Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
RR-91-09

RATMAN and its Relation to Other Multi-Agent Testbeds

Hans-Jürgen Bürckert
Jürgen Müller
Achim Schupeta

March 1991

Deutsches Forschungszentrum für Künstliche Intelligenz
GmbH

Postfach 20 80
D-6750 Kaiserslautern, FRG
Tel.: (+49 631) 205-3211/13
Fax: (+49 631) 205-3210

Stuhlsatzenhausweg 3
D-6600 Saarbrücken 11, FRG
Tel.: (+49 681) 302-5252
Fax: (+49 681) 302-5341

RATMAN and its Relation to Other Multi-Agent Testbeds

Hans-Jürgen Bürckert, Jürgen Müller, Achim Schupeta

DFKI-RR-91-09

Parts of this work are/will be published under the following references:

Bürckert, H.-J., Müller, J.: RATMAN: Rational Agents Testbed for Multi Agent Networks, in Demazeau,Y., Muller, J.-P.: Decentralized Artificial Intelligence, Proc. of the second workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'90), Elsevier Sc. Pub/North Holland, 1991 (forthcoming)

Müller, J.: Defining Rational Agents by Using Hierarchical Structured Knowledge Bases, IEE Computing and Control Division Colloquium on “Intelligent Agents”, Digest No. 1991/048, London, Feb. 91

RATMAN AND ITS RELATION TO OTHER MULTI-AGENT TESTBEDS

Hans-Jürgen BÜRCKERT, Jürgen MÜLLER, Achim SCHUPETA

DFKI (German Research Center on Artificial Intelligence)
Research group WINO
Postfach 2080
D-6750 KAISERSLAUTERN
GERMANY

RATMAN (Rational Agents Testbed for Multi Agent Networks) is a workbench for the definition and testing of *rational* agents in multi-agent environments. The special feature of RATMAN is the specification of such agents with hierarchical knowledge bases comprising all knowledge levels from sensoric knowledge to learning capabilities. In all levels only knowledge representation languages have to be used which are based on logic. On each knowledge level the designer may choose the granularity of knowledge for the agent to be designed and moreover he may decide whether the agent should have a certain skill at all. Thus it will be possible to construct a society of very heterogeneous agents from expert systems on one side of the spectrum to simple reactive agents on the other side. Since the aim of such a testbed is to get more insight in the behavior of intelligent agents' cooperating actions, RATMAN is providing a set of statistical and documentational features. In the second part other approaches to multi-agents environments are presented. AF, MACE, AGORA and MAGES are first characterized by their main features. Then their specialities are discussed and finally the boarders with respect to RATMAN are considered.

Contents

1. Introduction

The idea of modeling a world of autonomous cooperating agents grew rapidly in the last years [BG88], [DM90], [Huh87], [Huh90], [DM91]. What do we mean, when we talk about autonomous cooperating agents? The terminology in this area is not as clear as it should be. Speaking of an autonomous agent some authors think of a kind of robot, such stressing on motion and action skills, while others have in mind an expert system, which cooperates with other systems of different expertise, such stressing on the reasoning skills. Bond and Gasser [BG88] speak of an agent as a computational process with a single locus of control and/or "intention", but concede that this could not be treated as a definition. They also sketch the position of multi-agent (MA) systems within AI as follows:

Distributed artificial intelligence (DAI) is the subfield of AI, that is concerned with concurrency in AI at many levels. DAI branches into the area of distributed problem solving (DPS), which considers how the work of solving a particular problem can be divided among a number of cooperating and knowledge-sharing modules or nodes, on the one hand and into MA systems on the other. In MA systems the coordination of intelligent behavior of a collection of autonomous intelligent agents is the main concern. A third area - parallel artificial intelligence (PAI) - stresses more on performance problems than on conceptual advances and is involved in the development of parallel computer architecture, parallel languages and algorithms. But a sharp distinction between these areas could not be drawn as there is no clear commonly accepted definition of "autonomous cooperating agents".

From another perspective, two other major streams, two paradigms of viewing multi-agent systems can be identified in the literature, namely the behavior-based approach and the knowledge-based approach. The central idea of the behavior-based approach is that the agents react in response to environmental changes, where in the knowledge-based approach they act as a consequence of their reasoning about their goals and intentions.

In the context of behavior-based systems Connah, Shiels, and Wavish [CSW88a] describe the structure of artificial agents from a cognitive point of view. They develop an integrated architecture, where abstract cognitive activities emerge from a concrete, situated activity. In a continuation [CSW88b] the model of a testbed for cooperating agents is described in five big parts: Nature of the world, representation of position, shape, time, and causality. Steels [Ste89b] tackles the problem of cooperation between distributed agents. A behavior-based approach is used to examplify self-organization for establishing emergent functionality. The interesting point of this approach is that neither explicit, complex representation of the world nor explicit communication between the agents is used. A very similar approach is also studied by Moyson and Manderick [MM88] who describe the collective behavior of ants with the aim of simulating the emergence of self-organization by an intrinsically parallel algorithm. Further they propose a mathematical model of the behavior, which is used to compute the parameters in the simulation. Finally Maes [Mae89] gives an interesting approach to model the activation/inhibition of agents in an emergent, noncommunication agent environment. She represents the preconditions of activations as formulas. An agent becomes active if the precondition of a goal is proved to be valid.

For the knowledge-based approach essential ideas may be found in [Coe88] where an introduction to the interaction of multi-agent systems in terms of actual questions from the rational agent point of view is given. The special focus of the discussion is communication between agents. Tennenholtz and Moses [TM89] present a theoretical foundation for a multi-agent planning environment. The set of goals of different agents defines the "cooperative goal of the system". The abstract "cooperative goal achievement decision problem" is then the question whether each of the goals can be fulfilled by the agent society. Myerson [Mye88] discusses the communication activities between agents with different goals from the game theoretic point of view. By combining the "incentive constraints" of the partners an optimal plan between the different goals may be computed. Really logic based is the development of Mazer [Maz88]. He studies commitment problems in distributed environments. The approach is based on "knowledge logic", a temporal modal logic to describe the interaction of processes, which can execute events, via a communication system. Communication abilities, negotiation and planning for multi-agent systems of this category are reported in [KP89], [Wer88a], [Wer88b], [ZR89].

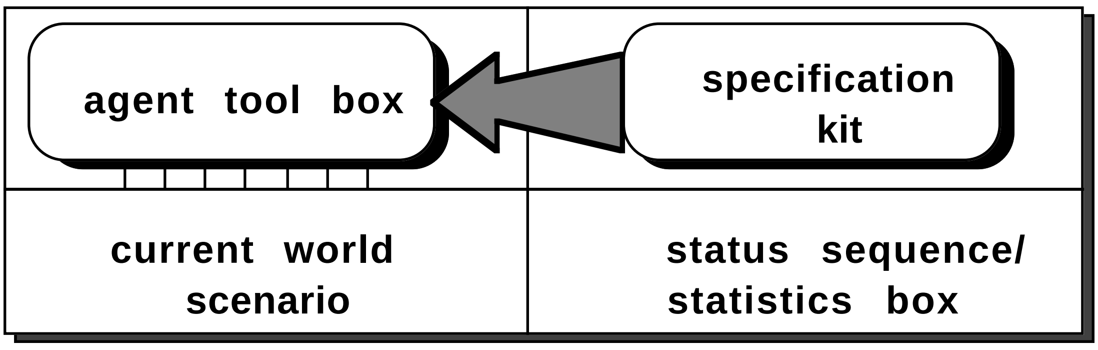
As this short and surely incomplete essay shows there is a great variety, a large spectrum of approaches and special features to describe agents and their behavior in a multi-agent society. The basic point, the salient research issue in multi-agent scenarios is to learn about the interaction of heterogeneous agents. The question is, what will happen if we give a task to a set of different agents in an multi-agent scenario. If the "social goal", as Werner [Wer88b] said, is reached by the agents, how did the agents succeed? If we change the skills of some agents, will it work again? Is there a higher principle about the behavior, no matter whether the agents can communicate or not, whether they are "intelligent" or not?

To study those questions we need tools to work with. Toolboxes, workbenches, testbeds, experimental environments to define single agents, to collect them in a society, to build a world for them and to give them a common goal to achieve. This is the main lack of the current research in this area! Many research projects envision some concepts, that can stimulate the multi-agent research, develop a scenario, which explains their ideas and shows the intended results and implement it (if at all) with a program, that is perfectly tailored to this scenario. These kinds of work usually lay the basis for further research, often leading to new and different points of view (paradigm invention). A few research projects try to define an architecture or a kind of framework for agents, that fall into a certain category or paradigm. They also present sample-scenarios, but their work is capable of being relevant to a whole class of problems (paradigm extension). To capture all the variety of approaches within one flexible universal testbed would lead to a unification of different paradigms as well as to a support of the invention and extension of new paradigms - at least as far as it reduces and accelerates their implementational work. MACE [GBH86], Agora [BAFLB87], AF [Gre87] and MAGES [BFS90] are first approaches to such a system and we will describe some aspects of them in comparison to our suggested system RATMAN after the presentation of its structure.

The main characteristic of RATMAN will be that it is purely based on logic, that is, it is possible within the logic paradigm to specify all individual features for simplest up to most complex agents, that the user may be one agent in the society and that there are various facilities that support the analysis of a session.

2. RATMAN: The Overall Structure

In the following figure we show the overall structure of the Rational Agents Testbed for Multi Agents Networks. RATMAN consists of four main modules: the agent tool kit, the specification kit, the current world scenario and the status sequence/statistics box.



The *specification kit* serves as the user interface to define the agents, their relations in the world, and their status in the agent society. It will provide several choices for general strategies to be performed and it can be used to specify what kind of status information and statistic data should be monitored by the system.

The *agent tool box* provides a scheme for a hierarchically structured knowledge base together with reasoning facilities to model all features of agents. Predefined knowledge may be used or be partially skipped and new knowledge may be introduced by the user.

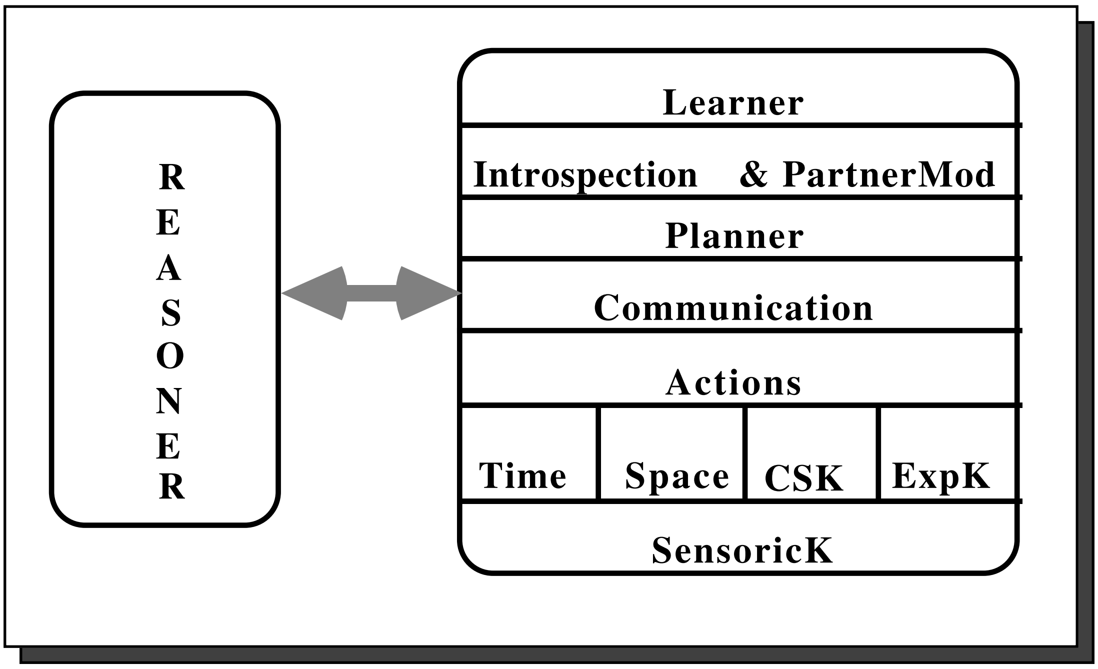
Each agent defined in the agent tool box will get a place in the *current world scenario*. Together with objects of the worlds the agent society is monitored to the user. Further this module provides a blackboard which serves as the communication platform for the agents and the user. Thus, if an agent is defined to have communication facilities, then he has access to the blackboard or parts of it. And the user itself may be seen as one of the agents.

The *status sequence and statistics box* has the task to show the sequence of the changing worlds, the activity potential of the agents, the internal clock, analysis of synchronization processes etc.

We will now unfold the overall structure to discuss the components in more detail. First we will describe what will be predefined in the agent tool box and then we will look at the features of the current world scenario. In order to demonstrate that and how both the agents' knowledge and skills and the world can be formulated in RATMAN with logics we provide a couple of examples in *logic based* description languages. The realization of special agents and societies will thereafter be discussed in the description of the specification kit. Finally some words have to be said concerning the status sequence and statistics box.

2.1 The Agent Tool Box

An agent is defined by his knowledge and his facilities, i.e. his knowledge base. Since both, knowledge and skills, should be represented in the knowledge base, it is organized hierarchically. And moreover, since different knowledge representation languages force a lot of trouble in the connection of the levels, it is demanded that - at least - the representation can be translated into a single paradigm, namely logic. The following picture shows the different levels of the hierarchical knowledge base.



We will first for each level give examples of the knowledge to be presented. Then we discuss the functional aspects of the hierarchical structure.

2.1.1 The Knowledge Levels

On the lowest level the *sensoric knowledge*, the information about the status of hands, feet, eyes, etc. are represented. This knowledge is basically propositional in nature, like:

```
RIGHT_HAND (empty )
LEFT_HAND (objekt-A )
BODY_POSITION (standing)
WORLD_MONITOR (accessible)
CURRENT_STATE(waiting)
```

The *second level* is the one which represents the knowledge base in the usual sense. It is divided into four parts: Knowledge about time, space, common sense and special expertise. The time slot holds general information like the handling of time intervals as defined by Allen [All84], definitions of day, week, minutes, hours etc., as well as special timing information which is valid in the specific world to be modelled.

```
TIME_POINT (X~Y)
HOUR (X~Y,X)
MINUTES (X~Y,Y)
if TIME_POINT (X) and TIME_POINT (Y)
  and BEFORE(X,Y)
then INTERVAL(X<~>Y)
MEETS (X<~>Y,Y<~>Z)
STANDARDISE ('morning', 06~00<~>12~00)
```

In the *space* part there are general and specific spatial information of two kinds, namely space knowledge of the world and spatial relevant information of the objects. The knowledge is given in terms of coordinates as well as in standardized space prepositions.

```
all X all Y [OVER(X,Y) iff (ON(X,Y) or
                    (exist Z (ON(Z,Y) and OVER(X,Z))))]
not [OVER (X,X)]
not [OVER (X,Y) iff OVER (Y,X)]
[OVER (X,Y) and OVER (Y,Z)] impl OVER (X,Z)
OVER_COORD (X,Y) iff y2 (X) ≤ y1 (Y) and
                    x1 (X) ≤ x2 (Y) and x2 (X) > x1 (Y) and
                    z1 (X) ≤ z2 (Y) and z2 (X) > z1 (Y)
POSITION (object-A, [1,2,3,4,5,6])
```

The *common sense* knowledge base holds general information about the world and special knowledge about the facilities, for instance:

```
if BODY_POSITION(sitting) then CURRENT_STATE(waiting)
if ON(X, board) then
        REACHABLE(X) iff BODY_POSITION(standing)
if CARRY(X, table) then NEED(X, help)
not CAN_MOVE(X : Block)
```

The *expert* knowledge might for example contain information about transport tasks, i.e., knowledge about several kinds of transport vehicles and transport connections of a forwarding agency: Transport vehicles are vehicles that can transport some goods, ships are defined as transport vehicles that can drive on the water, and of course all ships can carry all kinds of goods from Hamburg to London.[1]

```
Transport_vehicle := Vehicle ⊓ ∃ transport: Goods
Ship := Transport_vehicle ⊓ drives: Water
all X:Ship all Y:Goods CAN_CARRY(X, Y, hamburg, london)
```

As in the former knowledge bases general and specific knowledge is provided to define the agent, in the *action* knowledge base information about actions that are possible to perform are as well given as actions which can really be done by the agent. E.g. a child knows that there is an action DRIVING_A_CAR, which is only used for explaining the car driving action of an adult. On the other hand the action DRIVING_A_BIKE can be realized by the kid. To define actions the definitions of the lower knowledge base levels are used extensively.

```
STACK (X, Y, T)   if    (RIGHT_HAND (X) or LEFT_HAND (X)) at T-1
                        and REACHABLE(Y) at T-1
                        and (not (exist Z (ON (Z, X))) at T-1)
                        impl ON(X, Y) at T
MOVE (X, Y, T)   if     OWN_POSITION (X) at T-1
                        and not (exist Z (POSITION(Z, Y) at T-1)
                        impl OWN_POSITION (Y) at T
```

On the *communication* level various possibilities are provided. Simple communication will use bit vectors. The meaning of the bits will be predefined (by the user) and actions will be performed along the status of the bits.

```
CO_VEC(help_me, like_to_help, wait, first_me, give_me, stack)
if  CO_VEC(0,0,0,0,0,0,1) then STACK(A, B, NOW)
```

On a higher level a dictionary of key words is stored and the agents communicate by sending and receiving single words. The highest level is the generation and analysis of simple sentences in a predefined subset of natural language. It is intended to use a kind of Montague grammar and syntax [Tha89] which can be compiled into first order logics. Thus a sentence like

"STACK THE RED BLOCK ON THE TOWER"

will result in

```
STACK(object-1, [object-2,object-3], NOW)
```

after identifying red block with object-1 and the tower with the list [object-2, object-3].

From a more abstract view-point something like an "agent being within earshot of another agent" could be modelled by write access for the second agent to the blackboard part corresponding to the first one:

```
WITHIN_EARSHOT(X, Y) iff WRITE_ACCESS(Y, X)
```

[1] We use a concept language of the KL-ONE family to express knowledge about concepts as ships or trucks, which is known to be a subset of predicate logics (cf. [Neb90]) and can easily be combined with predicate logics via a constrained based approach [Bür90].

The *planner* part of an agent contains a couple of plans that can be combined and executed in order to solve any current tasks.

```
TRANSPORT(table) if exist X, Y exist T
                        MOVE(X, position(table)+1, T) and
                        MOVE(Y, position(table)-1, T)
if NEED(X, help) and exist Y WITHIN_EARSHOT(Y, X) then CALL(X, Y)
```

In the meta knowledge base used for *introspection* and *partner modelling* the agent's knowledge about his own knowledge and his partners' knowledge is available. Both are not copies of the own or the partners' knowledge bases, but models of these knowledge bases.[1] For instance, the agent might know about the structure of his knowledge base as presented above, i.e., he knows that he has knowledge about time and space, but perhaps does not know anything about the realization of this knowledge. He might know that he has an understanding of the spatial relation OVER, but he knows nothing about its realization via OVER_COORD.

```
I_KNOW(OVER(X, Y)) if I_KNOW(ON(X, Y))
I_KNOW(not ON(X, table) if I_KNOW(RIGHT_HAND(X))
```

The partner models might contain believes about facts that are known by all agents or by specific agents, for instance, an agent beliefs, that others know, that he needs help when he is calling them or that agents_1 is the expert for transportation by ships:

```
KNOWS(X, NEED(I, help)) if CALL(I, X)
all X:ShipTransport KNOWS(agent_1, X)
```

Finally there is the *learner* component – also a meta knowledge base – where all the agent's knowledge about his learning facilities is stored. Of course this heavily to relies on the introspection component, since the agent needs to know whether he already knows the "new" knowledge pieces or what he knows about these new things he is intended to learn.[2] This could for instance be a simple addition of new knowledge pieces after an inspection of the knowledge base. Which might just be checking if the agent does not know about this knowledge and that it is consistent with the existing knowledge.

```
ADD_TO_KB(X, ExpK) if INSPECT_POS (X, ExpK)
                        and EXPERT_KNOWLEDGE(X)
INSPECT_POS(X, Y:KnowledgeBase) if not I_KNOW(X)
                        and CONSISTENT(X ∪ Y)
```

[1] A way to avoid logical omniscience [Hin75], [Had88] in modelling beliefs could be to allow only derivations of a fixed depth or with certain resource restrictions. A good overview about logics of knowledge and beliefs can be found in [McA88], see also [Per85], [Per88].

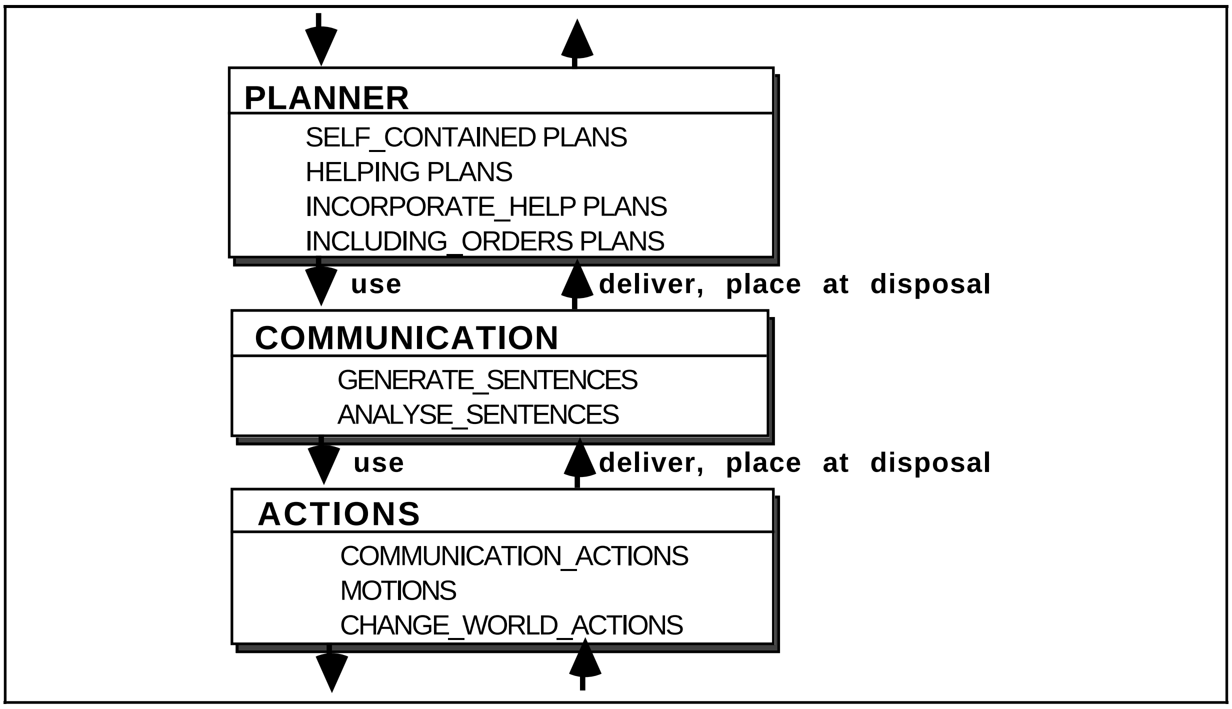
[2] Logical approaches to machine learning are, for instance, reviewed in [Kod86], see also [Mor89].

2.1.2 Functional Aspects

An Agent is in general determined by his knowledge, which also summarizes his skills and abilities. For this reason it is convenient that an agent is defined by means of a knowledge base. Since such a KB must be huge, it must be structured according to the various kinds of knowledge embedded in it. We use a hierarchical structure, where simple, propositional knowledge is stored on the lowest level and knowledge about knowledge, for instance used for introspection and learning, will be found at the top levels. The general principle underlying the structure is that a higher level uses concepts, which are defined in detail at a lower level. E.g. at the planning level a predicate ASK_FOR_HELP may be used. This predicate invokes the generation of an information block I and the activation of a predicate ASK&WAIT(I) on the (lower) communication level. ASK&WAIT(I) is then given to the action level where it is decomposed into send and receive actions, which need information about the communication status from the sensoric level as well as knowledge about time for their schedule.

The other way around, the level i places the defined predicates at disposal to the level i+1 and it delivers positive or negative answers to the asking higher level. In our example, if the communication channels are open (the agent can speak and listen), the send/receive actions are performed and the answers are given to the communication level, where they are analyzed. If there is a positive answer from another agent this information is passed to the planner level, where the ASK_FOR_HELP predicate becomes true and the respective arguments are instanciated by the necessary information (e.g. who could help).

The levels themselves are structured as modules to realize different agents by choosing different modules at the levels. If at the lowest level only propositions reflecting uni-directional communication channels are chosen instead of bi-directional ones, then the corresponding agent can only “speak” or “listen”, but not both.



For another example look at the planner level: If only the module SELF_CONTAINED PLANS is chosen, then no cooperation will take place, because the agent tries to fulfill his tasks without the incorporation of other agent in his plans. As a consequence, if a module is skipped on a lower level, modules on a higher level may be skipped automatically if they use predicates from the skipped one. Further it is possible to define a general setting, e.g. that each agent is able to communicate with all others. Then for example the COMMUNICATION_ACTIONS are part of each agents' action level and cannot be removed by the designer.

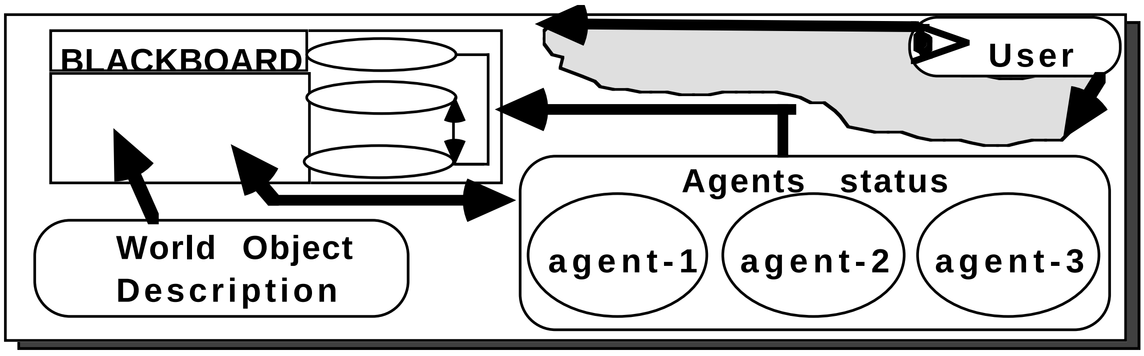
2.2 The Current World Scenario

The current world scenario window fulfills several different tasks: first of all it monitors the status of the actual world. On a blackboard information (location, current properties, relationships, etc.) about objects in the world are given. Also the locations and relations of the agents are represented on the world window of the blackboard. The second part of the blackboard is organized for the communication of the agents. Each agent will have a special slot to write at and will have reading access to some of the other slots according to his specified communication skills (see specification kit).

The current world window also shows some of the important activities of the agents in the *agents status module*. It is shown whether the agent is currently active or not, the actions to be performed next, the communication acts, what plans are focused etc. For example an agents' status might be:

```
agent-i:
status: waiting
action: if ON_THE_FLOOR(object-1) and AGREEMENT(agent-1)
              then PICK_UP(object-1)
communicate: READ(blackboard(com (agent-1))
current_plan:   1. LISTEN_TO(agent-1)
                2. if REALIZABLE(action) then PERFORM(action)
                                        else LISTEN_TO(agent-1)
```

In the knowledge base for the general description of the *worlds objects* (which is not completely visible to the user of course) the objects of the world are represented. Here again concept languages of the KL-ONE family [BS85], [Neb90] or a FRAME-based languages can be used [Hay79], [Min75]. These languages can be interpreted as sublanguages of first order predicate calculus and thus fit best in our basic paradigm to use logic within the whole system.



The *user* has access to the whole scenario. Thus the user may just have the role of an observer but he has also the possibility to act like a “god” who directly manipulates the world. Or, and this is the most interesting case, he may act as an agent in the world. E.g. he has his own communication slot on the blackboard and an agent slot in the agent status module.

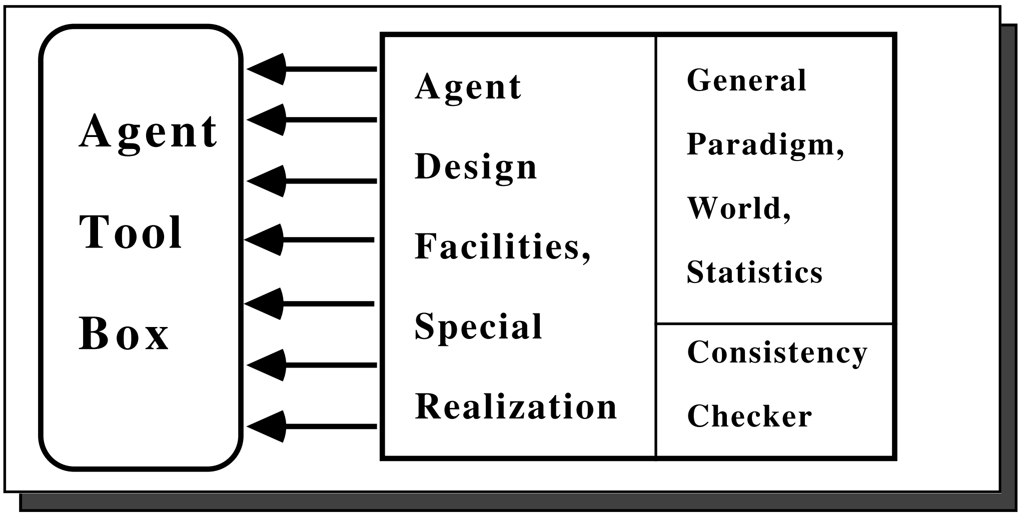
The current world scenario serves not only to give information of the actual status to the user, but it may be seen also as “the eyes and ears” of the agents. The agents may have access to at least parts of the scenario. So they “see” where objects and other agents are in the world via the blackboard. They may “hear” what is said by the other agents if they have access to the others communication slots. And even more they may get some knowledge (intuition) about the “mental stage” of the others by accessing the information of the agents status module.

Now, since the basic features of the agents and the world they live in are described, we will come to the specification kit, which is used to modify the predefined agents and world model.

2.3 The Specification Kit

The high level interface of the system is given through the specification kit. It is used to define general features of the system and the agent society and it serves to generate the different single agents. Further it has the task to check the consistency of the definitions. Thus, the specification kit is divided into three parts: The agent design facilities, the general paradigm module, and the consistency checker.

First of all the *general paradigm module* holds general information about the agent-society, like the number of agents in the system, whether their organization is anarchic or hierarchic [WHBSS81], whether there are agents with special responsibilities and more general if the society is homogeneous, clustered or inhomogeneous. A benevolence assumption [GGR86], [RG85] can be defined, e.g. whether the society tries to achieve a common social goal [Wer88a] or whether the agents are egoistic or just cooperative by self-interest.



Concerning the tasks given to the society it is to be defined whether it is one task which is to be solved by the agents without any help [SD83] or whether the designer himself will divide it into subtasks or even more whether the global task is not given to the agents, but each agent is given a specific task which then leads to a synergy effect [BG88], [Len75]. The question then is how will the agents tackle the tasks: by negotiation through communication, in performing a one-way cooperation or a mutual cooperation, in working in a master-slave relationship or in no predefined style [Wer88a]. For the interaction part the communication will play an important role in the all over design. It has to be specified whether there should be communication at all and if the agents can communicate, then, whether the communication will be primitive (setting and resetting bits in a specified vector) or high level linguistic [Wer88a]. Should it be plan and information passing in the internal language (parts of the knowledge base) or short sentences in a subclass of natural language? What should the effect of the communication and other information acquisition processes be? Just the addition of new knowledge into the knowledge base, increment or decrement of the activation level [Mae89], pragmatic effects on the internal state of the agent, dying out of inconsistent solutions by exchange of partial results or plans [LC81] or what so ever.

Finally the statistics box of RATMAN is initialized through the general paradigm module (see below).

While in the general paradigm module general decisions are specified, the *agent design facility* box is used to enable and disable parts of the agents knowledge base to simulate the design decisions made in general and to realize special agents in adding specific knowledge (e.g. filling the expert KB) or in specifying the communication channels. To each level of the agents knowledge base a level in the agent design facility box is associated to manipulate the knowledge base.

If we like to simulate a homogeneous society of dumb individuals in the sense of Maes [Mae89] and Steels [Ste89a] only the sensoric and the action parts of the knowledge base are to be enabled. Moreover high level actions (e.g., those which assume information through communication) will be enabled within the action level.

Suppose we like to combine simple expert systems, then the sensoric part is completely disabled (they usually have no representation of themselves), the expert knowledge base contains most of the information together with the action part and (since they are simple XPS) the learner, introspector and planner levels are completely skipped.

The task of the *consistency checker* is to report conflicts between the definitions of the general paradigm module and the specification of single agents in the agent design facility. For example, if it is defined that all the agents can communicate with each other on one hand and in the specification of an agent the whole communication level is disabled then there is an obvious inconsistency which has to be erased.

2.4 The Status Sequence and Statistics Box

The goal we have in using a testbed like RATMAN is to learn about the behavior of different forms of agents in different society structures and different worlds. In reaching this goal it is necessary to analyze not only a snapshot of the world at a certain point (like in the current world scenario), but to look at sequences of such snapshots. The *status sequence and statics box* has the task to show the user the sequence of the changing worlds. That is, essential information from the current world scenario, like locations of the world objects and the agents, is extracted and monitored sequentially to the user. With each world state the whole current world scenario is stored in the background to be reactivated by the user. So there is a possibility for the user to trace the sequence back, to restore the current world scenario at a given point and to enforce the generation of new worlds by changing some of the definitions for the society or the agents (e.g., adding or removing knowledge, communication channels etc.). Thus it is possible to generate a graph of possible worlds and to study the associated accessibility functions, like it is done in the possible world semantics of Modal Logics.

Associated with the status sequence the activity potential of the agents, the internal clock and a monitor for the synchronization processes is installed.

Summing up this paragraph, the main characteristic of RATMAN will be that

- it is purely based on logic, that is, it is possible within the logic paradigm to specify all individual features for simplest up to most complex agents
- the user may be one agent in the society
- there are various facilities that support the analysis of a session.

Especially the first point is not too restrictive as – in contrast to Steels argumentation [Ste89a] – there are a couple of logic based description languages most of which can be translated into derivates of first order logics [Ohl89] providing *efficient* operationalization by calculi based on Robinson's Resolution Principle [Rob65] enhanced with special unification or constrained solving algorithms. For this reason we think that it is now the time to put these logic based approaches together and to find out how far the logic paradigm will reach.

3. RELATED WORK

In the previous section we presented a coarse sketch of RATMAN's structure, that leaves open many implementational details. This is due to the fact that our work is still in a specificational state of progress. The study of the few other approaches to agent testbeds will give us hints how to cope with some problems. On the other hand, from a very global point of view, we wish to separate our work from depending too much on the object-oriented programming approach. In an object oriented environment everything is an object. Objects communicate via messages. Objects built up a hierarchy, usually using some mechanism of inheritance. Replacing the terminus object by agent leads straightforward to a kind of multi-agent system.

In contrast to this we hope to show, that it is possible to build a frame for multi-agent systems that is totally based on logic and is flexible enough to fit for several different paradigms, those based on knowledge and reasoning, but also those based on reactivity and emergent functionality.

3.1 Activation Framework

The Activation Framework (AF) [Gre87] provides a support tool for the implementation of real-time AI programs on multiple interconnected computers. To give a short sketch of AF:

- AF was developed out of the HEARSAY II system.
- AF-Objects (AFOs) communicate by messages.
- They are monitored by frameworks, that perform scheduling and maintain the connection to other frameworks.
- An activation level is attached to AFOs and their hypotheses.
- Uncertainty is handled by the activation level.
- Scheduling decisions also use the activation level.

AF grounds on the *community of experts* paradigm (cf. also [Len75] and [Min86]). Several expert objects, called AFOs communicate by sending messages. Each AFO integrates hypotheses and procedural code for a local knowledge domain, and can be thought of as being similar to a miniature HEARSAY II system with its local blackboard and a set of knowledge source procedures. Indeed [Gre87] traces the evolution of AF evolving from a distributed HEARSAY II approach used for the Distributed Sensor Network (DSN) project [Gre82] back to the HEARSAY II speech understanding system [EHLR80]. That is: From a pure blackboard structure to a distributed blackboard structure using messages to AFOs, which also communicate via messages and encapsulate hypotheses and knowledge procedures.

Each AFO has an input queue and an output queue to receive and send messages respectively. Generally an AFO will take one input message, process it and possibly place a message in its output queue and return control to the AFO scheduler. AFOs usually do not wait for events like e.g. answers to their messages. Each hypothesis has an *activation level* between -1 and 1 which specifies the belief in its truth. This is a key feature of AF. As observations are made, the activation levels of the hypotheses for particular end results are adjusted to reflect the weight of the evidence (akin to certainty measures in MYCIN). At any time a choice can be made and the differences between activation levels give a measure of uncertainty for that choice.

An activation level is also attached to each AFO as a whole and to each message. The sending AFO attaches an activation level to a message according to the level of importance. The activation of an AFO is the sum of all the message activations in its input queue and is used for scheduling purposes. If the activation exceeds some threshold, the AFO is scheduled and the AFO with the highest activation level runs first. The input queue is ordered according to time and activation level order, such that high level and older messages run first.

Messages also contain the names of the sending and the receiving AFO, a message type number and an ASCII-string data body. This leaves a high flexibility to the contents of a messages, the only constraint is that both the sender and the receiver have a common idea about the representation of a particular string.

AFOs are clustered by a *framework*, which is something, that is responsible for scheduling its AFOs and the message transport between them. It also builds up the connection to other frameworks' AFOs, using the mechanisms of message passing of the underlying operating system. Frameworks may support different languages, such that an arithmetic expert AFO can be written in FORTRAN 77 while an inference expert AFO can be written in Lisp. Frameworks also provide the basis for true concurrency, because different frameworks can run in parallel on different processors if the operating system supports an appropriate message passing mechanism.

To conclude and compare with RATMAN we would like to point out:

AF is based on an *object oriented approach* and restricted to the "community of experts paradigm". Though at the current state of the art this is the most promising and fruitful research direction in the multi-agent environments, we would like to integrate more flexibility in our testbed.

AF supports *modularity* on the basis of self contained agents and the message communication provides a clear and simple interface. In RATMAN we try to provide a modularity, that reaches beyond the level of agents. Each agent should be constructed modular. We try to realize this especially with the layered hierarchical knowledge base.

Messages in AF are ASCII-strings. This decision puts no constraints to the contents of messages, because each type of messages (e.g. numbers, hypotheses, procedures...) can be encoded as ASCII-string. In contrast RATMAN will provide features from speech act theories and natural language processing.

The use of an *activation level* for scheduling of AFOs and for decision making within the AFOs is central for AF. In real time AI-applications it is often necessary to make timely decisions with incomplete data. RATMAN is flexible enough to simulate such an activation level concept if needed.

AF is claimed to be a support tool for the implementation of real time AI-programs for distributed cooperative problem solving, while RATMAN should be a support tool and experimental testbed for research with different multi-agent scenarios. It is clear, that a comparison according to the scale of distributed problem solving would prefer AF, while using a scale of experimental testbed (e.g. flexibility) would prefer RATMAN.

3.2 Mace

MACE [GBH86] stands for Multi-Agent Computing Environment and is an instrumented testbed for building experimental DAI systems. The computing units in MACE, the agents, built an organization of problem solvers. Agents run in parallel, communicate via messages and have facilities for knowledge representation. MACE is implemented in Common LISP and also depends on the object oriented programming approach. Central components of MACE are

- an agent description language (ADL),
- tracing and debugging facilities,
- a description database,
- the MACE kernel (responsible for message routing, I/O, mapping agents on processors...),
- a collection of system agents with various tasks like command interpretation, agent building, monitoring agents...
- a collection of facilities, that can be used by all agents like standard messages, standard errors, a pattern matcher, standard engines...

MACE agents contain their *knowledge* in form of attribute values, they *sense* their environment by messages or via events and can take *actions* of three different types:

- Change the internal state (i.e. manipulating their attribute values);
- Send messages;
- Call for some monitoring request.

To carry out these *actions* the agents have locally defined functions, that implement their procedural knowledge - their skills. *Skills* cannot be referenced from outside, however they can be sent in messages to other agents. Skills are called by an agent's *engine*, which is the only active part of an agent. The engine defines the agents activities, interprets messages and manipulates the attributes. An engine is invoked by the kernel when a message for that agent arrives and deactivates itself when the agent has completed its current activity. MACE provides several simple engines. Each engine is wrapped into an *engine shell*, which is a part of the MACE kernel, that provides standard error handling, protects the engine from errors, message queueing and various debugging and tracing facilities. It also implements a state diagram using the agents attribute *status*, which can take one of the values “new”, “inactive”, “active”, “waiting” and “stopped”. Except the initialization procedure an agent usually cycles from “inactive” to “active” when it receives a message and from “active” to “inactive” by deactivating itself. The status “waiting” means waiting for an event after a request to the engine shell to monitor some event. The status “stopped” is used for debugging and tracing.

Sensing is implemented as receiving messages (passive) or being notified of events, that are requested for (active). Messages are queued in arrival order and contain the sender address. There are no restrictions to the message contents. An agent's address consists of the agent's name, its class, the node and the machine. Messages can be addressed to individuals, to groups of agents or to classes. Events, that are effects of some agent's visible actions (e.g. status change, agent creation or destruction...) are monitored by demons. They map event descriptions to messages and send them to those agents, that have requested these events. Events, that are the result of kernel actions (alarms, pattern-trigger ...) are monitored by imps, which are predicates on internal attributes. Imps can invoke arbitrary Lisp functions within the agent.

Agents contain knowledge as values of their *attributes*. A special attribute “acquaintances” contains organizational and interactional knowledge. Other types of knowledge are user defined. *Acquaintances* are the agent's model of the world, i.e. of other agents. Every other agent an agent knows is represented with qualifiers like: name, address, class, roles, skills, goals, plans. After the instanciation an agent only has a model of himself, of its creator and possibly of those agents, that it knows by specific ADL statements. Dynamic alteration of the world model is possible. A special system agent directory provides information about all agents of the system. Knowledge retrieval is done by selectors.

An *organization* is viewed as a structure of expectations and commitments about behavior. It exists only indirectly through the behavior of its members. Nevertheless MACE organizations are represented by a (virtual) communication agent, called the organization manager. It knows about the organization members. Messages sent to the organization arrive at the manager and are forwarded by it to the appropriate organization members. This means that the managers basic work is task allocation.

The *agent description language* (ADL) is a procedural language for describing database operations necessary to construct new agent descriptions. Descriptions of agents are stored in a database and are used to generate executable forms of agents. There is no direct support of hierarchical classifications and of attribute inheritance like in most other object oriented languages, but one can selectively import attributes, functions and engines from other agent-descriptions. One can copy a description of a related agent and modify it by appropriate deletions, imports or new declared attributes for example. Another facility of ADL is the possibility to define specific initialization procedures to be executed at instanciation time.

Because the execution of a MACE agent community involves random execution and message delays, there is also a MACE *simulator* implemented with a totally deterministic behavior. The simulator gives additional measurements of control for debugging and assures repeatable experiments.

MACE was used to implement Smith's contract net ([Smi81] and [DS83]), to build different forms of distributed production systems based on the rule-agent concept [GT86] and to build a simple distributed blackboard system [GBH87]. In comparison to RATMAN we summarize:

MACE realizes *true parallelism*. For the development of DAI systems the MACE simulator provides tracing and debugging facilities. RATMAN will not realize true parallelism, so that RATMAN is closer to the MACE simulator than to the MACE system itself. Nevertheless the blackboard architecture of RATMAN leaves open the possibility of true parallel executing agents in a later version.

Communication in MACE is via message sending. Agents sense their environment via messages and via events. In RATMAN every communication between agents and their environment (sensing) is done via the blackboard.

MACE provides a *description language* ADL, that puts some constraints to the formal architecture of the agents, but their skills could be defined as arbitrary LISP functions. RATMAN agents are specified by their hierarchical knowledge base.

Knowledge representation in MACE is done with the agents attributes especially the world model is represented by their *acquaintances*. In RATMAN each KR-formalism, that can be translated into an appropriate First Order Logic is usable.

3.3 Agora

Agora [BAFLB87] is an environment that supports the construction of large evolutionary programs that might be distributed over several parallel executing processors. In this sense it is just like AF more a construction tool, than a testbed. Its design was driven by the requirements of a distributed speech recognition system of the Carnegie Mellon University [BA86]. Agora provides:

- Support of different languages.
- Multiprocessing within heterogeneous systems.
- Handling of communication and control.
- Different computational models.
- A layered structure.
- An abstraction for computational components: Knowledge Sources.
- An abstraction of data components: Cliques.
- Knowledge representation with semantic nets.

Furthermore the structures can be modified during a run, such allowing evolutionary development of systems. Because the parallel virtual machine layer of Agora provides totally abstract components, different computational models can be realized by Agora's frameworks. For example communication can be done by using a data-flow paradigm, a remote procedure call paradigm or a blackboard paradigm, all depending on the specific definitions of Agora frameworks. The layered structure of Agora can be summarized as follows:

The *bottom layer* consists of a network of heterogeneous processors, that run the Mach operating system which is a Unix compatible OS for multiprocessors.

The *Mach layer* provides those abstractions, that Agora uses to distribute its computations among the machines. These abstractions are: message passing (obligatory for Agora), shared memory (optionally to improve the performance) and threads (optionally for fast creation of new computations).

The *parallel virtual machine layer* expresses computations machine independent in C or Common Lisp using Agora's primitives. These primitives provide virtual main memory in form of sets of homogeneous data elements, that are stored in structures called cliques, and provides special computational components called Knowledge Sources (KS). KS are activated by certain patterns of elements and exchange data via shared cliques. This layer is a kind of assembly language level of Agora.

The *framework layer* builds the interface to an application. A framework is like a special environment to interact with the user in terms of description, translation, debugging, editing, monitoring.... The framework translates the user-supplied code into the proper parallel virtual machine abstractions.

Above this stands the *layer of the application* as for example the structure of the CMU speech recognition system ANGEL.

The computational entities of Agora are the *Knowledge Sources* (KS). A KS contains one or more completely independent functions that manipulate the data of the cliques. Cliques can be shared among several KS. KS functions are activated by patterns expressed in terms of arrival events. E.g. one can specify an activation every time a new element enters a clique or only if this new element has a specific value in one field. One arrival event can be specified to trigger several function executions in different KS or only one execution.

A special element clique is used to implement a simple knowledge base in form of a *semantic net*. The frames of this network contain slots representing attributes or relations to other frames. A simple but effective set of primitives realizes a simple frame language CFrame, which allows the KS to manipulate the network thus sharing knowledge with other KS, which is a crucial possibility for building up frameworks. Typical Agora *language primitives* are procedures to instanciate KS, to get an access capability to elements, to add new elements to a clique, to terminate a KS, to set an activation pattern for a function and so on.

The power of Agora lies in the *parallel virtual machine layer*. It was designed to achieve two goals: To efficiently execute different programming models and to avoid the restriction to a specific computer architecture. A virtual main memory is provided through the global data structure of element cliques and the computational units have the form of KS.

The *framework level* above this layer encourages the distinction between an application engineer and a researcher. The structure of an application is implemented by a set of frameworks, which is the task of an application engineer. This set provides the tool for the researcher to design and test an actual system. The layered structure of Agora thus allows a very great flexibility, but the design of a new framework structure could be a time expensive task.

Concluding remarks:

Agora *supports different languages* and provides a great *flexibility* in terms of the programming model. The *parallel virtual machine layer* abstracts from the underlying computer architectures. Knowledge representation is based on a *semantic net* and the frame language CFrames.

While the parallel virtual machine layer represents a kind of assembly language for Agora, the *framework level* gives an application specific tailored user interface. The framework design will be speeded up by similarities with existing frameworks: Not every detail must be assembled from the bottom.

Two main characteristics of Agora are the support of *highly parallel computations* and the *pattern directed activation* mechanism. In RATMAN parallelism will only be simulated as stated earlier. Activation of agents will usually be by changes of the environment but other mechanisms are possible.

As with AF a direct comparison between Agora and RATMAN is not efficient, because Agora is a support environment for the construction of large distributed (not necessarily) AI- programs, while RATMAN is more a tool for learning more about the fundamental principles that apply to multi-agent systems.

3.4 Mages

MAGES [BFS90] is a testbed for Multi AGEnt Systems. It is useful for experimenting with different types of interactions between agents using heterogeneous architectures and behavior. A catchword characterization of MAGES is:

- testbed for research and education, provides flexibility to various aspects of multi-agent societies.
- based on the object oriented approach, implemented in the actor language ACTALK.
- agenttypes are organized in an inheritance tree.
- environments are active objects.
- communication is based on asynchronous message passing, environments additionally can use events and actions for communication.
- a translation procedure enables communication between agents of different lingual level.

MAGES is claimed to provide a maximum of *flexibility*, e.g. in terms of coordination and interaction protocols, agent granularity, knowledge representation, behavior and internal control. Furthermore it is told to be simple, user friendly and easy to debug. MAGES is implemented in SMALLTALK as an implementation of the actor language ACTALK[Bri89]. ACTALK is a modular language in which objects are already active and autonomous. Parallelism is managed and ACTALK provides an asynchronous communication mechanism as well as useful graphical tools.

ACTALK agents consist of a *behavior object*, which holds the agent's fields and methods and is a usual SMALLTALK process. It is encapsulated in an *actor object*, which manages the agent's mailbox. Messages have the usual SMALLTALK syntax and activate the corresponding method of the behavior object. For simple reactive agents this method will just perform some appropriate actions. Complex agents will use simple messages as frame based objects which have to be inserted into their knowledge base. They will send messages of self defined arbitrary complex message classes. A special *message translation procedure* can convert messages of complex classes to simple ones, such that complex agents need not care to what type of agent they send their complex messages.

The *communication model* is based on a high-level language, which includes acts of communication and protocols, but also can be used for simple communication models. With the help of the translation procedure heterogeneous agent societies, where communication takes place between simple and complex agents, can be designed. Protocols are used to open and end a communication and to manage the proceeding of a complex dialog. The possibility of dynamic destruction of agents leads to the blackhole structure: Destroyed agents are replaced by a blackhole that replies every message with a “no more receive” message. A three stage protocol, that uses special intention objects, guarantees mutual exclusion for shared resources.

MAGES agents architectures build the structure of an *inheritance tree*. Simple agents have only a very simple reasoning process and follow the stimulus-response model. Coarse grained agents have an explicit representation of their cognitive process, can reason about their skills and their domain and can work as individual blackboards. The basic behavior is inherited from the root class of agents, specialized agents override some methods with more sophisticated ones. The most basic

KernelAgent class defines its information only in terms of acquaintances and environment. The internal control follows a *basic behave cycle*, that realizes a simple reactive behavior to external input. In the ExpertAgent class this basic behavior is augmented with an *inference engine* to give the agent some reasoning capabilities. MessageAgents as a further step can integrate messages as part of their knowledge base. BlackboardAgents and HybridAgents provide most sophisticated facilities for reasoning and knowledge representation.

In MAGES also the *environment* consists of one or more agents. This way a centralized environment with a global view (e.g. environment means playing field) can be modeled as well as a distributed one with only a local view of the neighborhood (e.g. each field of the playing ground is represented by an environment agent). The connection between environment and agents is implemented by *asynchronous messages*, by *events* and by *actions* (that are special objects, whose changing is recognized by the agents). Another point of flexibility lies in the choice between a passive mode, where agents are informed of changes by the environment and an active mode, where agents actively scan their environment for changes. Thus MAGES as a generic platform provides various structures as environment and the user may choose one, that fits best to his application.

The *user interface* of MAGES is based on the Model-View-Controller of SMALLTALK. Specific browsers are used to display various information about agents, messages, KS and Blackboards. A menu driven inspection and manipulation of agents is provided and different views can be selected. The AcquaintanceNetworkView shows the channels of communication within a group of agents. The CommunicationView shows the message sending and the agent's mailboxes. The BoardView shows a grid with icons for the agents as a graphical representation of the environment. All the graphical features of the user interface are based on the graphical tools, that ACTALK already provides.

To built up *organizations* in a multi-agent society, two approaches have been suggested: Organizations exist implicit by the organized behavior of its members. Organizations are explicitly represented by certain structures. Both approaches can be modeled with MAGES: The first one by simply installing the desired behavior in the agents. The second approach uses groups, special agents, that are defined by its group members, a message interpreter and a behavior. Groups are dynamic in the sense that they can change their behavior on different types of agreement (e.g. election, consensus, leader domination...). The message interpreter accepts messages sent to the group and takes the appropriate actions like for instance distribute the message to specific group members or perform a kind of task allocation. The membership to groups can be gained via a biding protocol.

MAGES is centered around the *object oriented* actor language ACTALK. Several features of ACTALK fit perfectly into the multi-agent paradigm. RATMANs approach is centered around the hierarchical knowledge bases, thus any symbolic knowledge representation language can be used.

MAGES provides various parameters of *flexibility*, which is crucial for a testbed to be useful for research. We hope to be able to realize at least the same flexibility.

The *environment* in MAGES is realized as one or more active objects. This bears the possibility of modelling heterogeneous and even recursive environments. The interaction between agents and environments can be realized by messages, events and actions.

The *user interface* of MAGES also depends heavily on the features provided by ACTALK. For RATMAN we plan to use some XWindow based user interface.

3.5 Other Systems

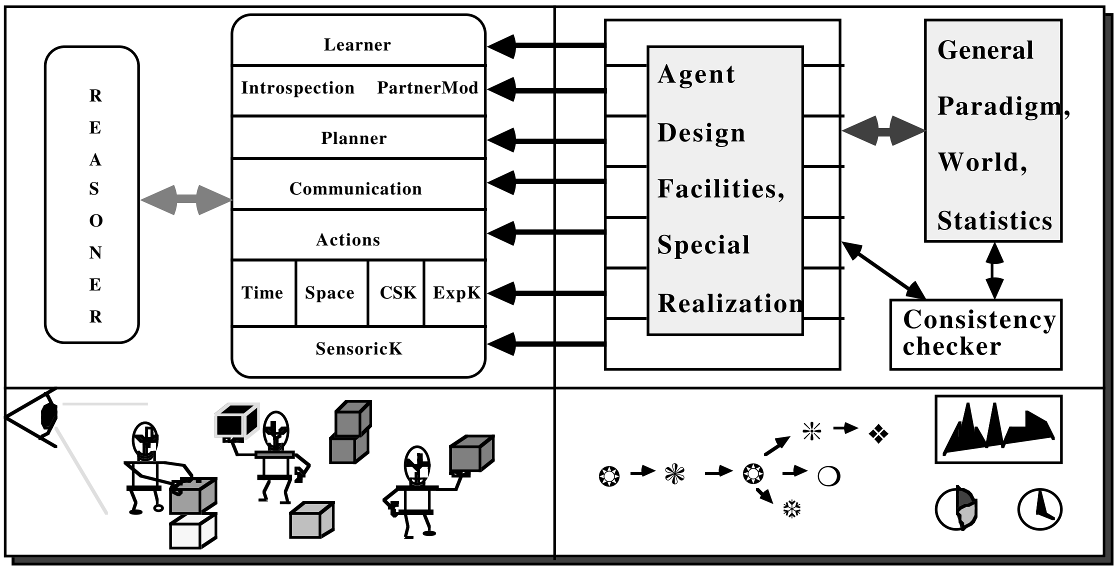
We tried to sketch some of the systems closely related to the intentions of RATMAN. However there are some other systems, that we only want to mention by name here, because a broader elaboration would not lead to aspects previously unexpressed .

MICE [DM89] is a testbed for intelligent coordination experiments and provides a 2D grid environment model. CoCo [Ish89] deals with concurrent and cooperative operator tasks. The PANDORA II [MIT89] system provides help for modelling autonomous agents and groups of them. Playground [FB89] is another object oriented simulation system.

Not explicitly a testbed, but as we think a very useful and constructive architectural design is presented in [HNS89]. Hultman, Nyberg and Svenson took a temporal logic as the basis of their system.

4. Discussion

We described the testbed RATMAN for experimentation in multi-agent scenarios which is again summarized in the figure below.



In constructing RATMAN we are motivated by the study of the behavior of agents in a society and the emergent functionality of the agent society per se. We are leaded by the necessity of combining most different models of agents from dump reactive ones to highly knowledge-based, intelligent forms, in a variety of social structures. Finally we are inspired by the idea of combining different classes of knowledge based on logic to work in a well understood and understandable paradigm. (However we know about the problems with respect to “meta knowledge” [Doy88], [Mae86] and meaning or semantics [Lak87].)

The advantage of RATMAN is its modularity. The knowledge base can be stepwise refined along the hierarchy and new paradigms can easily be added to the system. Even with an “O-version” interesting results might be achieved. In using the blackboard architecture in the current world scenario eyes and ears and other sensoric features of mobile agents can be simulated and moreover this features can be easily replaced in a scenario where really distributed, multi processor systems are connected to RATMAN (even replacement by connectionistic networks are possible [Dye89], [Sha89], [FSG89]). But the main point is that the principal decision to use logic as a basis results in a homogeneous approach which overcomes major difficulties of combining and translating different modelling paradigms.

The main problems lay on the more theoretical side. Though we like to use only knowledge representation languages that correspond to logic, each level of our knowledge base coincides with a whole field of research in AI and thus it could not be possible to build in all specific results of these fields. But nevertheless there must be a first approach to combine at least the basic results into one homogeneous system, even in standardizing apart some of the features and even if the standardization only partly fits with reality. The second point is that currently there is no tool to support the analysis of the behavior of multi-agent societies. That is, only the pure facts will be reported by the status sequence and statics box and it would be nice to have a kind of meta-system that supports the user in analyzing the results delivered by RATMAN.

ACKNOWLEDGEMENTS**:**
The multi-agent project is sponsored by the German Ministry for Research and Technology under grant ITW 8903 0.

5. References

[All84]	Allen, J.F.:Towards a General Theory of Action and Time, AI 23 (1984), pp. 123-154
[BA86]	Bisiani, R., Adams, D.A.: The CMU Distributed Speech Recognition System, 11th. DARPA Strategic Systems Symposium, Naval Postgraduate School, Monterey, CA, Oct. 86
[BAFLB87]	Bisiani, R. et al.: The Architecture of the Agora Environment, in [Huh87], pp. 99-117
[BFS90]	Bouron, T., Ferber, J.,Samuel, F.: MAGES: A Multiagent Testbed for Heterogeneous Agents, Proc. of the 2nd European Workshop MAAMAW'90, cf. [DM91]
[BG88]	Bond, A., Gasser, L.: Readings in Distributed AI, Morgan Kaufmann, Los Angeles, 1988
[BM90]	Bürckert, H.-J., Müller, J.: RATMAN: Rational Agents Testbed for Multi Agent Networks, Proc. of the 2nd European Workshop MAAMAW'90, cf. [DM91]
[Bri89]	Briot, J-P.: Actalk: a Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment, LITP University Paris VI, ECOOP'89
[BS85]	Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System. Cognitive Science 9(2), 171-216, 1985
[Bür90]	Bürckert, H.-J.: A Resolution Principle for Clauses with Constraints. Proc. of 10th Intern. Conf. on Automated Deduction, Springer LNAI 449, 1990
[Coe88]	Coelho, H.: Interaction Among Intelligent Agents, ECAI88, 717-718, 1988
[CSW88a]	Connah, D., Shiels, M., Wavish, P.: Towards Artificial Agents that can Cooperate, Technical Note 2643, Philips Research Labs, England, 1988
[CSW88b]	Connah, D., Shiels, M., Wavish, P.: A Testbed for Research on Cooperating Agents, Technical Note 2644, Philips Research Labs, England, 1988, also short version in ECAI88
[DM89]	Durfee, E.H., Montgomery, T.A.: MICE: A Flexible Testbed for Intelligent Coordination Experiments, Proc. of the 9th Workshop on DAI, Seattle 1989
[DM90]	Demazeau,Y., Muller, J.-P.: Decentralized Artificial Intelligence, Proc of the first workshop on Modelling Autonomous Agents in a Multi-Agent World, Elsevier Sc. Pub/North Holland, 1990
[DM91]	Demazeau,Y., Muller, J.-P.: Decentralized Artificial Intelligence, Proc of the second workshop on Modelling Autonomous Agents in a Multi-Agent World, Elsevier Sc. Pub/North Holland, 1991 (forthcoming)
[Doy88]	Doyle, J.: Knowledge, Representation and Rational Self-Governement, Proc. of the Reasoning about Knowledge (RAK 88) Conference, 345 - 353, 1988

[DS83] Davis, R., Smith, R.G.: Negotiation as a Metaphor for Distributed Problem Solving, Artificial Intelligence vol. 20, pp. 63-109, 1983

[Dye89] Dyer, M. G.: Symbolic Processing Techniques in Connectionist Networks and Their Application to High-level Cognitive Tasks, Int. GI-Congress on Wissensbasierte Systeme 89, Munich, Springer IFB227, Berlin, 173 - 185, 1989

[EHLR80] Erman L. et al.: The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty, ACM Computing Surveys, Vol. 12, pp. 213-253, 1980

[FB89] Fenton, J., Beck, K.: Playground: An Object Oriented Simulation System with Agent Rules for Children of All Ages, OOPSLA 89 pp. 123-137.

[FSG89] Frixione, M., Spinelli, G., Gaglio, S.: Symbols and subsymbols for representing knowledge: a cataloque raisonné, ECAI89, 3 - 7, 1989

[GBH86] Gasser, L., Braganza, C., Herman, N.: MACE: A flexible Testbed for Distributed AI Research, in [Huh87], pp. 119-152

[GBH87] Gasser, L., Braganza, C., Herman, N.: Implementing Distributed AI Systems Using MACE, Proc. IEEE Third Conference on AI Applications, Feb. 1987, also in [BG88] pp. 445-450

[GGR86] Genesereth, M.R., Ginsberg, M.L., Rosenschein, J.S.: Cooperation without Communication, Proc. AAAI-86, 1986

[Gre82] Green, P.E.: Distributed Acoustic Surveillance and Tracking, Proc. Distr. Sensor Networks Workshop, pp. 117-141, M.I.T. Lincoln Laboratory, Lexington, MA, Jan. 6, 1982

[Gre87] Green, P.E.: AF: A Framework for Real-Time Distributed Cooperative Problem Solving, in [Huh87], pp. 153-175

[GT86] Gasser, L., Tenorio, M.F.: Rule-Agents: A Distributed, Object-Oriented Approach to Production Systems Using MACE, DAI Group Research Note 4, DAI Group, Dept. of Computer Science, USC, 1986.

[Had88] Hadley, R.F.: Logical Omniscience, Semantics, and Models of Belief. Computational Intelligence 4, 17-30, 1988

[Hay79] Hayes, P.J.: The Logic of Frames. In: Frame conceptions and text understanding, (D. Metzing, ed.), deGruyter, 1979

[Hin75] Hintikka, J.: Impossible Possible Worlds Vindicated. J. Philosophical Logic 4, 475-484, 1975

[HNS89] Hultman, J., Nyberg, A., Svensson, M.: A Software Architecture for Autonomous Systems, Research Report LiTH-IDA-R-89-40, Univ. Linköping

[Huh87] Huhns, M.N.: Distributed Artificial Intelligence, Vol.I, Pitman/ Morgan Kaufmann Publ., San Mateo,CA, 1987

[Huh90] Huhns, M.N.: Distributed Artificial Intelligence, Vol.II, Pitman/ Morgan Kaufmann Publ., San Mateo,CA, 1990

[Ish89] Ishida, T.: CoCo: a Multiagent System for Concurrent and Cooperative Operator Tasks, Proc. of the 9th Workshop on DAI, Seattle 1989

[Kod86] Kodratoff, Y.: Learning Expert Knowledge and Theorem Proving. Springer Informatik Fachberichte 124, 164-180, 1986

[KP89] Konolige, K., Pollack, M. E.: Ascribing Plans to Agents, IJCAI89, 924-930, 1989

[Lak87] Lakoff, S.: Women, Fire and Dangerous Things, Chicago University Prss, 1987

[LC81] Lesser, V.R., Corkill, D.D.: Functionally Accurate, Cooperative Distributed Systems, IEEE Trans. on SMC, 11 (1): 81-96, Jan. 81, also in [BG88]

[Len75] Lenat, D.B.: BEINGs: Knowledge as Interacting Experts, IJCAI'75, pp. 126-133, also in [BG88]

[Mae86] Maes, P.: Introspection in Knowledge Representation, AI Memo 86-3, Vrije Universiteit Brussel, 1986

[Mae89] Maes, P.: The Dynamics of Action Selection, IJCAI89, 991-997, 1989

[Maz88] Mazer, M.S., A Knowledge Theoretic Account of Recovery in Distributed Systems: The Case of Negotiated Commitment, RAK88, 309-323, 1988

[McA88] McArthur, G.L.: Reasoning about knowledge and belief: a survey. Computational Intelligence 4, 223-243, 1988

[Min75] Minsky, M: A framework for representing knowledge. In: The psychology of computer vision (P.H. Winston, ed.), McGraw-Hill, pp. 211-277, 1975

[Min86] Minsky, M.L.: The society of mind, New York, Simon and Schuster 1986

[MIT89] Maruichi, T., Ichikawa, M., Tokoro, M.: Modeling Autonomous Agents and Their Groups, Keio University, in [DM90].

[MM88] Moyson, F., Manderick, B.: The Collective Behavior of Ants: An Example of Self-Organization in Massive Parallelism, AI Memo 88-7, Vrije Universiteit Brussel, 1988

[Mor89] Morik, K. (ed.): Knowledge Representation and Organization in Machine Learning. Springer LNAI 347, 1989

[Mül91] Müller, J.: Defining Rational Agents by Using Hierarchical Structured Knowledge Bases, IEE Computing and Control Division Colloquium on "Intelligent Agents", Digest No. 1991/048, London, Feb. 91

[Mye88] Myerson, R.B., Incentive Constraints and Optimal Communication Systems, Proc. of the Reasoning about Knowledge (RAK 88) Conference, 179-193, 1988

[Neb90] Nebel, B., Reasoning and Revision in Hybrid Representation Systems, Springer LNAI 422, Berlin 1990

[Ohl89] Ohlbach, H.J.: Context Logic – An Introduction, 13th German Workshop on AI, Springer IFB216, Berlin, 1989

[Per85] Perlis, D.: Languages with Self-reference I: Foundations. Artificial Intelligence 25, 301-322, 1985

[Per88] Perlis, D.: Languages with Self-reference II: Knowledge, Belief, and Modality. Artificial Intelligence 34, 179-212, 1988

[RG85] Rosenschein, J.S., Genesereth, M.R.: Deals Among Rational Agents, IJCAI'85, pp. 91-99

[Rob65] Robinson, J.A.: A Machine Oriented Logic Based on the Resolution Principle, J.ACM 12(1), 1965

[SD83] Smith, R. & Davis, R.: Framework for Cooperation in Distributed Problem Solving. IEEE Transactions on Systems, Man, and Cybernetics SMC-11 (3), pp. 161-169, (1983), also in [BG88]

[Sha89] Shastri, L.: Connectionism, Knowledge Representation and Effective Reasoning, Int. GI-Congress on Wissensbasierte Systeme 89, Munich, Springer IFB227, Berlin, 186 - 195, 1989

[Smi81] Smith R.G.: A Framework for Distributed Problem Solving, UMI Research Press, Ann Arbor, Michigan, 1981

[Ste89a] Steels, L.: Cooperation between Distributed Agents through Self-Organization, AI Memo 89-9, Vrije Universiteit Brussel, 1989

[Ste89b] Steels, L.: Issues and Open Problems in Subsymbolic Representation, AI Memo 89-13, Vrije Universiteit Brussel, 1989

[Tha89] Thayse, A.: From Modal Logic to Deductive Databases, Wiley, Chichester, 1989

[TM89] Tennenholtz, M., Moses, Y.: On Cooperation in a Multi-Entity Model, IJCAI 89, 918-923, 1989

[Wer88a] Werner, E.: Toward a Theory of Communication and Cooperation for Multi-agent Planning, RAK88, 129-143, 1988

[Wer88b] Werner, E.: Social Intentions, ECAI88, 719-723, 1988

[WHBSS81] Wesson et al: Network Structures for Distributed Situation Assessment, IEEE Trans on SMC 11 (1), pp. 5-23, Jan 81, also in [BG88]

[ZR89] Zlotkin, G., Rosenschein, J.S.: Negotiation and Task Sharing Among Autonomous Agents in Cooperative Domains, IJCAI89, 912-917

DFKI Publikationen

Die folgenden DFKI Veröffentlichungen sowie die aktuelle Liste von allen bisher erschienenen Publikationen können von der oben angegebenen Adresse oder per anonymem ftp von ftp.dfki.uni-kl.de (131.246.241.100) unter pub/Publications bezogen werden.
Die Berichte werden, wenn nicht anders gekenn-zeichnet, kostenlos abgegeben.

DFKI Publications

The following DFKI publications or the list of all published papers so far are obtainable from the above address or per anonymous ftp from ftp.dfki.uni-kl.de (131.246.241.100) under pub/Publications.
The reports are distributed free of charge except if otherwise indicated.

DFKI Research Reports

RR-92-46
Elisabeth André, Wolfgang Finkler, Winfried Graf, Thomas Rist, Anne Schauder, Wolfgang Wahlster:
WIP: The Automatic Synthesis of Multimodal Presentations
19 pages

RR-92-47
Frank Bomarius: A Multi-Agent Approach towards Modeling Urban Traffic Scenarios
24 pages

RR-92-48
Bernhard Nebel, Jana Koehler:
Plan Modifications versus Plan Generation: A Complexity-Theoretic Perspective
15 pages

RR-92-49
Christoph Klauck, Ralf Legleitner, Ansgar Bernardi:
Heuristic Classification for Automated CAPP
15 pages

RR-92-50
Stephan Busemann:
Generierung natürlicher Sprache
61 Seiten

RR-92-51
Hans-Jürgen Bürckert, Werner Nutt:
On Abduction and Answer Generation through Constrained Resolution
20 pages

RR-92-52
Mathias Bauer, Susanne Biundo, Dietmar Dengler, Jana Koehler, Gabriele Paul: PHI - A Logic-Based Tool for Intelligent Help Systems
14 pages

RR-92-53
Werner Stephan, Susanne Biundo:
A New Logical Framework for Deductive Planning
15 pages

RR-92-54
Harold Boley: A Direkt Semantic Characterization of RELFUN
30 pages

RR-92-55
John Nerbonne, Joachim Laubsch, Abdel Kader Diagne, Stephan Oepen: Natural Language Semantics and Compiler Technology
17 pages

RR-92-56
Armin Laux: Integrating a Modal Logic of Knowledge into Terminological Logics
34 pages

RR-92-58
Franz Baader, Bernhard Hollunder:
How to Prefer More Specific Defaults in Terminological Default Logic
31 pages

RR-92-59
Karl Schlechta and David Makinson: On Principles and Problems of Defeasible Inheritance
13 pages

RR-92-60
Karl Schlechta: Defaults, Preorder Semantics and Circumscription
19 pages

RR-93-02
Wolfgang Wahlster, Elisabeth André, Wolfgang Finkler, Hans-Jürgen Profitlich, Thomas Rist:
Plan-based Integration of Natural Language and Graphics Generation
50 pages