MODELING MOBILE AGENTS

Adelinde M. Uhrmacher, Petra Tyschler, Dirk Tyschler
Department of Computer Science, University of Ulm
D-89069 Ulm, Germany
http://www.informatik.uni-ulm.de/ki/personen/au.html
e-mail: lin@informatik.uni-ulm.de

KEYWORDS: Variable Structure Models, Mobile Agents, Distributed, Parallel, Discrete Event Simulation, Java

ABSTRACT

Agent-oriented software implies the realization of software components, which are mobile, autonomous, and solve problems by creating new software components during run-time, moving between locations, initiating or joining groups of other software components. Modeling and simulating those multiagent systems requires specific mechanisms for variable structure modeling.

JAMES, a Java-Based Agent Modeling Environment for Simulation, realizes variable structure models including mobility from the perspective of single autonomous agents. JAMES itself is based on parallel DEVS and adopts its abstract simulator model. Simulation takes place as a sending of messages between concurrently active and distributed entities which reflect the model's current structure. Thus, modeling and simulation are coined equally by an agent-based perspective.

INTRODUCTION

The definition of agents subsumes a multitude of different facets (Wooldridge and Jennings 1995). Agents are reactive, deliberative or combine reactive with deliberative capabilities. Their activities are called autonomous, as they do not require a frequent interaction with humans (Shoham 1993). In the context of software design, agents are defined as software components, which are mobile, autonomous, and solve problems by creating new software components during run-time, moving between locations, initiating or joining groups of other software components (Genesereth and Ketchpel 1994).

JAMES, a Java-Based Agent Modeling Environment for Simulation, constitutes a tutorial test bed which is aimed at supporting experiments with Artificial Intelligence programs as agents. Conceptualized with a layered architecture, its simulation layer provides the means for the description and a distributed parallel execution of variable structure models, including multiple mobile agents. For that purpose, JAMES reuses and combines concepts of distributed systems and parallel DEVS (Zeigler 1990; Chow 1996) with ideas of endomorphy and variable structure modeling (Uhrmacher and Zeigler 1996).

In many domains the systems' dynamics exhibits structural changes (Uhrmacher 1996). But there seems hardly to be a domain, where the phenomenon of structural changes, i.e. the appearance, disappearance, and movement of entities is so salient a feature as in the area of multiple agents interoperating on the Internet (White 1997). This is reflected in recent developments in programming paradigms (Lubomir *et al.* 1996), specifications (Milner *et al.* 1992; Asperti and Busi 1996), and frameworks (Lange and Chang 1996), all of which support the design of software systems as systems consisting of agents which interact among each other, and whose configuration and neighborhood is continually changing.

In this paper we will explore the simulation layer of JAMES and how the realized approach can be employed to simulate multiple mobile agents.

BASICS

On the simulation level, the model design in JAMES resembles those of parallel DEVS (Zeigler 1990; Chow 1996).

JAMES distinguishes between atomic and coupled models. An atomic model is described by a set of

input events, a state set, a set of output events, an internal, respectively external, transition function dictating state transitions due to internal events, respectively external input which is defined as bags over simultaneously arriving input events, an output function which generates bags of events as outputs, and a time advance function to determine the time of the next internal event. The communication of each model with the external "world" is determined by its input and output events. Its state is described by a set of variables. The state space of an atomic model might be structured to reflect aspects of deliberative agents, e.g desires, intentions, and beliefs. In the context of modeling mobility only the "beliefs" obtain some relevance. As in AgedDEVS (Uhrmacher and Arnold 1994), "beliefs" about itself and the world around it affect a model's behavior and determine its possibility to initiate structural changes.

A coupled model is a model consisting of different components and specifying the coupling of its components. Its interface to its environment is given by a set of external input, respectively output events, a set of component models, that may be atomic or coupled and the coupling which exists among the components. Coupled models allow modular and hierarchical modeling. As in DEVS, coupled models have no behavior, i.e. transition functions, of their own.

As models are allowed to produce their internal events in parallel a tie-breaking function is not necessary. However, conflicts between simultaneously executing internal and external transition functions might arise. In JAMES, they are dissolved by invoking the external after the internal transition function, by default (Chow 1996).

A SCENARIO

The scenario is motivated by the strategy employed in Telescript, which aims at supporting the design of autonomous programs, that are capable of physically moving through communication networks in order to perform specific service tasks. Thereby, the mobility of agents is motivated by the desire to utilize efficiently geographically distributed services. To reduce network communication resulting form client-server interactions, autonomous agents are dispatched to the remote server site where they perform the necessary interactions locally. After completing the task they carry the result back with them to the original client. Thus, instead of a back and forth of interactive negotiation between client and server, one round trip of an agent does the job (White 1997).

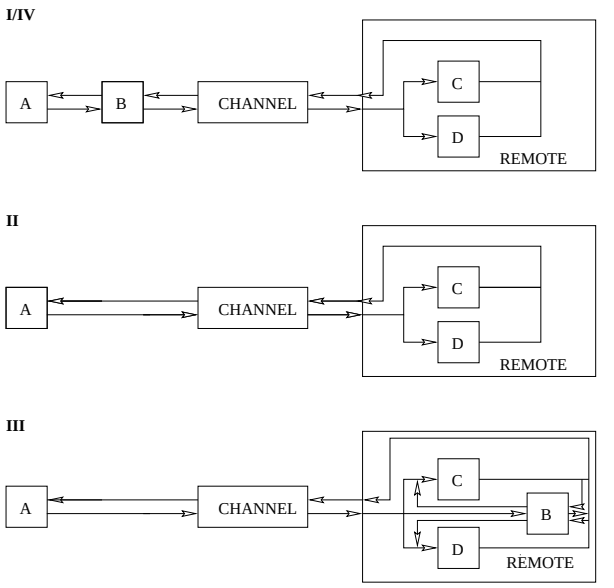

Figure 1: The example.

We will alter this scenario slightly. The remote site is represented as the coupled model REMOTE, which is connected to other models by the atomic model CHANNEL. The latter represents a frightfully slow communication channel. REMOTE comprises two atomic models which offer certain functionalities. In our example A asks B to solve a task. Both are located at the same site, The time they need for interaction can be neglected, i.e. the input and output ports are directly connected. Entities might function as clients and servers at the same time as some tasks necessitate the interaction with other "servers", whereas other tasks do not. In our example, B is server to A and a client of D and C. We assume that the task A asked of B necessitates a close interaction with the atomic models in REMOTE, so B decides to move there (Fig. 1).

MOBILITY IN JAMES

In JAMES, the environment whose structure can be accessed directly by a single model is locally restricted, thereby constraining conflicts that might arise between different agents manipulating concurrently their environment.

Models are able to create new models, to add existing models, to delete themselves, to remove them-

selves, and to determine their own interaction with their environment. To initiate structural changes outside their own range, they can launch corresponding requests to models which have direct access to the region of interest.

Those requests are handled like other messages. They are created by the output function, put into the output bags, are sent via the coupling structure, are collected within the input bags of the corresponding model, and evaluated by its external transition function. Thus, conflicts can be resolved locally by the model responsible for the region of interest. This rather distributed and delegating strategy in realizing variable structure models presumes certain conventions.

- Atomic models are equipped with default methods to commit suicide, to remove themselves from their environment, to create or add other models within the embedding coupled model and to alter their interaction structure with their environment.
- They are aware of their names which identify them uniquely within the universe of models.
- The initiation of structure requires usually knowledge about the surroundings of an agent, e.g. the creation of a model requires not only information about the model which is to be created, but also knowledge about its embedding within the existing environment. This knowledge is part of the model, i.e. it belongs to the model's beliefs, or the knowledge is communicated together with the request to initiate structural changes.
- Atomic models are equipped with methods which evaluate and execute requested structural changes. However, these default reactions to a request can be suppressed. The freedom to "decide" whether to follow a certain request, and the knowledge, i.e. beliefs, about itself and its environment distinguish active agents from more "reactive" models in JAMES.

Mobility implies that an agent (Fig. 1, **I**) disappears in one location and is sent to another (Fig. 1, **II**), where it reappears overall unchanged, with the same internal state and identity (Fig. 1, **III**). Mobility includes the adding and removing of models from coupled models, the change of interaction structure and in addition the possibility to send references, i.e., names within messages.

The output function is responsible for creating events, an agent's action respectively reaction to its environment. Events are encoded as messages which the name of the proposed addressee might be part of. Based on this information, recipients of the message decide whether the message concerns them or not. It is not used for actually sending the message to another model - to which models the message is sent depends on the coupling structure only. After a message is put into an output bag, a model has no control where the message is actually sent to. Whether models forward messages which are not addressed to them depends on the modeling as forwarding happens not automatically.

```
void outputFunction(State state) {
  ...
  if (state.phase == "move") {

    chargePort("outChannel",
               (new AddMessage
                   ( toLocation,
                     myName,
                     withCoupling)));
  }
```

Figure 2: Sending the request - an extract of the output function of B

The movement starts with B creating a message where it asks C to add it to the coupled model. The message specifies the desired coupling in the new location as well (Fig. 2). The message is sent via the output function. Within the internal transition the model changes its state and removes itself (Fig. 3). In JAMES, as in other DEVS derivates, output function and internal transition function are intrinsically connected. The output function is invoked just before the internal transition function, which serves our purpose just fine. In the moment the internal transition function is completed, the model B ceases to exist within the former coupled model and can be added to another coupled model. At any time it is ensured that a model is component of one coupled model, only.

The time the movement will take to complete depends on the CHANNEL which in our simplified model processes one message after the other with randomized delay (Fig. 1(II)).

The addressee C is responsible for adding B now in the context of the coupled model REMOTE (Fig. 4, Fig. 1(III)). C has only to include the class method for the external transition function. A default method

```
State internalTransition(State state) {
     ...
     if (state.phase == "move") {
        state.phase = "aftermove";
        beliefs.revise
           (new AddFact("Moved",
                        toLocation,
                        myName,
                        withCoupling));
        beliefs.revise
           (new AddFact("Removed",
                        oldCoupled,
                        myName,
                        oldCoupling));

        remove();
     }
     return state;
}
```

Figure 3: Updating the internal state and removing from the coupled model - the internal transition function of B

is triggered automatically through the arrival of messages, searching the input bags for requests, executing the required structural changes, and cleaning the input bags (Fig. 4). Structural changes are only executed if all information necessary is available either as part of a model's beliefs or as part of the received message. After B's job is completed B will migrate back, then A will be the addressee to which the request "to add B" will be sent (Fig. 1 (IV)).

SIMULATION

The purpose of JAMES is to support experiments with multiagent systems, which might be composed of several concurrently deliberating, e.g. planning or learning, agents. Therefore, models in JAMES are executed in parallel and in a distributed environment.

The simulation adopts the abstract simulator concept for parallel simulation developed by Zeigler and Chow (Zeigler 1990; Chow 1996). Following the DEVS tradition, active simulator objects are associated with model objects, i.e. *simulators* with atomic models and *coordinators* with coupled models. The simulation is realized as a communication between distributed concurrent simulator objects. The employed algorithm

```
class MyAtomicModel extends AtomicModel {
  ...
  State externalTransition
          (State state, double elapsedTime) {
    // use default handling for
    // structural changes:
    super();
    ...
    return state;
  }
}

class AtomicModel extends Model {
  ...
  State externalTransition
          (State state, double elapsedTime) {
    if structureChangeRequest() {
      changeStructure();
      // remove requests from input:
      cleanUpInput();
    }
    return state;
  }
}
```

Figure 4: Methods of atomic models in JAMES (extracts)

realizes a conservative strategy in parallel simulation, transitions do only take place when all necessary information has arrived. The realization of an optimistic strategy would require to define anti-messages that undo the effects of previous messages which might include also structural changes. Thus, the costs would likely exceed the gains of employing an optimistic strategy.

Besides the message types **, x, y* and *done* (Zeigler 1990), which signalize that the next internal event shall take place (***), that an input event, respective output event shall be processed (*x, y*), or a transition function has been completed (*done*), we introduce an additional message type, *struc2do*, which signalizes that and what kind of structural change takes place within the boundary of the coupled model. Same as *done, struc2do* signalizes that the external or internal transition function of a component has been completed. It contains the time of its next internal event and, in addition, a field which specifies the required change in terms of coupling and refers to components

that should be added to, or removed from the list of components. The coordinator is responsible for updating its components and the coupling, and to arrange the components according to their next internal event. Only the creation and deletion of models are actually done by the simulators. After the completion of the transition function a simulator notifies its coordinator to enforce the requested structural change.

E.g., the coordinator receives a *struc2do* which contains a notification that a model with name B shall be added, and that an interaction with C and D should be realized. The coordinator responses with updating its component and coupling list, it checks where B is located, and it sends the current time to the associated simulator of B to receive the time of the next internal event of the newly added component, and to make himself known as the simulator's new coordinator.

Whereas messages of type **, y, x* and *done* are exchanged between coupled and atomic models equally, *struc2do* is only sent from a simulator to its coordinator.

IMPLEMENTATION

Specific JAVA libraries provide the functionality for modeling and simulation in JAMES. As do other JAVA-based distributed simulation systems, JAMES employs the remote method invocation function of JAVA for a distributed execution of the simulation (Page *et al.* 1997). All nodes involved within the simulation are running the program JAMES. To initialize the simulation run includes the distribution of models with their associated simulators respectively coordinators, among the involved nodes. Currently this is done more or less arbitrarily. Simulators and coordinators which are located on one node are implemented as parallel threads. Each of the simulators and coordinators is executed sequentially. Simulators and coordinators located on one node communicate by direct message passing. The communication between simulators and coordinators that are located on different nodes is realized as synchronous notification-oriented message passing via the servers which are associated with each node (Colouris *et al.* 1994). Within the abstract simulator concept each coordinator "knows" how to reach its components and its "super" coordinator. Agent name and register services are structured according to the hierarchical message flow from and to coupled models. They are kept locally at the coordinators, e.g., a coordinator will typically need only the addresses of its component and its supercoordinator. However, in the rare moments of structural change, e.g. a model moves from one location to another, a server is known to all coordinators where the names of models and their addresses are registered.

MOBILITY AND VARIABLE STRUCTURES

The problem of describing and analyzing systems consisting of agents which interact among each other, and whose configuration or neighborhood is continually changing is addressed in a variety of approaches. In the context of discrete event simulation, variable structure modeling has gained a lot of attention during the last years (Barros 1996; Uhrmacher and Zeigler 1996). However, the focus has been on the creation and deletion of models and change of interaction, rather than on mobility. Most approaches support the initiating of structural changes top down the compositional hierarchy. The coupled model is responsible for initiating structural changes which are kept local to the coupled model (Barros 1996; Thomas 1996). They are not aimed at supporting the understanding of systems which emerge by the interaction and activities of autonomous and mobile agents. AgedDEVS, the sequential predecessor of JAMES, used also endomorphic agents, to initiate structural changes "bottom up". Invoked within the internal transition function, structural changes were only restricted by and directed to an agent's internal model, whose changes were immediately reflected in the environment. As only one agent at a time could initiate structural changes, a correct and repeatable execution of the model was guaranteed. In a distributed parallel setting more than one model might initiate a change of structure at a time. JAMES avoids conflicts by dividing the environment into barely overlapping regions which can be accessed by models. To initiate structural changes outside its boundary, agents have to turn to communication and negotiation. Conflicts which might still arise, e.g. one model wishes to delete its interaction with another model, whereas the other wishes to change it, are resolved by the coordinators which, for that purpose, are equipped with a set of simple conflict resolution rules.

The close relationship between variable structure models and mobility of agents becomes particularly visible when the mobility of agents is explored from an "organizational" rather than a "geographical" perspective. The latter raises different questions and challenges (Saphira 1997).

CONCLUSION

Based on parallel DEVS and the concept of endomorphy the simulation layer of JAMES provides a framework for the description and analysis of mobile agents. Since we decided to realize movements as the sending of references, i.e. names, models are removed and added from coupled models, instead of deleted and created. This facilitates the handling of mobility from a modeller's perspective. However, mobility might result in simulators that belong to the same coordinator but are executed in totally remote sites. As usually models in a coupled model are supposed to communicate more frequently, and so will the associated simulator objects, one might wish to keep the simulators which are associated with the components of a coupled model as closely located as possible. Due to combining the abstract simulator concept of parallel DEVS with variable structures in a distributed setting, the scenario we choose becomes applicable to the simulation in JAMES itself. In the moment we will explore more complex applications which involve not only but also deliberative agents, the distribution of simulators has to adapt itself dynamically according to the work load of the involved processes, according to the behavior of a model - a multitude of reactive models will not use as much time as one single planning agent to process - and according to the frequency of interaction. Thus, during simulation models and simulator objects will temporarily be moving through the network of involved nodes for reasons of efficiency.

REFERENCES

A. Asperti and N. Busi. Mobile petri nets. Technical Report UBLCS-96-10, University of Bologna, 1996.

F.J. Barros. The dynamic structure discrete event system specification formalism. *Transactions of the Society for Computer Simulation International*, 13(1):35–46, 1996.

A.C. Chow. Parallel devs: A parallel hierarchical, modular modeling formalism. *SCS - Transactions on Computer Simulation*, 13(2):55–67, 1996.

G. Colouris, J. Dollimore, and T. Kindberg. *Distributed Systems Concepts and Design.* Addison Wesley, New York, 1994.

M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

D.B. Lange and D.T. Chang. Ibm aglets workbench. White Paper, IBM, Japan, 1996.

B.F. Lubomir, F. Munehiro, and M.B. Dillencourt. Distributed computing using autonomous objects. *IEEE Computer*, 29(8):55–61, 1996.

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes I. *Information and Computation*, 100(1):1–40, 1992.

E.H. Page, B.S. Canova, and J.A. Tufarolo. Web-based simulation in simjava using remote method invocation. In *Winter Simulation Conference*, Atlanta, 1997.

International SRI. *Saphira Software Manual, Saphira Version 5.2*, 1997.

Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

C. Thomas. *Ein objektorientiertes Konzept zur Modellierung und Simulation komplexer Systeme.* Number 20 in Fortschrittsberichte. VDI, Dsseldorf, 1996.

A. M. Uhrmacher and R. Arnold. Distributing and maintaining knowledge - agents in variable structure environments. In *Proc. of the 5th Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, pages 178–184, San Diego, CA, December 1994. IEEE-Press.

A. M. Uhrmacher and B.P. Zeigler. Variable structure models in object-oriented simulation. *International Journal on General Systems*, 24(4):359–375, 1996.

A. M. Uhrmacher. Variable structure modelling - discrete events in simulation. In *Proc. of the 6th Annual Conference on Artificial Intelligence, Simulation and Planning in High Autonomy Systems*, pages 133–140, San Diego, CA, March 1996.

J.E. White. Mobile agents. In J.M. Bradschaw, editor, *Software Agents.* MIT Press, Menlo Park, Ca, 1997.

M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

B. P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models - Intelligent Agents and Endomorphic Systems.* Academic Press, San Diego, 1990.