Deutsches
Forschungszentrum
für Künstliche
Intelligenz GmbH

Research
Report
TM-98-09

The EMS Model

Jürgen Lind

January 1999

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH

Postfach 20 80
67608 Kaiserslautern, FRG
Tel.: + 49 (631) 205-3211
Fax: + 49 (631) 205-3210
E-Mail: info@dfki.uni-kl.de

Stuhlsatzenhausweg 3
66123 Saarbrücken, FRG
Tel.: + 49 (681) 302-5252
Fax: + 49 (681) 302-5341
E-Mail: info@dfki.de

WWW: http://www.dfki.de

The EMS Model

Jürgen Lind

DFKI-TM-98-09

This work has been supported by a grant from The Federal Ministry of Education, Science, Research, and Technology (FKZ ITW-).

ISSN 0946-008X

The EMS Model

Jürgen Lind

January 28, 1999

Abstract. The interface which connects the agent to its environment is usually based on the particular architecture of the agent. This makes it difficult for agents of different architectures to interact with each other within the same virtual environment.
In this paper, we present a model to describe the agent/world interface in a generic, architecture independent way. The usefulness of the model is demonstrated in a generic testbed library which facilitates rapid prototyping for experimental multi-agent systems. Also, a network protocol derived from the basic model which allows for the development of distributed testbeds is presented.

1 Introduction

Agent testbeds are needed to evaluate the strengths and limitations of agent architectures; examples of generic testbeds are (Durfee and Montgomery, 1989), (Pollack and Ringuette, 1990) or (Wooldridge and Vandekerckhove, 1994). However, a difficult problem in the field of multi-agent systems research is to have agents of different architectures interact within a single testbed because often, testbeds are build around a specific architecture and are thus not open for agents with another architecture.
In the multi-agent systems group at the DFKI, we are experimenting with three different architectures: InteRRaP agents (Müller, 1996) which integrate reactive behavior and deliberation, MECCA agents (Lux and Steiner, 1995) which concentrate on agent negotiation processes and SIFAgents (Funk et al., 1997) focusing on social interaction between agents. In order to compare these architectures or to combine their respective special abilities in a single (virtual) environment, we have to build a testbed which allows these different architectures (and therewith their different interfaces to their environment) to interact with each other. Furthermore, building such a testbed should be easy and straightforward, neglecting special (implementation) issues of the architectures.
Multi-agent systems can be distinguished into two classes: systems which are built from agents of the same class and with all agents working together on a common task are called *closed* systems. Usually, systems of this class are build by a single designer. Examples of closed systems include fleet scheduling for transportation agencies (Bürckert et al., 1998), distributed theorem proving (Denzinger, 1995) or engineering design (Shen and Barthes, 1995). On the other hand, we have *open* systems (Hewitt, 1985). The characteristic

feature of systems of this class is that they have agents with different architectures (or even with humans (van de Velde, 1997)) and that these agents usually pursue individual goals which can be (and usually are) contradictory. An example for this kind of system is Internet trading. There, an agent is told by its user e. g. to look for cheap flight tickets. To find the best offer, the agent must find a trading place where it can negotiate with ticket selling agents about about a possible deal. A more complex domain where agents from different designers meet with each other is robot soccer (Noda, 1995). In that scenario, the agents do not only communicate with another, but they can also act upon their virtual environment (the soccer field) and the changes of the environment have to be distributed among the playing agents.

Existing multi-agent architectures often neglect the agent/agent or the agent/world interface because they are designed for closed systems. However, when it comes to integrating agents in a heterogeneous system (even a closed system with a common task), the agent architectures which settle on a private interaction scheme will fail because they are not able to interact with agents designed for another interaction model.

Another important point with respect to agent interaction and interaction testbeds is how to reduce the design overhead introduced by re-inventing the wheel when setting up a new application. A lot of work can be saved by implementing reusable interaction patterns which can be refined to fit a particular application area.

In this paper, we present a general model for agent interaction in heterogeneous multi-agent systems. This model will be the basis for the implementation of reusable software components for agent simulation testbeds. A major advantage over existing approaches is the fact, that the implementation of the basic interaction model can vary from single process implementations using method invocation to realize the model primitives to distributed implementations using network protocols. However, the basic model and therewith the agent using it, remain unchanged regardless of which type of implementation is chosen. Furthermore, we provide a modular approach to various forms of agent interaction which helps the agent designer to rapidly build prototypical multi-agent systems by combining generic modules. The basic idea is, that there exist only a limited number of possible interaction schemes which can be described on a very general level and the resulting modules can be easily instantiated for particular domains.

This paper is organized as follows: firstly, we present the basic ideas of our framework and highlight the analogies which where used to derive the model. Then, we present the implementation of a library which is built upon the *Effector/Medium/Sensor (EMS)* paradigm and demonstrate how the library is used to build agent testbeds. A network protocol which allows for the development of distributed testbeds using the EMS model is presented in the following section. Finally, a conclusion and some outlook is given.

2 Basic Model

Our approach to find a uniform framework for agent interaction is loosely based on the concept common in Internet chatrooms. To enter such a virtual room, the user connects to a chatroom server with a (real or fake) name. Then, he or she is informed which other persons are currently in the room and he or she can send messages to either all people in the room or to individual persons. The basic idea of our approach is quite similar: an agent is equipped with a number of so-called *effectors* through which it can act upon its virtual environment and with a number of *sensors* through which it can perceive the state of the environment. When an effector or a sensor is created, it registers itself with a server (which will be called the *medium* for reasons explained below). Whenever the agent wants to perform the action associated with a particular effector, the effector sends a notification to the server to inform it about the effector activation. It is the the task of the server to calculate the effects of the activation on the sensors registered with it and to inform them about the change in the environment.
Now, why is the medium called "medium"? The reason for this naming scheme comes from the second analogy on which our model is based. In a physical world (e. g. verbal communication) an effector (the speech apparatus) changes the state of a medium (the air) which distributes its state changes to the sensors (the ear) connected to the medium. Thus, the medium implements a transition function which maps effector activations to sensor data. The signal flow of this example is depicted in figure 1.

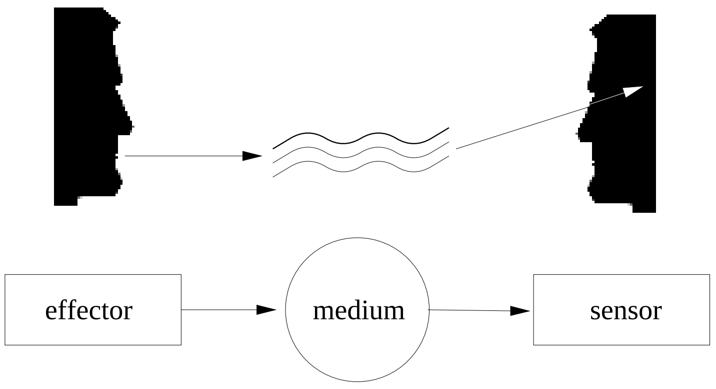

Fig. 1. EMS Model

Several effectors, media and sensors which belong together logically can be combined in a group in order to express their logic connection. We will call such a group an *Effector/Medium/Sensor (EMS) system* in the subsequent text; examples for EMS systems are communication or movement/vision.
In a EMS system for (verbal) communication (see figure 2) the agent has an effector which can be used to send a message to medium. It is then the task of the medium to route this message to the receiver(s) audio sensors.

It depends on the implementation of the medium which forms of communication are possible. In a one-to-one medium, agents can send messages to other agents without third parties perceiving the message exchange. In a broadcast medium on the other hand, each message is sent to all audio sensors registered with the medium which makes it impossible to send secret messages to another agent. Note that the medium in this example does not maintain any state information. It simply routes the message sent by the speaker to the respective audio sensors.

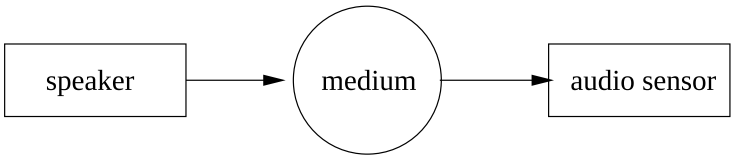

Fig. 2. EMS system for (verbal) communication

An EMS system for movement/vision is more complex then one for communication because it requires a topological structure of the underlying virtual environment. A major difference to the simple EMS system shown in figure 2 is the fact that the medium in figure 3 keeps state information, e. g. where the agents are currently located. In the Social Interaction Framework (SIF) (Funk et al., 1997) presented in the next section, we have implemented an EMS system which handles moving and visual sensing in a tile world.

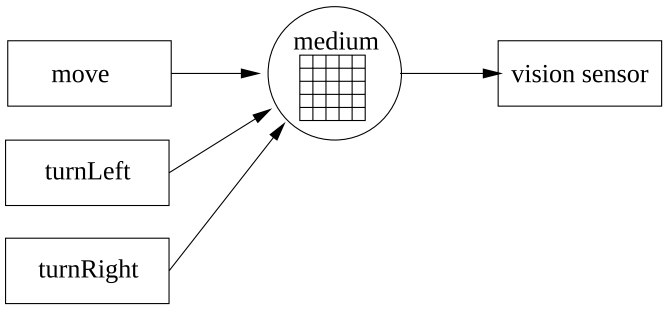

Fig. 3. EMS system for movement and vision

The underlying virtual environment is a rectangular area made up of tiles of the same size. These tiles induce a topology on the virtual world because each object has unique position in the world. Each agent is represented by its shape (e. g. triangle) and its color (e. g red). Furthermore, each agent has an actual direction and a current position. On the left hand side of figure 4, we have a red, triangular agent located at position (2,3) (with the upper left corner as the origin). The agents current direction (`NORTH`) is indicated by a small triangle at the top of the agent symbol. Please note in this respect, that the visualisation engine for a specific virtual world is more or less independent from the implementation of the medium. In the SIF library, we have decided to use the Java Interface to the underlying window system because it is straightforward to visualise the data model of the medium. However, we are currently implementing a visualisation engine based on VRML (The VRML consortium, 1997) in order to support more realistic 3D visualisations in future versions of our library.

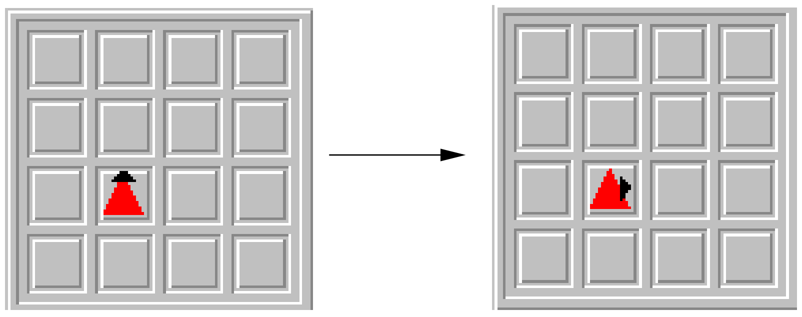

Fig. 4. Agent turning in SIF

The effectors to support the navigation in this world are a ”turn” effector which changes the current direction (see figure 4) of an agent and a ”move” effector effector which moves the agent one tile in its actual direction (if possible, e. g. if there is no obstacle in front of the agent, see figure 5).

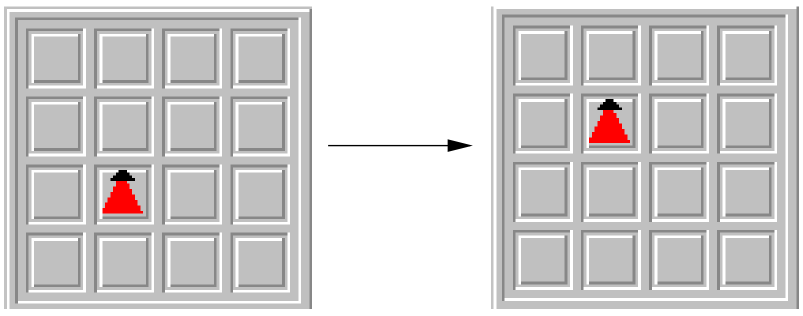

Fig. 5. Agent moving in SIF

Sensing in this EMS system is accomplished by a ”view” sensor. This kind of sensor has a perception range which limits the part of the environment which can be observed. Figure 6 shows the perception range of the agent. The agent can ”see” any object in its ”view” sensor range. However, the objects in our scenario are not labeled (e. g. as being an agents), rather, object are described by their external features (color, shape, etc.) and it is the task of the agent to find out which is which.

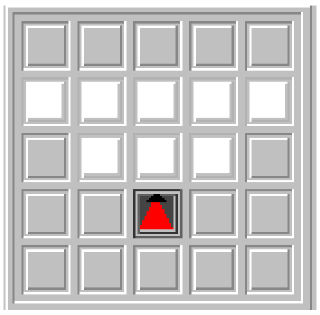

Fig. 6. Agent sensor range in SIF

3 The SIF Library

SIF is a library to build simulation testbeds for multi-agent applications with a focus on interaction among agents of different architectures. The library provides a number of generic objects and object connecting mechanisms in order to support rapid prototyping of experimental systems. It is based on the EMS paradigm presented in the previous section. The EMS paradigm facilitates the smooth integration of different agent architectures in a single application. The entire library is build in Java.
Before we go into the technical details of the library, the general idea of how to build an own application is given:

1. The system designer decides which forms of interaction are needed in his or her simulation. For example, in an application to evaluate negotiation protocols, verbal communication facilities are sufficient for the agents. In an application where the agents are situated in a virtual environment, on the other hand, mechanisms to describe the environment and the agents acting in the environment are needed.
   The result of this design step are the EMS systems needed for the particular application.
2. Since the SIF library is a generic basis for experimental applications, the selected EMS systems must be refined in order to fit the particular needs of the new application. This is done by inheriting from the respective base classes and adding the scenario specific parts. The result of this step are sets of refined effectors, sensors and media.
3. The media generated in the previous step are integrated in a single program which operates as the simulation engine of the new application. In our terminology, this engine is referred to as the “World Sever” of the new application because it implements the simulation specific “physical rules” of the agents environment.
4. The refined effectors and sensors are integrated in the agent architecture. Currently, only agent architectures implemented in Java can be connected to a world server, because the SIF library is written in Java. However, in section 4, we present a network protocol for the EMS paradigm

which overcomes this limitation and which allows us to have architectures implemented in arbitrary languages in the same simulation.

Each application built with the SIF library has the same basic control cycle: When an agent is started, it will at first create a number of effectors and sensors which register themselves with a medium of the world sever. Then, the SIF control loop depicted in figure 7 starts. Whenever an effector is activated (①), it posts an activation message (②) to the event queue (③) of the medium it is connected to. This queue is organized in FIFO manner in order to maintain the temporal ordering of the effector activations. To process a particular event, the dispatcher (④) of the medium calculates the effects of the effector activation. This part is usually application specific except for few general purpose EMS systems. If a sensor registered with the medium is affected by a particular action, it is sent a notification message which contains the new perception (⑤). The sensor itself posts (⑥) the new percept into the perception queue of the agent (⑦) which decides (⑧) about effector activations in the next cycle.

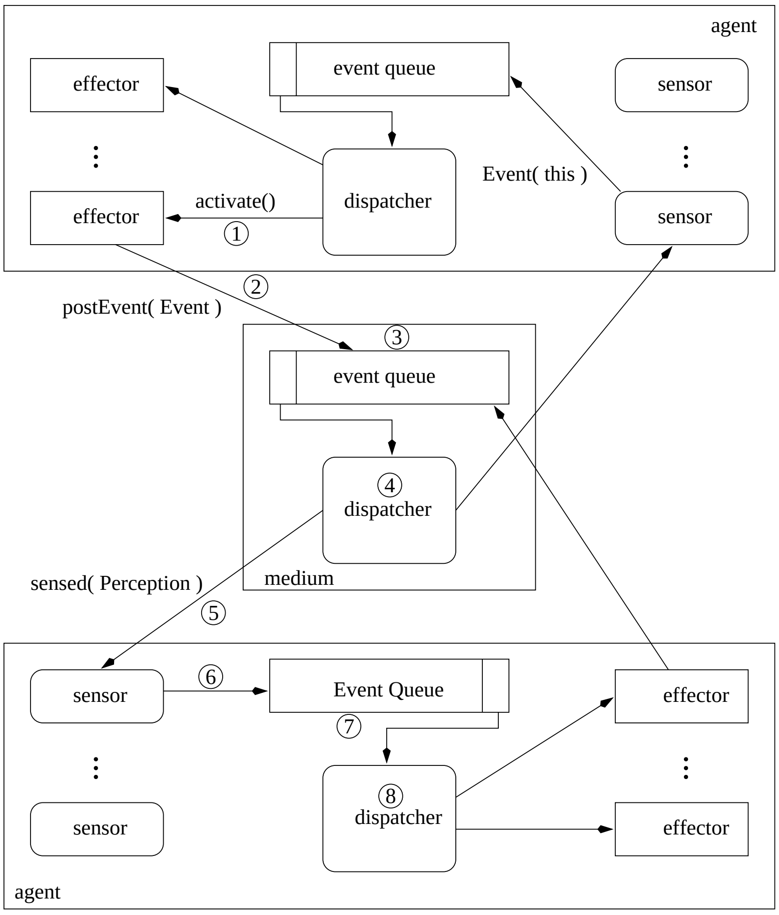

Fig. 7. EMS Model

3.1 Library structure

The overall structure of the SIF library is organized in three layers as depicted in figure 8. The lowest layer is the SIF kernel, which contains library specific data structures such as `SIFDataInputFile`, which supports data input in a SIF specific format or `SIFCoordinates` needed in simulations that have a topological structure.
The next level of abstraction is the Abstract Model Engine (AME) which implements the basic EMS model (figure 1). On this layer, we have a generic `effector`, a generic `medium` and and a generic `sensor`. This layer also contains the abstract base classes for percepts (`percept`) or for SIF agents (`SIFAgent`).
The top level layer of the library, the Generic Object Layer, contains classes that can be used directly by an application designer. The layer is subdivided into classes for effectors (e. g. `mover`, `turnLeft`, etc.), sensors (e. g. `viewer`, `audioSensor`, etc.) or percepts (e. g. `visionPercept`, `audioPercept`, etc.). The objects of these classes are then combined to form EMS systems as described earlier.
A special effector/sensor combination is `controlPadEffector` and `controlPadSensor`. To interactively control an agent in the tile world, it can be equipped with a `controlPadSensor`. If the graphical user interface (which is also modeled as agent in our framework) has a `controlPadEffector` which is linked to the agents Sensor, the user can interactively activate the effectors of the controlled agent and observe its perceptions. This form of user interface enable efficient debugging of the multi-agent application.

GOL	Generic Object Layer
AME	Abstract Model Engine
Kernel	SIF specific data structures

Fig. 8. The Layers of the SIF Library

3.2 Example

In this section, we present a simple example to demonstrate, how an agent acting in a virtual environment built with the SIF library is implemented. Assume a tile world as described earlier in this paper. The agent can move around in this world by turning and moving one step ahead and it perceive the state of its environment by a vision sensor as described before. The task of the new agent will be to search for an object (ideally another agent) in its environment and to follow that object. To achieve this task, the agent takes a random walk in its environment until it perceives another object. When it

has found one, it tries to keep this object in its visual range by moving into the direction of the object.
In the constructor for the new agent, the effectors and sensors the agent has are defined (line `35 - 38`). The newly created objects automatically register with their respective media in order to connect the agent to the simulation.
In each cycle of the simulation engine, the `act()` method of the agent is called. This method is the link between an agent architecture and the simulation engine as it implements the decision procedure of the agent. Any agent architecture implemented in Java can be used as decision procedure in the `act()` method and thus interact with agents of different architectures.
We have implemented a round-robin scheduling scheme for the agent activation because the Java Virtual Machine for the Solaris operating system does not support preemptive thread scheduling. Currently, the simulation engine chooses a random permutation of the existing agents and requests their respective effector activations by calling the `act()` method. In later version, however, the agents are supposed to act asynchronously on their environment.
In its `act()` method, the simple visual reactive agent checks for new percepts in its perception queue (lines `47 - 50`, see figure 7) and uses the most recent percept to decide about its next action. If it is a percept from the vision sensor, the agent checks if the visible area of its environment and counts the number of items in this area. If the number of objects is greater then 1, the agent moves one step in the direction of the objects (line `66`). If the visual area is empty, the agents moves one tile further with a probability of 25% or it turns with a probability of 37.5% to the left or right, respectively.

```
1  package src.GOL.agents;
2
3  import SIF.kernel.frameAtPosition;
4
5  import SIF.AOE.SIFObject;
6  import SIF.AOE.SIFAgent;
7  import SIF.AOE.percept;
8  import SIF.AOE.sensor;
9
10 import SIF.GOL.effectors.mover;
11 import SIF.GOL.effectors.turnerLeft;
12 import SIF.GOL.effectors.turnerRight;
13 import SIF.GOL.sensors.visionSensor;
14 import SIF.GOL.percepts.visionPercept;
15 import SIF.GOL.frames.emptyFrame;
16 import SIF.GOL.frames.agentFrame;
17
18
19 /** This agent is supposed to follow other agents by means of the percepts it receives
20     from the vision sensor.
21 */
22 public class visualReactiveAgent extends example.src.myScenarioAgent{
23
24   private visionSensor vSensor;
25   private mover        moveForward;
26   private turnerLeft   turnleft;
27   private turnerRight  turnright;
28
29   private frameAtPosition[][] visionField;
30
31   public visualReactiveAgent( String name, String form, String color ){
32
33     super( name, form, color );
34     visionField = new frameAtPosition[5][5];
35     moveForward = new mover( this );
36     turnleft    = new turnerLeft( this );
37     turnright   = new turnerRight( this );
38     vSensor     = new visionSensor((SIFAgent)this,-2, -2, 5, 5);
39   }
40
41   public void act(){

    percept p;
    int     counter = 0;

    super.act();
    if( not(queue.isEmpty()) ){
      do{
        p = queue.next();
      }while( not(queue.isEmpty()) );

      if( p != null ){
        if( p instanceof visionPercept ){
          Point size                 = ((visionPercept) p).getSize();
          frameAtPosition [][] field = ((visionPercept) p).getContents();

            for( int i=1; i<= size.x;i++ )
              for( int j=1; j<= size.y;j++ )
                if( field[i][j] != null )
                  if( not(field[i][j].getFrame() instanceof emptyFrame ))
                      counter++;
        }
      }
    }
      if (counter !=0)
        moveForward.activate();
      else
        if (Math.random() <=.25)
          moveForward.activate();
      else
        if (Math.random() <=.5)
          turnleft.activate();
        else
          turnright.activate();
   }

/** The method that will be run, when the worldserver starts the experiment.*/
 public void initialize(){
   vSensor.initialize(); // send signal to sensors to download information
 }
}
```

4 EMS/NP

Currently, the class of agent architectures that can be used in a SIF simulation is limited to architectures implemented in Java. This is due to the fact, that the actual implementation of the basic interaction model uses method invocations to accomplish the communication between the effectors, sensors and media. To overcome this limitation, we are developing a TCP/IP network protocol which implements the basic model and which enables us to integrate non-Java architectures in a SIF simulation by simply implementing the protocol in the respective goal language. The basic protocol primitives are shown in figures 9 through 11.

request	param	response
REGISTER	<TYPE>	<ID> | NULL
QUERY		
	SUPPORTED_EFFECTORS	<EFFECTOR_TYPE_LIST>
	WORLD_DIMENSION	<0>|<1>|...|<n>
	WORLD_SIZE	<SIZE_1><SIZE_2>...<SIZE_n>
	WORLD_TYPE	<WORLD_SPECIFICATION>
ACTIVATE	<ID> <DATA_LANGUAGE> <CONTENT>	

Fig. 9. The EMS network protocol: Effector/Medium

request	param	response
REGISTER	<TYPE>	<OK> | <NOTOK>
QUERY		
	SUPPORTED_SENSORS	<SENSOR_TYPE_LIST>
	WORLD_DIMENSION	<0>|<1>|...|<n>
	WORLD_SIZE	<SIZE_1><SIZE_2>...<SIZE_n>
	WORLD_TYPE	<WORLD_SPECIFICATION>
POLL		
		<DATA_LANGUAGE> <CONTENT>

Fig. 10. The EMS network protocol: Sensor/Medium

request	param	response
SENSOR_DATA	<DATA_LANGUAGE> <CONTENT>	

Fig. 11. The EMS network protocol: Medium/Sensor

In our implementation, each medium is a single process which waits for incoming requests on a TCP/IP port. The host addresses and the port numbers for different media can either be given to the agent by the designer or they can be requested from the *Medium Directory Service* which has the same fixed port number on all host computers.

In figure 9, the protocol primitives for the effector/medium communication are shown: To register with a medium, the effector sends a REGISTER message together with its type. If the medium accepts the effector, it returns a unique id with is later used by the effector to activate its respective action. If the medium decides to reject the registration of the effector (e. g. because it does not support the effector specific actions), it returns NULL. Prior to the registration process, the effector can request a list of effectors supported by the medium. This feature is useful if an effector is capable to simulate various effector types.

In some cases, an effector or a sensor needs to know about characteristic features of the virtual world. For example, to navigate in virtual world, the effector needs to dimension of the world and the sizes of the respective dimensions. Additionally, some information about the type of topology can be given.

The dimension of the world in which the medium exists is needed whenever an action or a perception requires to specify the location of some object. In a 0-dimensional world, all objects are located at the same position, i. e. the position of the objects is not important. An example of a 0-dimensional world is the EMS system for communication. In an n-dimensional world, the location of an object is given by a vector with n elements; for example in the grid

world described earlier, the location of each object is described by a vector with two elements.
The type of a topology describes how the dimensions of the position vector are linked with each other. In a finite world, for example, the agent cannot move any further on an axis as the maximum coordinate whereas in a torial world, the agent reenters the world on the opposite end of an axis whenever it exceeds the maximum coordinate.
The protocol between a sensor and a medium is essentially the same as the effector/medium protocol except for the exchange of sensory data: if the sensor is an *active* sensor (which is indicated as part of its `TYPE` information passed during the registration process), it needs to execute a `POLL` request whenever it wants an update of its current perception. If, on the other hand, the sensor is *passive*, it waits for incoming percepts sent by the medium using the `SENSOR_DATA` request. In either case, the perception of the sensor is described by its `DATA_LANGUAGE` which indicates the data format (e. g. Java Serialized Objects, First Order Formulae or simply ASCII Strings) and the actual content of the message. The separation of the goal language and the protocol language is a common practice which is, for example, also used in KQML (Finin and Fritzson, 1994).

5 Conclusion

In this paper, we have presented a general model for agent interaction based on effectors, media and sensors that can be used to describe various forms of interaction in a generic manner. Furthermore, we have shown how this separation of entities involved in an interaction leads to a modular design of simulation testbeds based on EMS systems and demonstrated that the approach facilitates the integration of agents with different architectures (InteRRaP, MECCA and SIFAgents) in the same virtual environment. Finally, we have also shown how the basic model can be implemented in different ways (e. g. method invocation or network communication using TCP/IP) yielding to the most appropriate implementation for a specific scenario.
In our ongoing work, we are currently evaluating the network protocol presented in section 4. Especially, we are looking for a minimal set of features (e. g. size, topology, etc.) which can be used to characterize a maximum number of virtual worlds. We are also working on a scenario which models resource allocation and resource usage processes in multi-agent systems. The goal of this project is to investigate the usefulness of explicitly modeling social relations among the interacting agents.

6 Acknowledgments

The model presented in this paper developed during the fruitful discussions at the DFKI, especially with Petra Funk, Christian Gerber, Michael Schillo and Christoph Jung.

Bibliography

[Bürckert et al., 1998]Bürckert, H.-J., Fischer, K., and Vierke, G. (1998). Transportation scheduling with Holonic MAS, the TeleTruck approach. In *Proc. PAAM98*.

[Denzinger, 1995]Denzinger, J. (1995). Knowledge-based distributed search using teamwork. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 81–88, San Francisco, CA.

[Durfee and Montgomery, 1989]Durfee, E. H. and Montgomery, T. (1989). MICE: A flexible testbed for intelligent coordination experiments. In *Proceedings of the 1989 International Workshop on Distributed Artificial Intelligence (IWDAI-89)*.

[Finin and Fritzson, 1994]Finin, T. and Fritzson, R. (1994). KQML — a language and protocol for knowledge and information exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, pages 126–136, Lake Quinalt, WA.

[Funk et al., 1997]Funk, P., Gerber, C., Lind, J., and Schillo, M. (1997). Social interaction framework—a generic testbed for social agents. Technical report, DFK. To Appear.

[Hewitt, 1985]Hewitt, C. E. (1985). The challenge of open systems. *Byte*, 4(10).

[Lux and Steiner, 1995]Lux, A. and Steiner, D. (1995). Understanding cooperation: an agent's perspective. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*.

[Müller, 1996]Müller, J. P. (1996). *The Design of Intelligent Agents: A Layered Approach*, volume 1177 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

[Noda, 1995]Noda, I. (1995). Soccer server: a simulator of robocup. In *Proceedings of AI symposium 1995*. Japanese Society for Artificial Intelligence.

[Pollack and Ringuette, 1990]Pollack, M. E. and Ringuette, M. (1990). Introducing the Tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 183–189, Boston, MA.

[Shen and Barthes, 1995]Shen, W. and Barthes, J.-P. (1995). Dide: A multi-agent environment for engineering design. In *Proceedings of the ICMAS95*, pages 344–351.

[The VRML consortium, 1997]The VRML consortium (1997). *The Virtual Reality Modeling Language, ISO/IEC DIS 14772-1*. unknown.

[van de Velde, 1997]van de Velde, W. (1997). Co-habited mixed realities. In Hattori, F., editor, *Proceedings of the IJCAI'97 workshop on Social Interaction and Communityware*, Nagoya, Japan.

[Wooldridge and Vandekerckhove, 1994]Wooldridge, M. and Vandekerckhove, D. (1994). MYWORLD: An agent-oriented testbed for distributed artificial intelligence. In Deen, S. M., editor, *Proceedings of the 1993 Workshop on Cooperating Knowledge Based Systems (CKBS-93)*, pages 263–274. DAKE Centre, University of Keele, UK.

The EMS Model

Jürgen Lind

TM-98-09
Research Report