that the agents under consideration believe they can act.

At last all agents either have retracted their wish or a preliminary coordination of those which still believe they can act is accomplished in actarray.

Function: `detect_leave_conflicts` of method conflict-resolution-1

Parameters: agent (actual parameter SELF), place of next goal, actual list of alternatives

Value: The new list of alternatives. Side-effect: Indication whether a conflict has occurred

Preconditions: like conflict-resolution-1, plus:

- Tested agents believe they can act and are not idle
- There are leave-orders on the agenda
- The leave-subgoals are ordered by priority of the orderer

Postconditions:

- The list of alternatives (return value) is manipulated correctly
- Possible conflicts of type 1 are indicated

Description of the function:

Each leave-order is processed separately. If the forbidden place in the leave-order is just the place the agent is heading to, a conflict has emerged. The function sets a flag to indicate that. In the other case, the alternatives must be checked. If the agent which sent

---

[22]The same position, or one below, or one above.

the order to flee has a higher priority than the agent under consideration or there are still other alternatives, the forbidden place will be removed from the set of alternatives.

Function: `plan_*alternatives*` of method conflict-resolution-1
see function plan_*alternatives* of function make_new_decision (paragraph creating new goals).

Acting

Method: `synchronize` attached to type INFO_AGENT

Parameters: none
Call structure: synchronize
Preconditions:

the slot actarray is preliminary filled with information about the schedule of the agents which believe they can act in parallel

- no conflicts of type 1 and type 2 exist yet
- the state of agendas and blackboard do not interest anymore

Postconditions:

- The entries in actarray are modified in such a way, that no agent which is in the queue to go to place X, but still at place Y, would be blocked by another agent in the queue of place Y, which acts earlier than the first agent

Description of the method:
Synchronize tests whether an agent A which is in a rear position of a queue, momentary is located at a place P, where other agents want to go. If nothing would be done, at least the first of the other agents willing to go to place P would block A so that its plans to move are frustrated. Big-Brother detects these situations and inserts delay-entries (NIL's) at the beginning of the list at just that entry of actarray that corresponds to place P. The number of delay-entries depends on the position of A in the queue and the number of delays already inserted at entry P of the actarray. More details are omitted, the method is not complicated, but rather technical.

Method: `cause_to_act` attached to type INFO_AGENT (i.e. Big-Brother)

Parameters: pause; the degree of slow motion for graphic output

Call structure:
```
cause_to_act
    test_acting_capability
        withdrawal
    perform_parallel_move
        graphics
            show_move_up
            show_move_horizontal
            show_move_down
            display_message
        update
            update_info-agent-slots
            update_agent-slots
    reset_flags
    check_deadlock_condition
```

Preconditions:

- *synchronize* has eliminated the major scheduling clashes
- The information up to now about the schedule of the agents which believe they can act is stored in actarray

Postconditions:

- The blackboard is empty
- The agendas of those agents which really acted contain only the final goal if they did not reach it within the move
- The agendas of the agents on their final goal place is empty
- The agendas of the other agents, which did not act, remain unchanged
- The move was shown graphically
- The knowledge bases of the agents are updated adequately
- Possible deadlocks after the performed moves are checked

Description of the method (domain specific features) :

The body of the method is an endless loop which must be exited explicitly. In each cycle the first elements of the lists in the entries of actarray are regarded, because they are the potential candidates for one parallel move. At the end of the cycle these first elements are removed.

In *cause_to_act* the first thing to do is to verify whether the agents which have announced that they can act and still believe it, really can. Finally in this moment it is clear whether an agent, which should leave its place to give place for another agent willing to go right there, actually will leave. Up to now the agents believed they can go, now they are going to know it exactly.

The function *test_acting_capability* performs these tests and, furthermore, some sophisticated synchronization tasks. After this the order and the parallelism of one joint move is fixed. All agents which are at first position of their lists in the respective entries

of actarray, will act in parallel. If there is NIL at this position, no agent will go to the corresponding place at this instance of moving.
Then the exact new position of the agents is determined and written at the first position of the pairs in actarray (see above). This is especially important when an agent flees from the goal place of another. There may be a big surprise when the second agent realizes that it gets a position which it had not foreseen.
A list is built consisting of just the agents which can perform one parallel move.
Then the function *perform_parallel_move* is invoked with this "actlist" as an argument. This function manages the technical details of the moves, the graphical display and updates the knowledge bases of the information agent as well as the concerned other agents.
The loop of acting runs as long as there are still non-NIL entries in the actarray.
In the end the flags of all agents are reset as a basis for a new cycle beginning with the reflection about new/old goals (*create_next_goal*, see above). The blackboard is cleared.
At last, possible deadlock situations are detected and the reasons therefore are eliminated (see function *check_deadlock_condition* for a closer description).

Function: `test_acting_capability` of method cause_to_act
Parameters: info-agent (i.e. Big-Brother)
Value: only side-effects are interesting
Preconditions: see method cause_to_act, plus:
- actarray (slot of info-agent) is not empty

Postconditions:
- only agents which really can act are denoted in actarray

Description of the function (domain specific features) :
Only the first elements of the lists in the entries of actarray are regarded. The proceeding for each place (each entry in actarray, respectively) is as follows.
The agent represented by the first element of the list under consideration in an entry of actarray is called "testagent". The agent on top of the goal place of *testagent* is "agent_at_goalplace". The main work is to do, in case *agent_at_goalplace* exists, i.e. the goal is occupied, and this agent does not tolerate *testagent.*
Now *agent_at_goalplace* must also be denoted in actarray to let *testagent* a chance to act.
A search is started.
If an *agent_at_goalplace* is actually found in actarray, a double-index memorizes its position: (x y), where x is the index of actarray and y is the position in the current list in entry x. There are two possibilities:

y is zero, which implies *agent_at_goalplace* wants to act in parallel with *testagent*. That is fine, the only problem is: *agent_at_goalplace* itself may not know if it really can act. Thus it possibly depends on a third agent. The chain of dependencies is recorded in a data structure. If the last agent in that chain fails to keep on acting, all others must reject to act, too.

If y is not zero, things become harder. *agent_at_goalplace* will act, but unfortunately later than *testagent*. Then we use a tricky heuristic: We know that *agent_at_goalplace* is not blocked and that *testagent* would tolerate it (because this does not hold vice versa). testagent and *agent_at_goalplace* can in fact act in parallel, namely if *agent_at_goalplace* goes to *testagent's* former place. This is possible because the agent under *testagent* will surely tolerate *agent_at_goalplace*. Later *agent_at_goalplace* may pursue its genuine subgoal. We call this heuristic the concept of "acting pairs", because two agents join to make a deal with advantages for both.

Now to the case *agent_at_goalplace* must leave (from *testagent's* point of view) but has no ambitions to act. Then intentions to act of *testagent* must be frustrated, and so all the other agents which perhaps depend on *testagent's* leaving.

Another basic situation emerges when *agent_at_goalplace* tolerates testagent. This seems to be harmless and not worth to be regarded. But what to do if *agent_at_goalplace* wants to act later than *testagent* ? testagent cannot move, *agent_at_goalplace* would surely be blocked. *testagent* also cannot be delayed (as in *synchronize*) anymore, because all dependencies of delays would be confused. We preliminary have chosen the safest way: *testagent* resigns and cancels its ambitions to act.

Function: `perform_parallel_move` of method cause_to_act
Parameters: info-agent (i.e. Big-Brother); actlist: list of parallel acting agents; pause
Value: only side-effects are interesting
Preconditions:

- Exact positions of the goal places of the moving agents are calculated and enscribed in the parameter *actlist*.

Postconditions:

- One parallel move (all agents that could act contemporary) was displayed on the output window
- The knowledge bases of the respective agents are updated accordingly

Description of the function:

Despite of all tests to ensure a "perfect" schedule of the moving agents there are still rare configurations, where mutual influences and effects of action might produce forbidden

moves. To avoid exhaustive and cumbersome investigations, a last check is performed at the beginning of *perform_parallel_move*: Can the agent really, really, really act ? If not, it is retracted from actlist, no more fuss.
Then the functions *graphics* and *update* are called. We will not discuss them further in details here (for *graphics* see Appendix (6)). *update* not only changes the knowledge bases of the agents which act and that of Big-Brother. It must also take under consideration the knowledge of those agents which had been under the moved agents and of those which are now under them. The information about the upper neighbours must be updated.

Function: `check_deadlock_condition` of method cause_to_act
Parameters: info-agent (i.e. Big-Brother)
Value: only side-effects are interesting
Preconditions:

- All moves are performed, displayed and the knowledge bases of the agents are updated correctly
- Especially: Agents which acted have at most their personal goals at their agenda, the agendas of the other agents remain unchanged

Postconditions:

- Unless all agents are at their final goals at least one agent is active and attempts to reach a goal in the next reflection/acting cycle

Description of the function:
A special kind of deadlock may occur, if the agents have performed their moves. We call it "goal deadlock". A goal deadlock demands several contextual factors:

- All agents have either an empty agenda or *(goal...)* is the only entry
- Not all agents have reached their goals
- No agent can reach its goal within the next move, so no one realizes the necessity to send another agent off

All agents will stay inactive, according to the paradigm of eco-problem solving: "I will do nothing unless I am forced to."
This situation is detected by Big-Brother, it will intervene and force the agent with the highest priority of the ones which did not reach their goal yet, to become active.

4.5.3. Heuristics

In this chapter we want to point out the special heuristics employed in TOHPAR and interpret them from a more abstract point of view. The heuristics are an important part of the procedural knowledge of the agents. They may roughly be divided into two main groups, where each group consists of two subgroups. The main groups are:

- Planning the next action of each agent
- Handling of conflicts

In the following the heuristics have got a number. This number is only used for identification, it has no meaning for the actual run of TOHPAR.

Planning the next action

This means pursuing the way to the goal and evolving suitable strategies to find good candidates for the next move.

1. Modifying and preparing the current knowledge

Heuristic 1 (rule-1; method create_next_goal)
If you have some identical orders from different senders, act in favor of the sender with the highest priority.

Heuristic 2 (rule-2; method create_next_goal)
If a former chosen but not yet performed action is suddenly forbidden, it must be discarded at once.

Heuristic 3 (rule-3; method create_next_goal)
If a possible alternative is suddenly forbidden, it also must be discarded immediately.

Heuristic 4 (rule-4; method create_next_goal)
To provide a basis for a new and unbiased decision of the next action, all prior decisions must be retracted and joint with the rest of alternatives.

Heuristic 5 (plan_*leave*; method create_next_goal)
If all alternatives are forbidden, ignore the sender has the lowest priority if this alternative is achievable. Senders with higher priority will not be hindered.

2. Making the decision what to do next

Heuristic 6 (plan_*alternatives*, plan_*next_goal*; method create_next_goal)
If your personal goal is reached or at least attainable, pursue it and forget all the other subgoals.

Heuristic 7 (find - / select_heuristically_best_alternative; method create_next_goal, conflict-resolution-1)
To select one from several alternatives build disjunct sets of decreasing priority. These sets must be subsets of the current set of alternatives. Then take the two sets which are not empty and have the highest priority. Choose one of both, where the weighted likelihood for the better alternative is 5:1.
From the chosen set select one alternative with an appropriate selection function.

Heuristic 8 (plan_*next_goal*; method create_next_goal))
If there is still only one alternative and nothing new has evolved, keep on pursuing this alternative. May be it works this time.

Conflict handling

Conflicts are pretty frequent phenomena in parallel scenarios. The hard task is to detect them. Once detected, an appropriate means to eliminate them should be not far away.

1. Conflicts concerning the decisions about the next action to perform

Heuristic 9 (methods conflict-resolution-1, conflict-resolution-2)
Conflicts with forbidden goals are resolved in negotiations, whereas conflicts emerging from clashing subgoals are solved by appropriate retracting of wishes (decisions) to act.

Heuristic 10 (detect_leave_conflicts; method conflict-resolution-1)
If a forbidden alternative is equal to the newest subgoal, a conflict (of type 1, see above) has emerged. Otherwise taboos which occur during negotiations are only obeyed if either alternatives would be left or the sender has a higher priority than the receiver of the taboo.

Heuristic 11 (method conflict-resolution-2)
Wishes with full specification of the goal have a higher priority than those with only a partial specification.

2. Conflicts emerging from scheduling clashes

Heuristic 12 (method synchronize)
If an action causes a blockage of a potential future actor, these action must be delayed appropriately. The actions must be performed at least in parallel.

Heuristic 13 (test_acting_capability; method cause_to_act)
If an action cannot be performed because another future actor is an obstacle, try to build "acting pairs". Cause the obstacle first to make an intermediate action to give place for yourself under the premise that the other agent can later perform its planned action without problems.

Heuristic 14 (test_acting_capability; method cause_to_act)
If an action cannot be performed because the actor would block a future actor (synchronization effects may have been compensated), the action is cancelled.

Heuristic 15 (check_deadlock_condition; method cause_to_act)
If no agent wants to do something, the one with the highest priority of those which have not reached their goals yet, is forced to become active.

4.5.4. Benchmarks

TOHPAR gives no options to choose different heuristics. So the results are preliminary presented solely with a comparison to the result of the best sequential heuristics of TOHSEQ (A more abstract comparison of the two scenarios is made in chapter 4.6.).
The implementation of the parallel scenario produces no deterministic behaviour. If an agent has more than one alternative and cannot decide the very best, it will choose its next goal at random. Each alternative has an identical likelihood. But a decision among alternatives, which do not differ from a local point of view, may convey global consequences. So the results may differ significantly.
The results are given in Figure 4.20 as follows: We always have n=10 agents, the number of places varies (first diagram x-axis, second diagram y-axis). The first ten bars in the upper diagram (the total number of moves) represent ten ordered test results, the 11th bar

the average of them, and the 12th bar represents the result of the best sequential heuristic. The second diagram reveals time measures, a minimum time and a maximum time for each configuration are presented.

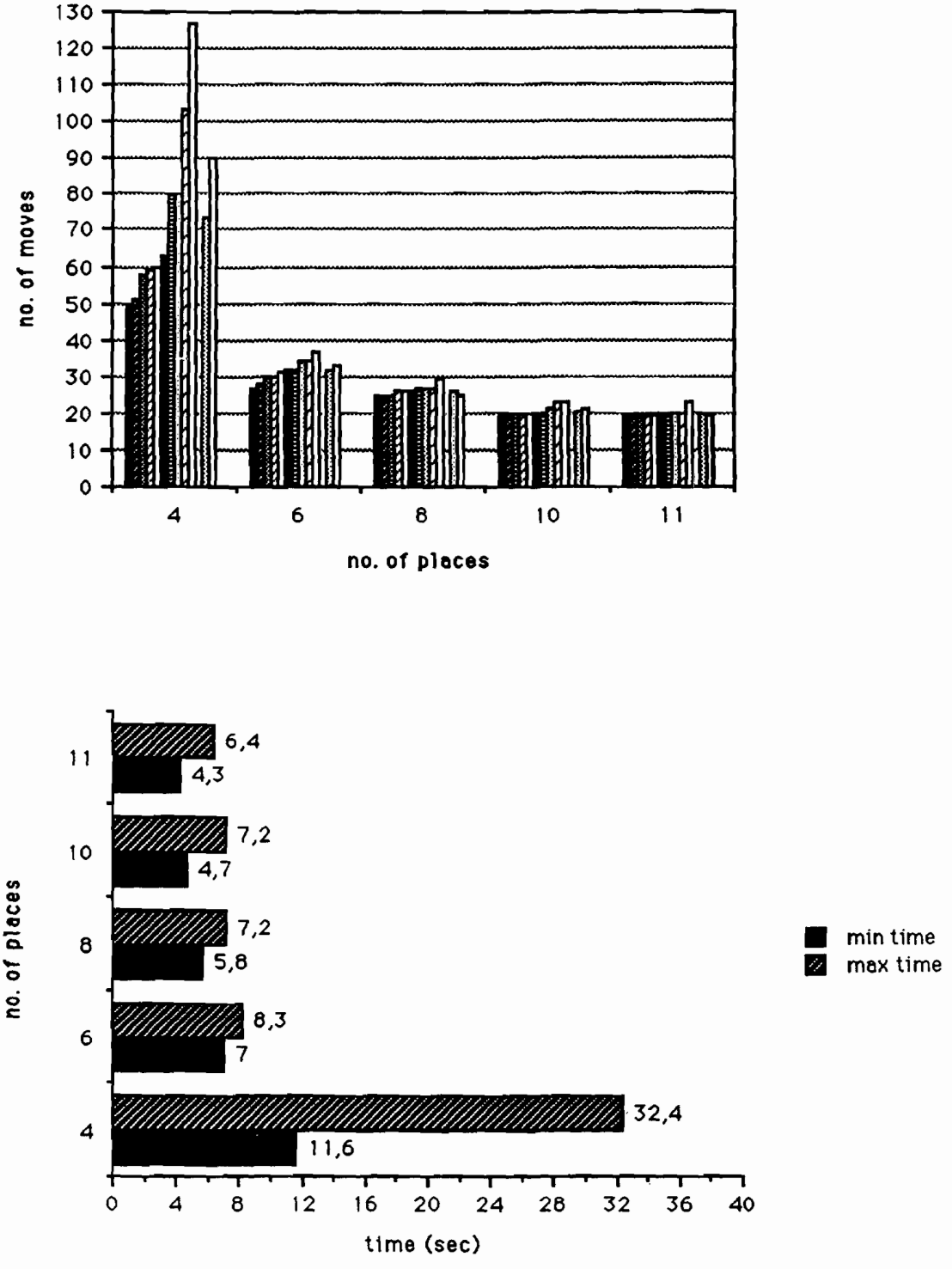

Figure 4.20 : Benchmark results for TOHPAR

To make hard statistic statements, a multitude of tests would be necessary. Then we suppose that a Gaussian distribution would emerge.
What is already significant at the results presented here, that TOHPAR achieves its best results when there are many agents in relation to few places. The more places the configuration gets, the worse become the results compared to the best results of TOHSEQ.

Summary of the Test Results

It is very impressive that the distributed implementations produce such good results. Compared with the information about the effort of traditional search (cf. capter 4.2.), where, for instance, in a scenario with n=10 agents and m=4 places 4^{10} different states potentially must be traversed and a search tree for "brute force" breadth first search comprises about 20^{50} expanded nodes. If we really would proceed in this way, the "agents" would perform 20^{50} moves until they reach the goal state. In this perspective the results ranging from 50 to 229 moves for the parameters above are indescribably better. And consider the amount of time which would be necessary for 20^{50} moves, which is a number almost beyond our conception. The time results of a few seconds (see above) hence also show the feasibility of the distributed approach.

The results must also be seen in the light that only very simple local heuristics were used to facilitate the decisions of the agents. These coarse "rules of thumb" suffice to come very close to the optimum and also generate satisfying solutions were no optimum is known at all. This is a promising experience in order to cope with large search spaces where no global strategies are possible. From another point of view, from the local and distributed perspective, certain problems may become manageable.

4.6. Comparison of Sequential And Parallel Version

The parallel version TOHPAR is much more complicated than the sequential version TOHSEQ. Do the results substantiate the efforts ? At a first glance this does not seem to be so. The results of TOHPAR are pretty often even worse compared to the best heuristics of TOHSEQ. We will now try to explain this apparently strange behaviour and then argue why the parallel version is of principal importance after all.

TOHPAR provides a non-deterministic behaviour. The problems going along with that and with the parallelism (cf. chapter 4.8.) have a larger impact on the result, if the problem must be solved almost sequential ("many places, few agents"). Then a wrong or short-sighted local decision can hardly be compensated by effective parallel moves, because these are not often possible.

So the restriction to sequential execution of actions becomes an advantage and, on the other hand, the amount of freedom lying in decisions and possible parallel moves grows to a major drawback.

A observer of the parallel scenario who is unfamiliar with the principles of the behaviour of the agents will perhaps be disappointed of the average amount of parallelism in TOHPAR. He watches the proceeding and wonders why many agents which have performed an action often remain idle a long time, whereas only a few really act. They argue that these idle agents could better have rearranged in parallel so that, for instance, more places become free and other agents with a higher priority may go t'.ere.

This is a good idea, but it cannot be executed within the paradigms we presumed for multi-agent scenarios.

First, the knowledge of the agents is restricted. They only know their personal situation, the wishes and orders of the other agents. But to recognize that it is, e.g. convenient to build an intermediate stack to clear as much places as possible demands a huge portion of global view and global problem solving strategies. Also planning more steps into the future could support a more rational behaviour.

Second, the agents are rather passive than busy per se. This is an intrinsic issue of the strategy of eco-problem solving we adopted for our scenario. If the agents are not blocked, but cannot reach their desired goal, they do nothing unless they are ordered to go away. They simply wait either for a kick or that their goal becomes attainable. But when they receive an order to flee, this radically changes. They become active until this order is fulfilled. Blocked agents also begin to do something, and that is to send messages to order their upper neighbour to go away. So the central message type "go away but avoid..." causes agents to become active when their genuine goals are out of reach. But after the agent has sent off all obstacles and performed its move, it will stay

idle until either its lower neighbour wants to act or another agent tries to occupy the first agent's location.

The parallel scenario TOHPAR is worth the effort put in it. Many procedures merely have the duty to provide the pseudo-parallelism. Thus in some sense we make a huge effort in creating effects (e.g. parallel interferences) where we must elaborate more issues to tame them again (negotiation, synchronization). But just these effects enrich the study of TOHPAR. They naturally cannot occur in the sequential setting. In detail, we are confronted with:

- Incompatibilities of concurrent decisions
- Interferences by parallel acting
- The demand for negotiation
- Conflict-solving strategies
- The possibility to ask for help

All these items either are not relevant in TOHSEQ or do simply not emerge. But natural multi-agent scenarios heavily cope with these aspects. Thus TOHPAR may provide a basis for further work in DAI planning concerning the behaviour of more natural Multi-Agent worlds.

The sequential version cannot achieve this. Its relevance was to get a first feeling for the scenario to be modelled and to find some elementary heuristics. Also this objective was accomplished.

4.7. A Model For The Agents' Behaviour

In this chapter we want to provide a more formal model of the agents' behaviour in the parallel realization TOHPAR. We use the means of a non-deterministic finite automaton to reveal the changes of the inner states of the agents which act in the scenario.
The agents use four flags in their knowledge base to control their behaviour and to show in which inner state they are. Despite of the explanation of the meaning of the flags in "General System Design" (see chapter 4.5.2.), they were not explicitly mentioned when describing the procedural knowledge of the agents ("The Modules Of The System").
Figure 4.21 shows the graph of the automaton. As usual, nodes represent states and edges represent changes of states. There are four types of edges, corresponding to the execution of four main modules of TOHPAR:

①: create_next_goal ②: inform_other_agents
③: conflict-resolution-1 ④: cause_to_act

The other methods between ③ and ④ do not change the states anymore, they are omitted for the sake of a comprehensible notation. Also the elimination of "goal conflicts" after performing the parallel moves does not have an explicit corresponding edge. It causes the effect of a way via edge ④ to state 0 and then via ① to state 8.
The states got a running number by interpreting the four flags as a four bit integer ("+" means 1, "–" means 0; the first flag is the min-bit). The states 6,7,14, and 15 cannot occur because the conflict-flag and the wait-flag are never set simultaneously. [23]
The traversal of the graph allows sequences of edges in the form $[① [② ③]^{+} ④]^{*}$, where *,+,[,] are the usual meta-symbols. These possible sequences correspond to the general system design. ① - ④ is the overall "reflecting/acting" loop, lasting until each agent has reached its goal. ②,③ is the negotiation cycle, where conflicts must be resolved.
It is very obvious when agents are really involved in negotiations and when they are only waiting for others to finish them. Changes of states following ②- or ③-edges indicate there is a new decision born or a decision must be corrected. On the other hand ②,③-edges going back to the same state show: This agent does not join the negotiations at the the moment, it is idle. There are two cases: "idle–" : The agent waits for a complete new loop of the scenario, because it cannot act in the current cycle. "idle+" : The agent waits for the other agents which may act in parallel with it. It believes that it can act. In this situation a very special change of state, marked with a "*" at the respective edges, may take place. An agent believes it can act, but another agent frustrates this believe by changing its decision. This is the case when a corrected decision blocks the wishes of other agents. These other agents are then back in negotiations again. The edge marked

[23] An agent cannot be waiting when it is involved in a conflict. The conflict has to be resolved at once.

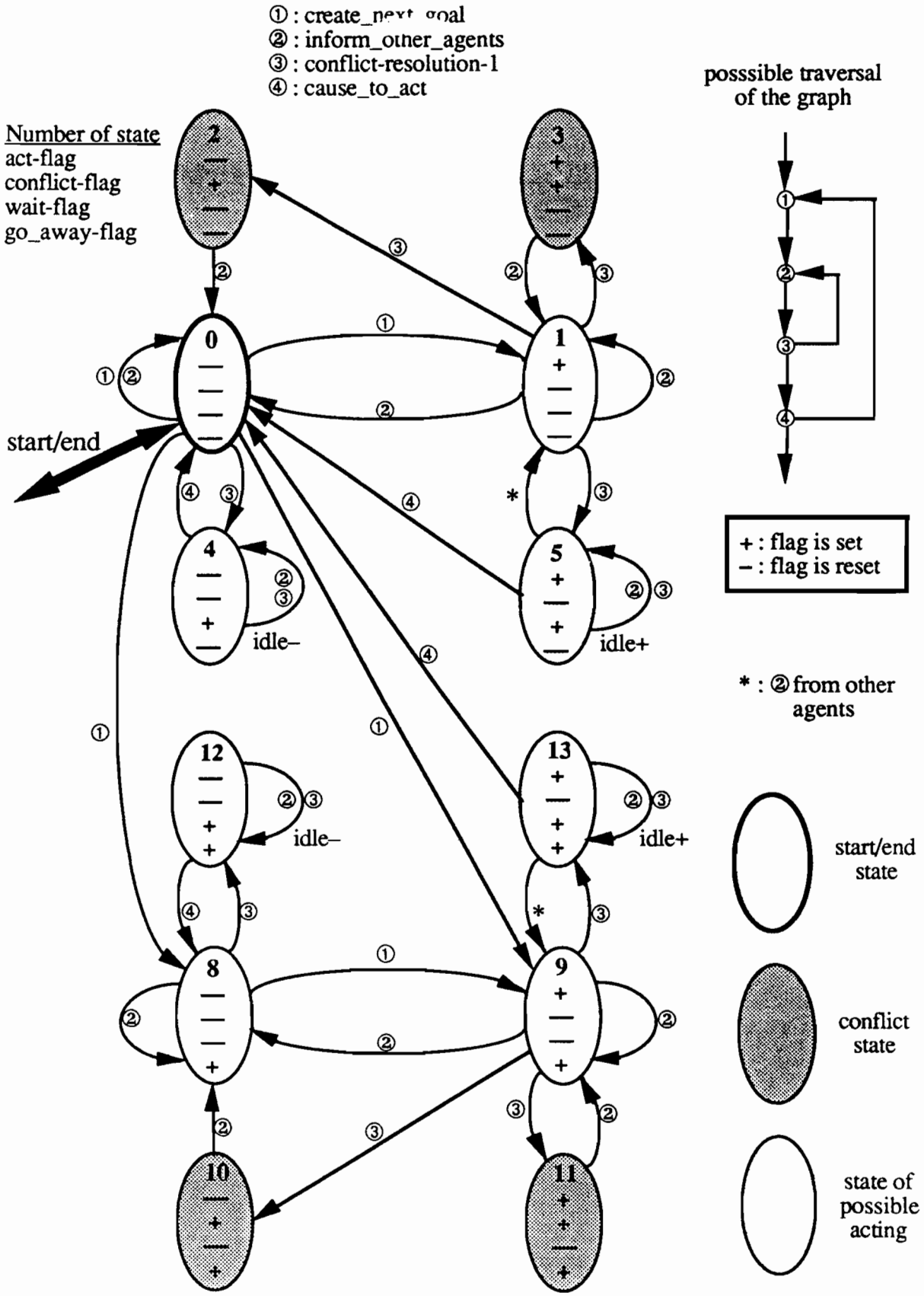

Figure 4.21 : The Graph of the Automaton Model

with a "*" represents the only change of state (internal knowledge) which is not induced by the agent itself but by another one.

Much could be said about the implicit information lying in the graph. But this would likely become a boring repetition of the exhaustive descriptions above. After all the model provides a concentrated and informative overview about the possible behaviour of each individual agent.

4.8. Open Problems

The benchmark results prove the potential capabilities of the parallel version TOHPAR, but also the current handicaps (cf. chapter 4.4.4. and 4.5.4. for detailed benchmarks). Two major research areas still have to be tackled, the heuristics and the problem of parallelism per se.

For the heuristics, exhaustive validation tests must be conducted. Each heuristic must possibly be refined or changed. The next step would be to parametrize the heuristics and provide the agents with the feasibility of learning. They could memorize which heuristic worked well in what special situation or vary the utilization of heuristics when a solution does not come nearer over a series of cycles.

Parallelism conveys several classes of problems. First, an adequate simulation environment must be installed. Although there has been much effort put in TOHPAR to provide pseudo-parallelism, it cannot be as flexible and natural as a system with real parallel processes. Especially the static system design with the sequence of the module executions inhibits many interesting effects: Asynchronism, critical races of messages, temporal dependencies. On the other hand the realization in TOHPAR lets many agents spend much time in waiting for others to negotiations or actions. Simple delays where the agents would do absolute nothing, could otherwise be possibly used for estimating and evaluating future actions, or helping others by, say, extensive computations.

Another issue is the prediction of the effects of parallel acting. In fact there is still the possibility of everlasting loops of actions. Because the agents only have a restricted world view, one single decision may appear optimal, whereas a collection of decisions during a parallel move need not be beneficial for all. Goals of one agent may be frustrated by moves of other agents. If these interferences occur frequently, a loop may emerge which cannot be exited. This problem is hard to handle.

One solution, which was already tested with good results, is the probabilistic selection of an alternative. We do not use always the same decision criteria, but allow some exceptions from the - normally - best choice. The best choice gets the weight 5:1 with respect to the next good choice. But if there is only one choice, this idea cannot work.

Bookkeeping procedures to detect loops in behaviour are another means. But this would bring an enourmous overhead of data to store. Also the search for actions already performed in the past, will last longer from move to move. So this "brute force" method seems to be inappropriate.

A formalism to detect and eliminate such loops should be developed. It could be leaning on related work on deadlocks in operating systems [Ne86].

5. The Towers of Hanoi in the General DAI Framework

5.1. Embedding In DAI Terminology and Classification

The scenario of the Towers of Hanoi shall be characterized now in the light of the overall DAI terminology. First of all, it must be mentioned that the implementation of TOHPAR was made rather straightforward and the whole project is surely still a prototype. Therefore a classification must strictly distinguish between general aspects and special ToH features of the system. We want to point out the general DAI aspects in the following.

The kind of problem solving fits perfectly in Ferber's paradigm of *eco-problem solving* ([Fe90], see also chapter 3.6.). The agents proceed in pursuing their goals, thereby sending other agents as obstacles off. These agents obey the order and leave their place; after finding an alternative to move, they go on with their own goals. Due to the local decisions and to some aspects of non-deterministic choice, the visible behaviour resembles sometimes a natural population of dumb individuals like ants.

The eco-metaphor implies in some sense that the agents tend to be more re-active than active in their world. They act only when they can reach their goal in one step or when they are sent off by other agents. Their decisions comprise no difficult planning in advance, only the actual situation of the scenario is regarded and perhaps one possible next step of other agents is taken into consideration. Though each agent has a small and restricted knowledge base, it is more appropriate to characterize the ToH-agents as behaviour-based (i.e. reactive) than as knowledge-based.

Embedded in the general characterization of eco-problem solving is the proceeding to determine the next move of each agent. This is done by heuristical planning, where the heuristics are strictly local and employed from the agent's individual point of view. Because planning and execution are steadily interweaved, i.e. after one planning step follows (at most) one action, the ToH scenario is an example for distributed planning, where planners and actors are identical.

The agents behave cooperative to a considerable extent, benevolence would be an almost too weak classification. In fact, when an agent sends another one off, a temporary master-slave relationship is established. The agent, which is sent off (the slave) must obey the order of its master. After it has done this (or while doing this), the slave may build up dependencies to other agents where it is then the master.

The communication is performed either by direct message passing (writing on one other's agenda) or by broadcasting via a blackboard. The messages are formalized and are implicitly processed according to their types. There is no explicit communication protocol.

Negotiations are performed by fixed rules and priority relations. They occur in certain conflict situations and can be always resolved.

The whole actual situation in the scenario is known by the agents. Thus there does not exist any uncertainty about the environment. Only the intentions of other agents are not transparent at all. They only announce one single next step in the moment they believe they can act. Thus a certain amount of uncertainty about the inner status of other agent remains, the "why does an agent act like that" is normally unknown to others. More knowledge of the other agents' intentions would supply more constraints for an agent's next decision. Up to now three main types of constraints exist:

- physical constraints: an agent cannot reside somewhere in the air, according to the law of gravitation it needs permanent support (except it is just acting)
- general laws/rules of the scenario: an agent must not go to a place where it is not tolerated (for ToH: where a smaller agent is already located)
- transient constraints: temporary restrictions to move to certain places (for ToH: "go away, but do not go to X" implies place X is forbidden for the next move)

At last, some words about the system conception. The scenario is implemented in a synchronous manner, i.e. all the problems with asynchronism like message delays, message crossing, inconsistent states, etc. do not occur. A global blackboard and a global model of the environment is managed by a single informatior agent, to whom all "normal" agents have access. Furthermore all agents have individual agendas to handle the local decisions. The system thus is a blackboard/agenda system.

In [Hu87] eight dimensions to classify DAI systems are presented. We apply this characterization also to the TOHPAR system. Though it cannot be objective and may further be confusing in some aspects, it gives a coarse overview of some features of TOHPAR with respect to other systems. The "▲"-mark indicates the circa-position of TOHPAR.

Dimension	Spectrum of Values		
System Model	Individual... ▲	...Committee...	...Society
Grain	Fine...	...Medium... ▲	...Coarse
System scale	Small...	...Medium...	...Large ▲
Agent Dynamism	Fixed... ▲	...Programmable...	...Teachable... Autodidactic
Agent Autonomy	Controlled...	▲ ...Interdependent...	...Independent
Agent Resources	Restricted... ▲		...Ample
Agent Interaction	Simple...		▲ ...Complex
Result Formation	By Synthesis...	- not relevant -	...By Decomposition

5.2. Relations To Other Scenarios and Further Work

The Towers of Hanoi scenario may be a nice setting to demonstrate the capabilities and foundations of multi-agent problem solving and distributed planning. But after all, it is only a toy example, applications of practical relevance seem to be far away. We will now investigate which steps are necessary to come to real world scenarios.

One central condition for problems which can be adequately modelled in the paradigm of eco-problem solving like the ToH problem is that all components of the scenario must be known in advance. All parts have determined positions, attributes and perhaps a priority value. Furthermore, the "goal" of the system must be individually, consistently, and simply decomposable to each concerned part.[24] When each part is at its goal place, the overall goal is fulfilled. Presuming these preconditions and an adapted version of the heuristics presented so far, the parts of the problem become the active agents, which will pursue their own goals without central control and thus at last attain the overall system goal.

At a first glance, these preconditions seem to be very restrictive demands. But it is astonishing how large the class of problems is, which can be handled. Arrangement-, sorting-, and assembly-problems. All have in common that the components have an exact initial state (position, attributes) and a well-defined goal (normally the position). The way from start to goal is difficult and relevant because of large search spaces, insufficient global knowledge and heuristics (see chapter 4.2.).

But how to achieve the modelling of such problems ? First of all, the heuristics employed in the ToH scenario have to be thoroughly investigated and generalized. Also the relevant attributes have to be abstracted.

A first step would be a Blocks World scenario. A wider variety of agents has to be represented, different "skills" and different attributes must be regarded. But the Blocks World is not too far away from the Towers of Hanoi. A straightforward implementation can be achieved soon. But then a generalization is absolutely necessary. It could end in a prototype of a simple multi-agent shell to provide a framework for problems like those mentioned above.

The results gained should put forward the implementation of real world scenarios. Examples are (cf. chapter 2.2.) a shunting station and a loading yard of a transport agency. In both settings all important parts are modelled as autonomous agents. This will prove the feasibility of the multi-agent approach to more complicate scenarios.

There is a good hope that the new paradigm of eco-problem solving might pass the border of toy applications to really relevant problems.

---

[24] This is not possible for puzzles like "tangramm", where the goal decomposition is the genuine problem and the way from start to the decomposed goal is trivial.

6. Conclusion

We have introduced some central concepts of Distributed Artificial Intelligence and implemented a Tower of Hanoi scenario in the eco-problem solving metaphor. The aspects of this implementation were exhaustively outlined.

The ToH scenario was only a first step to get experience in multi-agent systems, insight into the problems of parallelism, and an understanding of the capabilities of local heuristics. This step was successful. It was shown that rather simple local heuristics can produce a considerably good global result.

The most important next step is to distinguish between special issues which are tailored to the ToH setting and general ideas which are usable for more universal multi-agent settings. The long-term goal is to construct a multi-agent shell or workbench where several different scenarios can be modelled and tested.

It must not be forgotten that certain serious problems of multi-agent interaction are not treated in the ToH implementation: Asynchronism is avoided, so very hard problems concerning time and consistency simply cannot occur. The subgoals do not interact, so a solution can always be found. But what is in the opposite case ?

To find answers to questions like these and to provide a test environment is the goal of a new project of the German Research Center on AI (DFKI). It was outlined in the proposal for a multi-agent workbench, RATMAN ([BüMü90], see also chapter 2.8.). In the end a wide variety of different multi-agent systems will be able to be modelled with RATMAN. The results and the expertise gained in various simulations will guide the further work.

To better understand the potential of a society of agents, an interdisciplinary approach seems to be urgent. Researches from various disciplines should engage in concerted studies. Social scientists, linguists, cognitive scientists, psychologists, philosophers, and biologists may bring together their knowledge in order to find the foundations why, for instance, human societies function as well as they actually do.

But it is doubtful whether human societies should really be the goal of modelling multi-agent communities. I have the feeling that this is not desirable at all. Why should we simulate humans, as long as we have real human experts ? This question may sound heretical, but it is meant as follows: It may be not an appropriate (even not a possible) way to take the position that agents must be made more and more "intelligent", so that at last, the society of agents resembles a society of humans with respect to certain problem configurations. Perhaps the unbounded enhancement of individual expertise concerning problem domains will come soon to a limit, where no progress is achievable anymore. As seen in chapter 3.6. the power of the overall system decreases when the agents become locally too intelligent. A possible consequence is that it is useless that agents augment their expert knowledge if their "social knowledge" does not increase. Social knowledge

comprises, for instance, the will and the ability to reach a quick and good compromise. It is easy to imagine endless discussions of human experts, say surgeons about the explanation of seldom symptoms, when the opinions differ. Such a system is not efficient.

I propose to restrict on problems where complex global strategies can be broken up into simple local strategies (heuristics). This could put forward a new AI paradigm, resembling Minsky's society of mind [Mi86]. Then a complex system like the human would be completely built up by simple local processes, which are by far not exact, but heavily communicative in order to reach quite a good global solution[25]: the human behaviour.

But, as mentioned above, a copy of the human (a "humunculus") should not be the goal of DAI research, nor a copy of the human's societies. Surely there is a lot to learn from a collection of such "meat-machines", this may be an impetus for research in the next years.

DAI research has now to prove that the ambitious goal of realizing the first real world scenarios and thus gaining practical relevance can be attained

[25]Concerning the average case...

7. References

[Al84] James F. Allen
Towards a General Theory of Action and Time
Artificial Intelligence 23 (2) 1984 pp.123-154

[BiAlFoLeBa87] R. Bisiani, F. Alleva, A. Forin, R. Lerner, M. Bauer
The Architecture of the Agora Environment
in [Hu87] pp. 99-117

[B 89] Alan H. Bond
The Cooperation of Experts in Engineering Design
in: [GaHu89] pp. 463-484

[CaMcSt83] Stephanie Cammarata, David McArthur, Randall Steeb
Strategies of Cooperation in Distributed Problem Solving
in: Proceedings of IJCAI 1983 pp.767-770

[Ca88] Luis Eduardo Castillo Hern
On distributed artificial intelligence
in: Knowledge Engineering Review Vol.3 No.1 (March 1988) pp.21-57

[Ch81] B. Chandrasekaran
Natural and Social System Metaphors for Distributed Problem Solving: Introduction to the Issue
in: IEEE Transactions on systems, man, and cybernetics
Vol. SMC-11 No.1 (January 1981) pp.1-5

[Ch87] Ernest Chang
Participant Systems for Cooperative Work
in: [Hu87] pp. 311-339

[CoMePo89] S.E. Conry, R.A. Meyer, R.P. Pope
Mechanisms for Assessing Nonlocal Impact of Local Decisions in Distributed Planning
in: [GaHu89] pp. 245-258

[CoMiCa90] Rosaria Conte, Maria Miceli, Cristiano Castelfranchi
Limits and Levels of Cooperation : Disentangling Various Types of Prosocial Interaction
in [MA90] pp. 207-217

[DaSm83] Randall Davis, Reid G. Smith
Negotiation as a Metaphor for Distributed Problem Solving
in: Artificial Intelligence 20 (1983) pp.63-109

[De87] Keith S. Decker
Distributed Problem-Solving Techniques : A Survey
in: IEEE Transactions on systems, man, and cybernetics
Vol. SMC-17 No.5 (September/October 1987) pp.729-740

[DuLe89] Edmund H. Durfree, Victor R. Lesser
Negotiating Task Decomposition and Allocation Using Partial Global Planning
in: [GaHu89] pp. 229-243

[DuLeCo87] Edmund H. Durfree, Victor R. Lesser, Daniel D. Corkill
Cooperation Through Communication in a Distributed Problem Solving Network
in: [Hu87] pp. 29-58

[Fe90] Jacques Ferber
The Framework of Eco-Problem Solving
in: [MA90] pp. 103-114

[FiLo86] Nicholas V. Findler, Ron Lo
An Examination of Distributed Planning in the World of Air Traffic Control
in: Journal of Parallel and Distributed Computing 3 (1986) pp.411-431

[Ga89] Les Gasser
MACE: High-Level Distributed Objects in a Flexible Testbed for Distributed AI Research
ACM SIGPLAN Workshop on Object-Based Concurrent Programming, San Diego,USA
SIGPLAN Notices Vol.24 No.4 (April 1989) pp.108-110

[GaBrHe87] Les Gasser, Carl Braganza, Nava Herman
MACE : A Flexible Testbed for Distributed AI Research
in [Hu87] pp. 119-152

[GaHu89] Les Gasser, Michael M. Huhns (eds.)
Distributed Artificial Intelligence (Volume II)
London: Pitman 1989

[GaHu89a] Les Gasser, Michael M. Huhns
Themes in Distributed Artificial Intelligence Research
in: [GaHu89] pp. VII-XV

[Gi87] Matthew L. Ginsberg
Decision Procedures
in: [Hu87] pp. 3-28

[Gr87] Peter E. Green
AF : A Framework for Real-Time Distributed Cooperative Problem Solving
in: [Hu87] pp. 153-175

[Ha85] John Haugeland
Artificial Intelligence - The very Idea
Cambridge,MA: MIT Press 1985

[He86] Joachim Hertzberg
Planerstellungs-Methoden der Künstlichen Intelligenz
in: Informatik-Spektrum (1986) 9: pp.149-161

[He89] Joachim Hertzberg
Planen
BI-Wissenschaftsverlag 1989

[He90] Joachim Hertzberg
KI-Lexikon : Planen
in: KI 1/90 p.22

[Hu87] Michael M. Huhns (ed.)
Distributed Artificial Intelligence
London: Pitman 1987

[HuBrAr90] Michael M. Huhns, David M. Bridgeland, Natraj V. Arni
Distributed Truth Maintenance
in [MC90]

[HuMuStBo87] Michael M. Huhns, Uttam Mukhopadhyay, Larry M. Stephens, Ronald D. Bonnell
DAI for Document Retrieval : The MINDS Project
in: [Hu87] pp. 249-283

[JaDo87] V. Jagannathan, Rajendra Dodhiawala
Distributed Artificial Intelligence : An Annotated Bibliography
in: [Hu87] pp. 341-390

[KaRo89] Matthew J. Katz, Jeffrey S. Rosenschein
Plans for Multiple Agents
in: [GaHu89] pp. 197-228

[KeHoHu89] J.O. Kephart, T. Hogg, B.A. Hubermann
Dynamics of Computational Ecosystems : Implications for DAI
in: [GaHu89] pp. 79-95

[KK89]
KK-Lisp-Manual
AG Siekmann, FB Informatik, University of Kaiserslautern 1989

[KoPo89] Kurt Konolige, Martha E. Pollack
Ascribing Plans to Agents (Preliminary Report)
SRI International, Menlo Park, USA 1989

[KrMa90] Thomas Kreifelts, Frank von Martial
A Negotiation Framework for Autonomous Agents
in: [MA90] pp. 169-182

[KrViWoWoi89] Thomas Kreifelts, Frank Victor, Gerd Woetzel, Michael Woitass
A Design Tool for Autonomous Group Agents
in: Proc. of the First European Conference on Computer Supported Cooperative Work
Gatwick, London, UK 13.-15.9.89

[KrWo88] Thomas Kreifelts, Gerd Woetzel
Conversational Systems: A conceptual model for off-line group support systems
in: Proc. Australian Comp. Conf. '88, Symposium 3: Computer Support for Groups,
Sydney Australia 21.-23.9.1988

[LaChRoMc89] D.M. Lane, M.J. Chantler, E.W. Robertson, A.G. McFadzean
A Distributed Problem Solving Architecture for Knowledge Based Vision
in: [GaHu89] pp. 433-462

[LeCo81] Victor R. Lesser, Daniel D. Corkill
Functionally Accurate, Cooperative Distributed Systems
in: IEEE Transactions on systems, man, and cybernetics
Vol. SMC-11 No.1 (January 1981) pp.81-96

[MA90]
Proceedings of the 2nd European Workshop on Modelizing Autonomous Agents and Multi-Agent-Worlds (MAAMAW '90); Saint Quentin en Yvelines, France 13.-16.8.1990
Organized by ONERA, BP 72, F-92322 Chatillon Cedex

[Ma90] Frank von Martial
A Conversational Model for Resolving Conflicts among Distributed Office Activities
in: COIS 90 - Conference on Office Information Systems, Cambridge-MIT
New York: ACM 1990

[Mc84] Bonnie McDaniel
Issues in Distributed Artificial Intelligence
in: International Conference on Data Engineering, Los Angeles 1984
IEEE Comp. Soc. Press 1984 pp. 293-297

[ME90] MEDLAR Group Linz
Toy Examples of Multitasking Problems
RISC-Linz, 1990

[MC90] MCC (Microelectronics and Computer Technology Corporation)
Proceedings of the 10th International Workshop on Distributed Artificial Intelligence
Bandera, Texas; October 23-27, 1990 (MCC Technical Report ACT-AI-355-90)

[Mi86] Marvin Minsky
The Society of Mind
New York : Simon & Schuster 1986

[Ne86] Jürgen Nehmer
Betriebssysteme
Script, FB Informatik, University of Kaiserslautern 1986

[Ni80] Nils J. Nilsson
Principles of Artificial Intelligence
Palo Alto: Tioga Publishing 1980

[NiAiRi89] H. Penny Nii, Nelleke Aiello, James Rice
Experiments on Cage and Poligon : Measuring the Performance of Parallel Blackboard Systems
in: [GaHu89] pp. 319-383

[Pa87] H. Van Dyke Parunak
Manufacturing Experience with the Contract Net
in: [Hu87] pp. 285-310

[Ri83] Elaine Rich
Artificial Intelligence
New York : McGraw Hill 1983

[Ri89] Michael M. Richter
Prinzipien der Künstlichen Intelligenz
Stuttgart; Teubner 1989

[RoBr89] Jeffrey S. Rosenschein, John S. Breese
Communication-Free Interaction among Rational Agents : A Probabilistic Approach
in: [GaHu89] pp. 99-118

[RoSySaFo90] Stephen F. Roth, Katia P. Sycara, Norman Sadeh, Mark Fox
Distributed Constraint-directed Search in Resource-limited Domains
in [MA90] pp. 207-217

[Sh87] Lokendra Shastri
A Connectionist Encoding of Semantic Networks
in: [Hu87] pp. 177-202

[Sr87] N.S. Sridharan
Semi-Applicative Programming : Examples of Context Free Recognizers
in: [Hu87] pp. 203-245

[St87] S. Steel
The Bread and Butter of Planning
in: Artificial Intelligence Review (1) 1987 pp. 159-189

[StGö84] Herbert Stoyan, Günter Görz
LISP - Eine Einführung in die Programmierung
Berlin,Heidelberg,New York,Tokio: Springer 1984

[Sy86a] Symbolics Inc.
User's Guide to Symbolics Computers (1)
Cambridge, MA 1986

[Sy86b] Symbolics Inc.
Programming the User Interface Volume B (7B)
Cambridge, MA 1986

[TeMo89] Moshe Tennenholtz, Yoram Moses
On Cooperation in a Multi-Entity Model (Preliminary Report)
Weizmann Institute of Science; Rehovot, Israel 1989

[We88] Eric Werner
Toward a Theory of Communication and Cooperation for Multiagent Planning
in: Theoretical Aspects of Reasoning about Knowledge : Proceedings of the 2nd Conference; Mosche Y. Vardi (ed.)
Morgan Kaufman Publishers 1988 pp. 129-143

[We89] Eric Werner
Cooperating Agents : A Unified Theory of Communication and Social Structure
in: [GaHu89] pp. 3-36

[We89] Eric Werner
A Unified View Of Information, Intention and Ability
in: [MA90] pp. 69-83

[WoKr83] Gerd Woetzel, Thomas Kreifelts
Deadlock freeness and consistency in a conversational system
in: Proc. IFIP WG 8.4 Working Conf. on Office-Information-Systems : The design process; Linz, Austria 15.-17.8.1988 B. Pernici, A.A. Verrijn-Stuart (eds.)

[YaHuSt85] Ju-Yuan David Yang, Michael N. Huhns, Larry M. Stephens
An Architecture for Control and Communications in Distributed Artificial Intelligence Systems
in: IEEE Transactions on systems, man, and cybernetics
Vol. SMC-15 No.3 (May/June 1985) pp.316-326

Appendix: Implementation

We do not go too deep into technical details. The two versions TOHSEQ and TOHPAR are implemented without language hacks or the utilization of special machine features. This should make it easy to transfer both systems to other environments. Only the aspects concerning graphic output require some caution. Here the special programming language has facilitated the implementation considerably.

(1) Details of TOHSEQ

The flow diagrams in chapter 4.4.2.2. which point out the "concrete realization" of TOHSEQ reveal most of its implementation. Because we used object-oriented programming features, especially in the paradigm of sequential message passing, the concrete realization can be regarded as corresponding to the implementation. In other words, the flow diagrams are translated straightforward onto the machine. This seem to be obvious and rather natural, but in the parallel version TOHPAR this will be no longer the case.

We have defined the tuple AGENT in a "defstruct-form" with some additional entries:

```
(defstruct (agent :named)
   name              ; identifier of each agent: integer 1,...
   place             ; actual place (tower): integer 1,...
   position          ; actual position (height) in tower: integer 1,...
   goal_place        ; final goal place: analog "place"
   goal_position     ; final goal position: analog "position"
   forbidden_place   ; place forbidden for next move [26] :analog "place"
   blocker)          ; "upper neighbour" of agent : instance of AGENT
```

TOHSEQ is started with the call of the top-level function *hanoi*::

```
(hanoi  <number of agents; integer 1,...; default 3>
        <number of places; integer 3,...; default 3>
        : from <starting place; integer 1,...,no of places: default 1>
        : to <goal place; integer 1,...,no of places: default 2>
        : heuristic <choose heuristic (cf. 4.4.3.); integer 1,...,7; default 1>
        : graphics <request graphic output; t/nil; default t>
        : statistics <request statistic evaluation; t/nil; default nil>
        : pause <"slow motion" of the moves (delay in sec.); default 0,3>)
```

[26]The heuristics of TOHSEQ mainly operate in the method *go_away_but_avoid* which decides where to go when an agent is sent off by another one. Only heuristic 7 draws wider circles (cf. chapter 4.4.3.). Because of the possible reflection and correction of an agent's intention, it must memorize where the one wanted to go which previously sent him off. The entry "forbidden_place" fulfills this need and is reset to zero when an agent really has performed an action.

(2) Details of TOHPAR

The step from the implementation level to the conceptional level in TOHPAR provides a higher abstraction level compared to the sequential version. Whereas the sending of a message in TOHSEQ really could be accomplished by a simple message passing mechanism and thus the whole paradigm was realized straightforward, we now rigorously separate "message passing" on the conceptual level and "message passing" in the implementation.

The overview in Figure A.1 gives only a coarse overview of the gap between conception and implementation of the parallel scenario.

	Conceptional View	Implementation	Comments
agents	Instantiated tupel AGENT	Instantiated defstruct object	Similar to TOHSEQ
messa-ges	Information from other agents or from an external mediator as entries in private agendas or public blackboards	messages: lists with a special format agendas, blackboards: lists of messages message passing: appending messages to lists	No isomorphism between message passing and calling a method like in TOHSEQ
me-thods	Units of scheduling the scenario, granting the chance to do some-thing. Strictly spoken, transparent at the conceptual level	Realized by the LISP features *defmethod* and *send*. Methods may call further submethods	On the conceptual level the term *method* is unknown
acting, schedu-ling	No limitation to parallelism. Each agent can try to act each time	Fixed scheduling algorithms settled by the framework of the call-structure of the methods.	The interferences of parallelism are simulated as thoroughly as possible

Figure A.1 : Some aspects of TOHPAR

The agents - conceptionally regarded as mathematical tuples - are implemented like follows:

```
(defstruct (agent :named)
   name              ; identifier of each agent: integer 1,...
   place             ; momentary place (tower): integer 1,...
   position          ; momentary position (height) in tower: integer 1,...
   blocker           ; "upper neighbour" of agent : instance of class AGENT
   agenda            ; list of next goals, alternatives, orders to flee,...
   act-flag          ; indicates inner status: t/nil (explanation see above)
   conflict-flag     ; analog to act-flag
   wait-flag         ; analog to act-flag
   go_away-flag)     ; analog to act-flag
```

```
(defstruct (info-agent :named)
   blackboard        ; list of entries in a fixed format, information for all agents
   number_of_agents  ; parameter from the function call
   number_of_places  ; parameter from the function call
   agentarray        ; array 0..number_of_agents - 1 of agent
   placearray        ; array 0..number_of_places - 1 of list (of agent)
   actarray)         ; array 0..number_of_places - 1 of list (of list)
```

The top-level function "hanoi-parallel" is called by the user with the following arguments:

```
(hanoi-parallel
        <number of agents; integer 1,...; default 3>
        <number of places; integer 3,...; default 3>
        : from <starting place; integer 1,...,no of places: default 1>
        : to <goal place; integer 1,...,no of places: default 2>
        : statistics <request statistic evaluation; t/nil; default nil>
        : pause <"slow motion" of the moves (delay in sec.); default 0,3>)
```

The graphic display is obligatory, heuristics cannot be chosen any longer - the best will always be employed.

(3) Parallelism is Simulated in TOHPAR

An ideal parallel scenario (cf. Figure A.2) with no restrictions behaves completely asynchronous and has no global states. Temporal aspects are fundamental. To model and to formalize this likely chaotic behaviour would be a hard task, even on machines which offer massive parallelism.

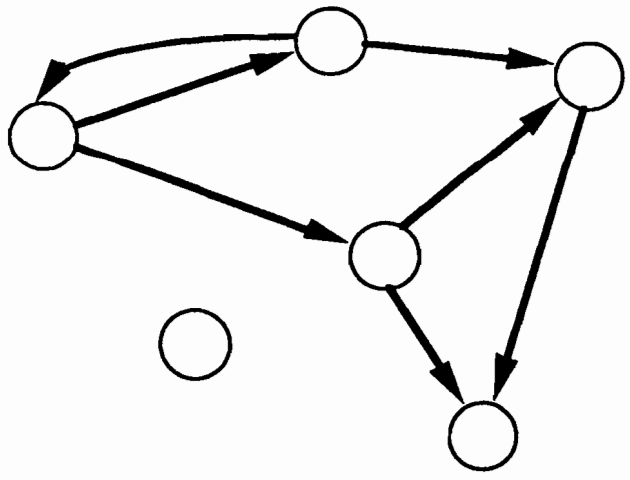

Figure A.2 : Parallel Exchange of Information

In TOHPAR we are mainly constrained by a one-processor machine, so we must simulate the parallelism. This "pseudo-parallelism" is achieved under several strict presumptions.

Perfect synchronism

Different agents which perform the same part of action need exactly the same amount of time. Agents which do not act in the moment delay themselves for the time the others act. This implies that it is impossible that one agent overtakes an other.

A mediator

The information agent Big-Brother recognizes possible clashes when the agents after all try to act. It synchronizes and takes care for the maximum amount of parallelism.

Fixed scheduling for all agents

This ensures a very limited interaction and interference between the agents. When agents do not get in conflicts with their actions, the order of acting is not substantial.

Two aspects of parallelism in the scenario shall be investigated. The "fundamental parallelism" and the visible parallel performing of moves.

The fundamental parallelism concerns the whole proceeding. The point here is the simulation of an almost parallel behaviour in all steps of problem solving. We achieve this by cutting up the scenario-model into the modules presented in chapter 4.5.2. Each of these modules is executed by each agent in the order Agent '1', Agent '2', ..., Agent 'n'. This can be regarded as a special kind of a fixed "round robin" scheduling [Ne86]. For example: After Big-Brother has initialized the scenario, Agent '1' creates its new goal, then Agent '2', then Agent '3' and so forth. After Agent 'n' has created its new goal, the next module (*inform_other_agents*) is invoked from Agent '1' again.[27] These real sequential

[27]Incidentally, this provides global states: From the parallel point of view the module the system is in for the moment represents the state, or seen from the real implementation, which of the agents is just operating.

sub-actions of the agents are regarded and conceptualized as completely parallel. All agents create their new goals, all agents inform the others about their new intentions,...
But to make this approach sound, the sequential evaluation may by no means produce results different to those which would have been produced by parallel processing. How can we achieve this demand ?
The modules were formed in such a way that a strict separation between internal and external processing is promoted. This changes turn in turn, if one module is processed internally by all agents, the next will be processed externally.
Figure A.3 shows this symbolized. At the right margin the respective modules are denoted, the graphics depict the abstract ToH world. Bold arrows indicate how the modules are possibly traversed. The exchange of information (message passing) is represented by thin arrows. If an arrow points to the same agent where it started, it works only "internal". Arrows pointing to the blackboard symbolize "writing" and, vice versa, those which start at the blackboard mean "reading".
"Internal" means entire working on the agent's own agenda. No messages are posted to other agents, nothing causes changing influence from the environment. It is quite obvious that a sequential internal processing must produce the same results as a theoretical parallel one.
When agents proceed externally, things become harder. They send messages to other agents. The order (and thus the sequence) does not matter, because the agents receive all what comes in without evaluating it. Evaluation follows later in the next internal step. They write messages on the blackboard. Also the order is not substantial, sequential processing produces the same results as parallelism.
Hence the strict separation of external proceeding is the foundation of the simulation of parallel behaviour.
At a first glance all problems concerning the simulation of parallelism seem to be solved. But a closer look to the negotiation cycle (especially *conflict-resolution-1*) reveals certain problems:

- When agents change their decision, the time they do it can be very important for other agents which depend on these agents in any way.
- When agents ask for help or generate new orders to cause others to flee, the order may influence succeeding agents in the sequence.

These influences and effects must be excluded! The strict separation of internal and external proceeding is a solid base, but not sufficient. Additionally, certain testing routines must be built into the procedural knowledge of the agents. For instance, it must be checked whether agents, which are concerned by a changed decision eventually must be torn back into negotiations again. Or it must be ensured that really all agents recognize all changes, thus certain routines are evaluated several times.

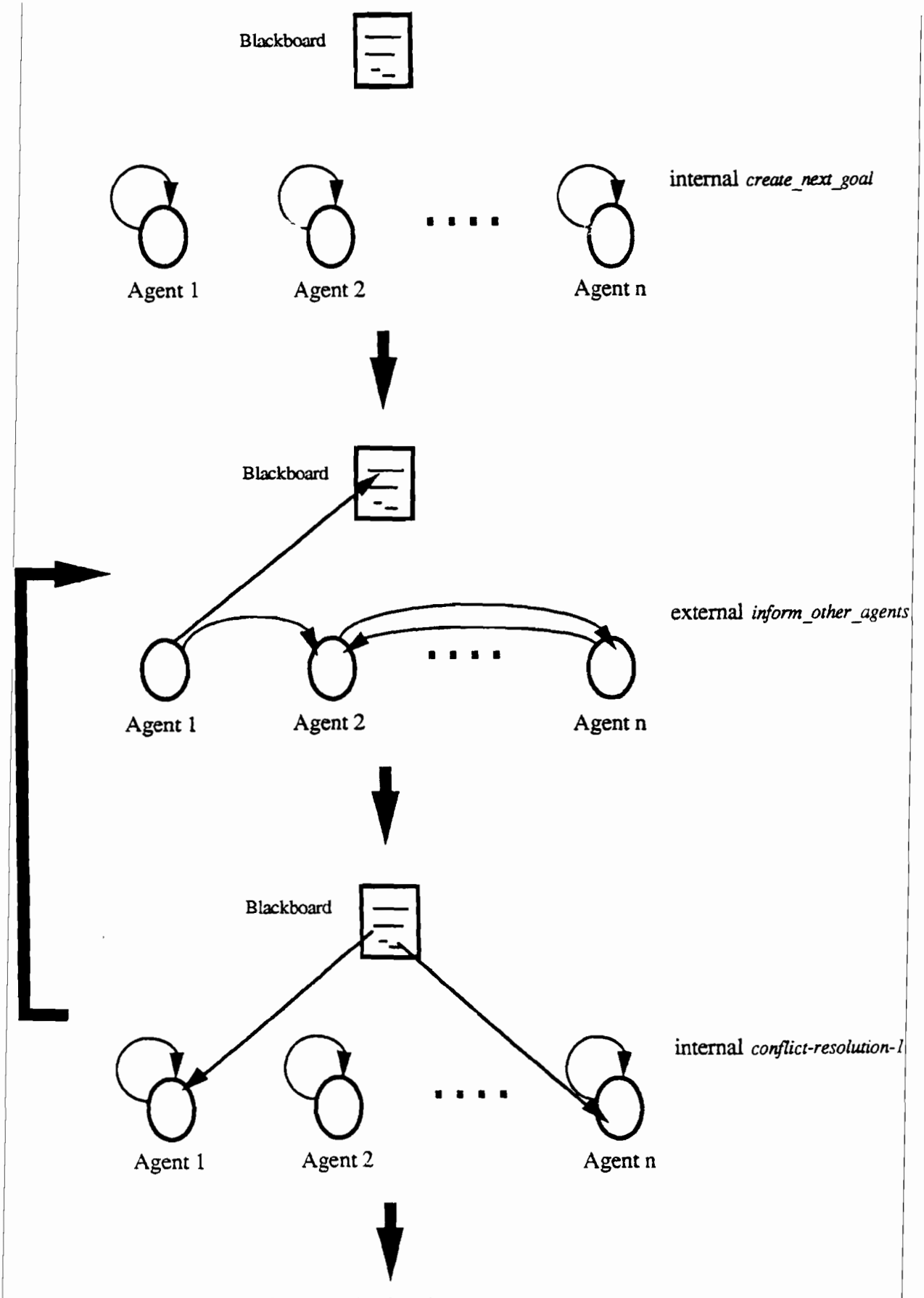

Figure A.3 : The exchange between internal and external behaviour of the agents

The other aspect of parallelism comprises the visible simultaneous acting (moving) of the agents. This is the part of TOHPAR where the user is directly confronted with parallelism. Here the module *graphics* is, strictly spoken, cut up further. If every agent, which is in the set of the joint actors would proceed *graphics* solely, a illusion of parallelism could not be constructed. We use another way: *graphics* is called with a list of agent-names and for each element of the list, a substep is performed and displayed:

1. All agents (disks) in the list are lifted (really: one is lifted after the next).
2. All agents of the list move to their new places (again, really one after ...) and analogously for the move down. The speed of computation evokes the illusion of parallelism.

For further information about graphics see chapter (6) in this appendix.

(4) The Implementation Language

TOHPAR and TOHSEQ are implemented in KK-LISP[28] [KK89], which is mainly based on Common LISP.

KK-LISP consists of the Common LISP kernel and some additional features like module oriented trace and check facilities. Features which cannot be easily implemented in other LISP dialects are intentionally excluded. Especially things like coroutines and multitasking are not supported. This is a main drawback when implementing parallel scenarios. Therefore parallelism must be simulated in KK-LISP. On the other hand portability to other LISP dialects or even other languages increases.

KK-LISP recognizes special declaration in both interpreted and compiled code and eliminates thus this potential source of incompatibility. Interpreted programs return identical values with respect to their compiled equivalents. KK-LISP consists of a rich amount of program and control constructs to assist the writing of a compact and readable code.

Some differences to Common LISP are:

- a more powerful, but Common LISP incompatible defstruct form.
- the window system
- additional iteration functions
- additional functions in the field of list processing

The efficiency of compiled KK-LISP code is only restricted by the underlying LISP implementation.

[28] KK-LISP is an acronym for "Karlsruhe & Kaiserslautern LISP".

(5) Machines

The two versions of the ToH scenario were implemented on a Symbolics 3640 workstation [Sy86a]. This is a LISP machine with some special architecture features:

- tagged architecture
- multiple caches
- hardware stack management
- pipelined instruction cycles
- parallel processing (but no parallel LISP features, unfortunately)
- hardware assisted garbage collection

The Symbolics software environment is called "Genera" and provides comfortable programming tools like mouse-sensitive parts of the screen, menus, and a window system. Several windows for special use are predefined. There are, for instance, editor, file system, LISP interpreter, terminal emulation, debugging tools, and a mail system. All may be arbitrarily selected and left.

(6) Graphics

We exploit the features of the window system of the Symbolics software environment Genera 7.1 [Sy86b].
The function `tv:make-window` to generate the static window is in the package "tv". To invoke the graphic output, several messages can be send to the created window object. From KK-LISP the function `sys:send` therefore must be employed, `send` isolated works only for defstruct objects. Following methods of the window-object are involved:

`:draw-rectangle`	: lets the "agents" appear or vanish
`:draw-line`	: draws the ground and the places
`:set-cursorpos`	: output of textual information
`:clear-rest-of-line`	: dto.
`:string-out`	: dto.

The principle of the graphical output is like follows:

1.) Initialization

Draw the agents at their start positions, draw the ground and the "places" (sticks).

2.) Parallel move

a. - let all moving agents vanish on their old place
- draw all moving agents "in the air" just above their old place
- pause

b. - let all moving agents vanish
- draw all moving agents "in the air" just above their goal place
- pause

c. - let all moving agents vanish
- draw all moving agents on their goals
- ready

The three substeps of a., b., and c. must be performed very fast to give the illusion of movement and parallelism.

What parameters do influence the shape of the output-window ? First, there are static parameters implemented as global values. They determine the outer form (size and position) of the window and will never be changed (The values denote the number of pixels).

height_of_window	700	--> size of window
breadth_of_window	1000	--> dto.
upper_margin	300	--> partition of window
lower_margin	75	--> dto.

Dynamic parameters are computed at run time because they depend on the number of agents and places (Their values are numbers of pixels, too). They take care for a convenient pictorial representation of the output window, e.g. the ratio breadth/height of the "agents" must be tolerable. Furthermore, the exact positions of each agent have to be computed pretty fast.

distance_of_places	--> the gap between the sticks
height_of_agents	--> self-explaining
dif_of_breadth	--> difference in length between adjacent agents
min_breadth_of_agents	--> self-explaining, arbitrary fix value (here: 10)

The *breadth* (also in pixels) of each agent is stored in a slot of the class AGENT having the same name.

The dynamic parameters are computed as follows:

distance_of_places = [*breadth_of_window*/no_of_places]

height_of_agents = [(*height_of_window* - *upper_margin* - *lower_margin*)/ no_of_agents]

dif_of_breadth = [(distance_of_places - min_breadth_of_agents) / (no_of_agents - 1)]