Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

LINKING SPATIALLY EXPLICIT PARALLEL CONTINUOUS AND DISCRETE MODELS

Boleslaw K. Szymanski
Gilbert Chen

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180, U.S.A.

ABSTRACT

This paper advocates the use of mobile agents for linking simulations running on different computers. A Mobile Component approach is proposed to enhance reusability of existing simulations and to improve efficiency of component based simulations of complex systems. A basic unit of the mobile component simulation is a simulation server with a communication interface to mobile agents. Each mobile agent links and coordinates component's execution. We used this approach to implement a combined Lyme disease simulation. It consists of a partial differential equation based continuous simulation and parallel discrete event simulation with explicit space representation. The performance of this implementation is presented to demonstrate the feasibility of the Mobile Component approach. In addition, a process-port model of simulation is discussed. Its implementation allows efficient linkage of simulation servers, if they are programmed in a simulation language supporting the process-port model. We finally show that the performance of the Mobile Component approach could be significantly improved by using compiler techniques to eliminate overhead of communication among simulation servers.

1 INTRODUCTION

An exponential growth in processor speed and network bandwidth in the last decade enables development of large simulations of unprecedented fidelity and computational complexity. In particular, it has become feasible and computationally efficient to create large-scale simulation by integrating several existing models. Component-based modeling technique is a convenient way of building integrated simulations for distributed and heterogeneous computational environments. A large simulation can be partitioned into a number of components that interact with each other. The interaction itself is relatively simple and therefore easy to describe. Building each component may require knowledge from specific disciplines, which makes modeling a whole system by a single team difficult.

Another advantage of component-based approach is that it facilitates simulation reusability. Existing models can easily be integrated with new ones that are built from scratch. Since verification and validation of a simulation is a very tedious and difficult task, using a simulation already verified is very valuable. Hence, modern simulation design paradigms support reusability as a means of reducing the cost and complexity of the design of large simulations.

In the next section, we briefly review High Level Architecture (HLA), the best-known component approach for simulation, and point out its advantages and disadvantages. Section 3 introduces mobile agent technology, which allows autonomous programs to dynamically link distributed simulations. In Section 4, we propose a new approach, referred to as Mobile Component approach, to coordination of distributed simulations based on mobile agent technology. In Section 5, we present a case study of the simulation of Lyme disease. In Section 6, we introduce a process-port model that results in efficient linkages within a mobile component simulation. Section 7 concludes the paper.

2 THE HLA

Developed by the U.S. Department of Defense, HLA (see U.S. Department of Defense, Judith et al. 1998) provides software architecture for integration of a wide variety of simulations. A major design goal of HLA was to provide a modeling mechanism for reusing existing simulations so that the cost and time required to create new ones can be decreased. The U.S. Department of Defense (DoD) has mandated that the HLA should be used across all classes of simulations within the DoD. In addition, the HLA has been adopted as the standard for distributed simulation by the Object Modeling Group, and is currently under consideration for becoming IEEE standard 1516.

The HLA standard is composed of three parts:

1. *HLA rules* that describe design principles and constraints on HLA-compliant federates (simulations) and the entire federation. A federation is a combined simulation system that is created by integrating federates.
2. *HLA object models* that describe the critical aspects of simulations and federates shared across a federation.
3. *An HLA interface specification* that describes the runtime services provided by the Runtime Infrastructure (RTI) to federates and by federates to the RTI. The Runtime Infrastructure is responsible for executing a federation.

In HLA, reusability is understood much broader than the common notion of reusability in the software engineering community (Ernest 1998). Runtime information of all elements within a simulation (federate) is collected by a federate wrapper and available to the RTI. The RTI acts as a communication bus. All communication between federates is implemented on a subscription basis and must go through the RTI. A federate may subscribe to a specific object class and may have the RTI notify that federate whenever a new object of this class is discovered. The subscriber federate may also request that it will receive updates to the subscribed object whenever its attributes are changed. The publishing federate notifies the RTI whenever an object's attribute value changes, which are then sent to the subscriber federates.

There are two problems with this approach. First, the HLA requires that the simulation is built in accordance with its framework. Although in theory it is possible to wrap up a legacy simulation as an HLA-compliant federate, a considerable amount of effort must be put to program the wrapper. Often writing such a wrapper is more difficult than writing a federate from scratch. Second, the subscribing and publishing scheme imposes a tremendous burden on communication bandwidth (Wayne and Gerald 1999). Some techniques, such as Remote Data Filtering (William et al. 1998), have been proposed to reduce this communication overhead. Yet, the fact that federates cannot transmit messages directly to others prevents efficient communication implementation.

3 MOBILE AGENT

A close examination of the idea behind the HLA approach reveals that the underlying communication scheme of the HLA is based on a Client/Server paradigm. The six groups of services defined by the HLA standard can be clustered into two classes. Some services, called federate-initiated, are provided by the RTI and invoked by federates. In contrast, RTI-initiated services are provided by federates and invoked by the RTI. The RTI has an interface named *RTIambassador* that defines all federate-initiated services. Similarly, a federate must implement a *FederateAmbassador* interface for all RTI-initiated services. As a result, both the RTI and each of the federates may act as a client and a server at the same time.

One limitation of the Client/Server paradigm is its lack of flexibility. The set of services provides by the server is defined statically. Thus, it is impossible for a server to meet unforeseen requirements without modifying its interface. This is exactly the reason why the HLA designers have put a significant amount of effort to define enormous runtime services that enable cooperating a wide variety of simulations nowadays. Whether or not the HLA can meet all future needs is of course unknown.

Another limitation of the Client/Server paradigm is that it results in inefficient communication. Each request invoked by a client might require multiple two-way messages. For simulations that need intensive inter-component interaction and that execute on limited-bandwidth networks, this often is a serious inefficiency.

The described above limitations of the Client/Server paradigm result from the design in which the server but not the client has the code to perform the service. Therefore, the client must send a request to the server that then executes the corresponding service and sends the result back to the client. If the code would be able to migrate from host to host, the disadvantages of Client/Server paradigm would have been overcome. This is the idea behind Mobile Code paradigms of which the following are the three most important ones (Carlo and Giovanni 1997, Antonio et al. 1997):

1. *Code on Demand paradigm* in which the client fetches the service code from the server and executes that code. This method can save bandwidth if the size of code is smaller than the size of data produced by that code.
2. *Remote Evaluation paradigm* in which the client possesses code that performs the service. However, the client needs to send this code to the server for execution. In this solution, the main goal is increased flexibility. Binding the service code to the client instead of keeping it with the server makes modification and upgrades of the service easy. Another advantage of this approach is simplification of balancing the load between servers, because each server can execute any service under such an arrangement.
3. *Mobile Agent approach* that can be viewed as a generalization of Remote Evaluation. In this approach, there is no distinction between a client and a server. All hosts on the network behave the same way. An agent, consisting of the executable code and the intermediate state, can migrate from

host to host and interact with other agents. As a result, both the issue of communication efficiency and the issue of load balance can be addressed.

Thanks to its effectiveness, the Mobile Agent approach has been an active research area in the past few years, while Code on Demand and Remote Evaluation approaches attract few researchers. Various Mobile Agent systems are now available, most important among them are D'Agent (Robert et al. 1996) and Aglets (Aglets Software Development Kit). Applications of Mobile Agent systems include network management (Alan et al. 1999), information retrieval, electronic commerce, and others. Mobile Agents has also received much attention in the simulation community. They can be applied to the distributed simulations in two ways:

1. *Data Filtering* (Linda et al., Linda et al. 1999). An approach has been proposed that uses mobile agents to link a continuous simulation with a discrete event simulation. By sending a mobile agent to the remote data server to perform a remote computation such as filtering the data, transmission of large data sets is avoided. Only the relevant data, that often contain a relatively small part of the full data, is sent back.
2. *Mobile Simulation*. To reduce the variance of results, a simulation must be run for a long time, often repeatedly. As a result, the size of the simulation code is often small compared to the amount of data produced by such a run. Therefore, when linking multiple simulations, it is beneficial to move all simulation code together to a powerful multiprocessor instead of running them on separate hosts. Using this approach can significantly reduce the overhead of communication among simulation components. For instance, if the TCP/IP based message passing can be replaced by the shared-memory message passing, time savings can be very significant. Moreover, this overhead can be totally eliminated if some compiler techniques are used to reconfigure the simulation, as we will describe in the later sections.

As simulations are becoming larger, networks of computers are increasingly attractive platforms to execute them. The Mobile Agent technology provides an efficient way of implementing simulations on such a platform, making the Mobile Agent an increasingly important direction in simulation research. In the next section, we present a novel approach, named Mobile Component Approach, which applies Mobile Agent technology to component-based simulations.

4 MOBILE COMPONENT APPROACH

Currently, two challenges for a good component-based modeling technique are:

1. ease of linking existing simulations, and
2. facilitating collaborations in building new simulation.

These two goals are what the HLA wants to achieve, too. However, in our opinion, the HLA falls short of both of them. In case of the first goal, the HLA does allow easy linking of existing simulations, but such simulations must be built in accordance with the HLA framework. It remains unclear whether the HLA can meet the second goal because two main disadvantages of the HLA are its inflexibility and inefficiency.

We propose Mobile Component approach to solve the above problems. In this approach, mobile agents link together simulation components. The main benefit of such arrangement is that the mobile agent can choose the best host to execute on. The communication flow between the agent and the component simulations may not be symmetric. Some components may have more intensive communication with the linking agent than the others. Therefore, an efficient solution is to put the agent and the component that requires maximum bandwidth on the same host. This solution can be modified and the agent can dynamically migrate to other host, if it detects changes in communication flow during a simulation run.

Another important feature of Mobile Component approach is the concept of a simulation server. Each simulation is required to define an interface that provides sufficient functionality to link it with other components. Mobile agents interact with the simulations only through such interface. In this way, security issue is circumvented. By limiting the access to components, this paradigm distinguishes itself from the majority of the current mobile agent systems in which security is the important issue that has attracted a lot of researchers.

An interface consists of methods and events. Mobile agents invoke methods while simulations trigger events. Both methods and events define an argument list and a returned data type. An interface may have multiple implementations to maximally utilize the capabilities of the underlying computer hardware. For instance, if the simulation and the agent are located on the same host, shared-memory implementation can be used to avoid the relatively slow speed of TCP/IP.

Currently, the research on Mobile Component approach focuses on the first goal, which is to facilitate linking of existing simulations. We will discuss how it meets the second goal in Section 6.

5 CASE STUDY: LYME DISEASE SIMULATION

Lyme Disease is prevalent in the Northeastern United States. People can acquire the disease by coming in contact with a tick infected with the spirochet, which may transfer into the human blood, causing an infection. Even though the most visible cases of Lyme disease involve humans, the main infection cycle consists of ticks, mice and deer. If an infected tick bites a mouse or a deer, it becomes infected. The disease can also be transmitted from an infected mouse to an uninfected, feeding tick. Ticks undergo three life stages: they are born as larvae, transform then into nymphs and finally mature into adult ticks. Larvae and nymphs prefer feeding on mice, while adult ticks bite only mammals, mainly deer. The seasonal cycle of the disease, and the duration of the simulation, is 180 days, starting in the late spring. This time is the most active for the ticks and mice. For example, during that time mice are searching for nesting sites and may carry ticks a considerable distance. The cycle of Lyme disease is shown in Figure 1.

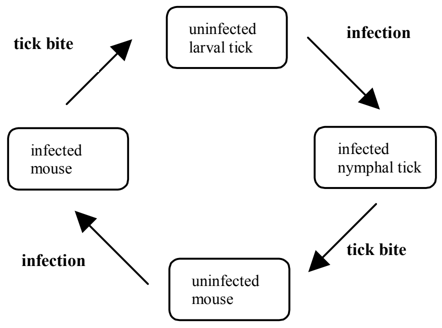

Figure 1: The Cycle of Lyme Disease

We have already built a parallel discrete event simulation (PDES), in which deer and mice are modeled as individuals and space is discretized into a grid of locations (Ewa et al. 1996). Ticks are treated as a "background", a distinct feature of each space location. The density of ticks is computed independently at each location. Hence, it changes in discrete steps even between neighboring locations. Another simplification in this model is an assumption that ticks are totally immobile themselves and spread over the space only by being carried around by the animal on which they feed. The simulation uses the optimistic protocol. To reduce the overhead incurred by rollback, it employs Breadth-First Rollback (Ewa and Boleslaw 1997) that limits the number of events that need to be rolled back in response to a straggler.

A more accurate model requires that tick density changes continuously in space and ticks themselves spread out by crawling in response to the level of crowding at each point of space. In such a model, ticks density is described by a set of partial differential equations (PDE). Incorporating a PDE solver directly into the parallel discrete simulation could be extremely difficult, because it changes fundamentally the nature of discrete event simulation; typically, the whole program would have to be redesigned.

Alternatively, a separate PDE solver can easily be built using either a standard or customized numerical package. Then, the PDE solver and the parallel discrete event simulation need to work collaboratively in order to simulate the Lyme disease more accurately. Using Mobile Component approach, both of them are viewed as simulation servers that need to define an interface for mutual collaboration. Then, an agent can be built to synchronize them, as shown in Figure 2 (Gilbert et al. 2000).

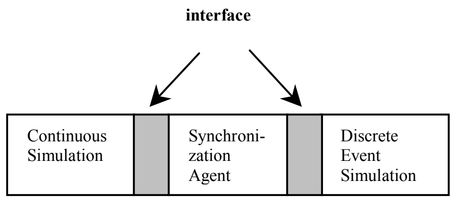

Figure 2: Linking Two Simulations

5.1 Modified Discrete Event Simulation

To reflect the mobility of ticks, the original discrete event simulation has been extended with an interface through which the tick state can be changed. Whenever the discrete event simulation needs to access the density of ticks, it triggers an event indicating that the tick density has changed in a particular location. That event is then passed to the interface, and is received by a mobile agent which can then decide whether to respond immediately or later, based on the timestamp of the event.

```
interface DES
{
methods:
        void init();
events:
        void tickbite(int proc_id, int x, int y,
                    double time, TICK& tick);
        void tickdrop(int proc_id, int x, int y,
                    double time, TICK& tick);
        void tickbite_undo(int proc_id, int x, int y,
                    double time, TICK& tick);
        void tickdrop_undo(int proc_id, int x, int y,
                    double time, TICK& tick);
};
```

Figure 3: Interface of the Discrete Event Simulation

5.2 Continuous Simulation

There are five types of ticks in the Lyme disease simulation: uninfected adult ticks, infected adult ticks, susceptible nymphs, infectious nymphs and questing larvae. To describe the tick population, we use a reaction-diffusion model. A reaction term summarizes spatially localized processes of birth, death, and when applicable, developmental advance and infection transmission. The parameters of the reaction terms are independent of spatial location. Diffusion terms involve the second-order partial derivatives. Diffusion approximates biological dispersal of ticks in response to the overpopulation.

To solve the above partial differential equations, we chose the fully discrete finite difference method which discretizes both in time and space dimensions. Thus, the continuous domain of the equations is replaced by a discrete mesh of points and the derivatives are replaced by finite difference approximations. In addition, the PDE solver needs to be able to rollback simulation time because it is linked with an optimistic discrete event simulation whose events may be executed out of the temporal order.

```
interface CS{
methods:
      void init(int x, int y);
      void write(int x, int y, tick_type t, double density);
      void read(int x, int y, tick_type t, double& density);
      void forward(double time);
      void backward(double time);
events:
      void forward_complete();
      void backward_complete();
};
```

Figure 4: Interface of the Continuous Simulation

5.3 Synchronization Algorithm

The synchronization between continuous and discrete event simulations uses approximations on the simulated time. The mobile agent that is responsible for the synchronization keeps track of the simulated time of the continuous simulation. If a discrete event arrives with a timestamp falling into the range between current and the next simulated time of the continuous simulation, then this event is processed immediately. If the event timestamp is greater than the next step continuous time, the event must be stalled for later execution. The mobile agent advances the continuous simulation into the next step only when all simulated times in discrete processes have past the next continuous simulated time.

5.4 Experiment Results

In our experiments, the parallel discrete event simulation is based on MPI and runs on a 16-node IBM-SP2 machine. Another standalone program running on the IBM-SP2 provides the interface to the parallel discrete event simulation. The continuous simulation running the PDE solver executes on a 12-node SGI Origin 2000 shared-memory multiprocessor. The PDE solver uses Pthread library to distribute the computation over multiple processors. In addition to the computation processors, an additional communication processor is assigned that runs the interface opened by the PDE solver.

5.4.1 Discrete Event Simulation as a Simulation Server

As described earlier, the original discrete event simulation is modified to enable a mobile agent to change ticks state variables. To demonstrate feasibility of the Mobile Component approach, we built a mobile agent whose only function is to drive the discrete event simulation. This agent simply records the density of ticks without changing them, except in the initialization phase where certain amounts of ticks are distributed over the two-dimensional space.

This agent is written in the Java-based Aglet system developed by the IBM Tokyo Research Laboratory. It can either run on the remote computer from which the user gets access to the simulation sever, or migrate to the IBM SP2, on which the computer simulation server resides. Table 1 shows a significant speed difference between these two options. The parallel discrete event simulation uses four processors.

Table 1: Comparison of Simulation Execution Times with a Mobile Agent Running on Different Computers

Agent host	Execution Time
Remote computer	134 seconds
IBM SP2	74 seconds

This experiment shows how useful a mobile agent can be. Suppose a biologist uses the discrete event simulation to study dynamics of Lyme disease. She might want to add or remove ticks at selected points in space and the simulated time. The traditional approach would require that the biologist have access to and understanding of the source code. He would have to modify, recompile and run the source code. This is an error prone and difficult process. In contrast, using our approach, the biologist only needs to modify the mobile agent described above. The user is completely separated from the internal details of the simulation.

5.4.2 Linking Two Simulations Together

In our first attempt to link continuous simulation with discrete event simulation, we used the Aglet system to build a mobile agent, as illustrated in Figure 2. The communication between the agent and the interfaces is implemented in TCP/IP. However, the preliminary results were very disappointing. The simulation speed was extremely slow. A simulation run that ends at the 80th day of simulated time takes 1932 seconds, whereas in our earlier effort to link together the same simulations, the same computation took only 52 seconds.

We first suspected that this outcome was caused by the use of JAVA programming language that could be up to ten times slower than C/C++. So we rewrote the agent in C++ (it was not a mobile agent in a strict sense, but we decided to focus on speed not on mobility at this point). The execution time decreased to 1320 seconds, indicating still very slow implementation. This indicated that the TCP/IP communication was the source of the bottleneck. Hence, we replaced the TCP/IP communication between the agent and the continuous simulation running on SGI Origin 2000 with the shared-memory interprocess communication entirely within SGI machine. The result showed a great improvement, the execution time dropped to 289 seconds.

However, the communication between the agent and the discrete event simulation is still implemented in TCP/IP. Unfortunately, IBM SP2 is not a shared-memory computer. And more, it seems that the version of MPI currently installed on our IBM SP2 does not support message passing between two programs running on the same processor. Thus, just to demonstrate how fast this simulation can run, the interface provided by the discrete event simulation was moved to the SGI Origin 2000. The agent uses shared-memory message passing to access both interfaces. The execution time improved further into 117 seconds.

The experiment shows that in the component based approach the communication among different components might become the bottleneck that degrades the performance greatly. Efficient communication is the key to an efficient implementation using this approach.

6 COMMENTS ON MOBILE COMPONENT APPROACH

The experiment with linking two different simulations shows that the Mobile Component approach has the ability to reuse existing simulations with little extra programming effort. However, the best result achieved by the agent approach is still twice slower than the approach based on direct connection between components. This performance gap is caused by the overhead induced by communication between the agent and the continuous simulation. While the directly connected approach accesses the tick density through memory references, the agent approach uses shared-memory based message passing. Therefore, an important question related to the Mobile Component approach is whether a more efficient linkage is possible.

Efficiency is a fundamental issue in computer science that in our case has two aspects. First, there is a speed with which a program executes. Second, there is the cost of program development. In many cases, these two aspects are contradictory. For instance, the component-based approach is a convenient method to build a large simulation, however, it introduces considerable overhead incurred by the communication between components. This problem can be overcome by using of a process-port model and a new simulation language that implements this model. Then, compiler techniques can be utilized to eliminate the component communication overhead.

6.1 A Process-Port Model

The Process-Port Model is an extension of the classical logical process view in PDES community (Richard 1990). Each component is viewed as a logical process with a number of ports through which the process interacts with others.

A legacy simulation can be easily wrapped up as a Process-Port model by providing an interface. Notice that every method or event in the interface has a corresponding port. For instance, a method corresponds to a port that receives all arguments of the method and sends out only those arguments passed by reference.

Table 2: Comparison of Different Implementations of the Linking Agent. In the Directly Connected Approach (Gilbert et al. 2000), there Doesn't Exist a Specialized Agent. Instead, an Extra Communicating Thread in the Continuous Simulation is Responsible for Cooperating with the Discrete Event Simulation. It Uses Memory References to Access Tick State Variables, and Uses TCP/IP to Interact with Discrete Event Simulation.

	Source Language	Communication between Agent and Continuous Simulation	Communication Between Agent and Discrete Event Simulation	Execution Time (seconds)
Directly Connected	C++			52
Mobile Component Approach	Java	TCP/IP	TCP/IP	1946
	C++	TCP/IP	TCP/IP	1320
	C++	Shared-memory	TCP/IP	289
	C++	Shared-memory	Shared-memory	117

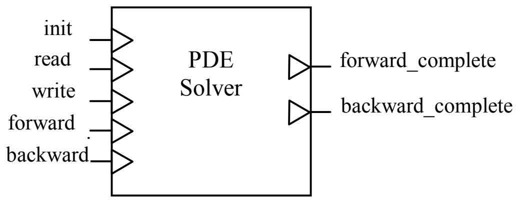

Figure 5: A Process-port View of the Continuous Simulation

6.2 A Language for Process-Port Model

It is beneficial to have new simulations written in a simulation language that enforces the Process-Port model. The detailed discussion of such a language is beyond the scope of this paper. We only outline here its major features:

1. Such a language may employ a communicating extended finite state machine (CEFSM) as the implementation of the Process-Port model. The CEFSM model enables the compiler to exploit the data and control flow information.
2. The language should facilitate mobile computing. A program written in this language should be able to migrate across a network of processors equipped with the language compiler.
3. Prior to execution, a program written in this language may pass through a configuration phase in which the compiler optimizes away unnecessary message passing overhead between processes.

6.3 Unifying Simulation Server with Agent

Both the simulation server and the mobile agent can be programmed using the new simulation language. They can freely roam on the network and look for the most suitable computers for their execution. After configuration, they could be merged as a single program that can execute with maximum efficiency.

The users still should be able to program the simulation in other programming languages. In such case, the Mobile Component approach would degrade to a simple linking approach that doesn't provide the most efficient linkage. However, such flexibility may be convenient for some users for various reasons.

6.4 Comparison with the HLA

As stated above, the HLA is an approach using a bus communication. The RTI serves as a bus into which simulations conforming to the standard can be easily "plugged". The disadvantage is quite obvious; the RTI tends to be a serious bottleneck that degrades the performance when the simulation becomes large.

In contrast, the Mobile Agent approach is hierarchical. A number of processes can be grouped into a composed process that behaves as a simple process.

However, the Mobile Federate Approach is by no means a substitute for the HLA. Rather, it is an alternative to it. Both approaches address different problems. For the HLA, it is the interoperability that concerns the designers most. For the Mobile Federate Approach, the main consideration is the efficiency. There might be cases where one approach is preferable to the other.

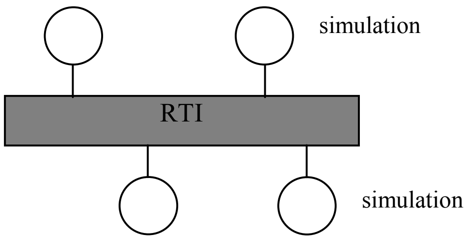

Figure 6: The HLA Approach

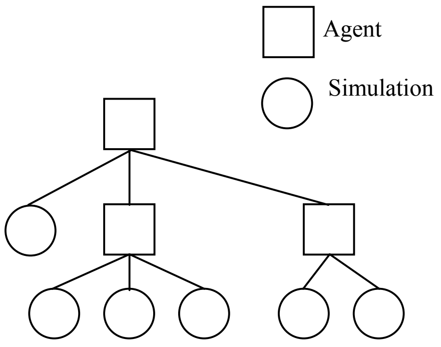

Figure 7: The Mobile Component Approach

7 CONCLUSION

The Mobile Component approach has been motivated by the attempts to link continuous simulation and discrete event simulation in order to create a more accurate model for Lyme disease. The mobile agent proved to be a useful method for linking different simulations. With the concept of a simulation server, the role of mobile agent in collaborating simulation has been clarified. Such clarification seems an important step towards wide spread use of distributed simulations. The Process-Port model suggests a novel approach to efficient component-based simulations. Its usefulness needs to be verified by future research.

ACKNOWLEDGMENT

This work was partially supported by the NSF Grant KDI-9873138. The content of this paper does not necessarily reflect the position or policy of the U.S. Government – no official endorsement should be inferred or implied.

REFERENCES

Alan Bivens, L. Gao, M.F. Hulber and B.K. Szymanski, 1999. Agent-based network monitoring. *Proc. Autonomous Agents99 Conference, Workshop 1, Agent Based High Performance Computing: Problem Solving Applications and Practical Deployment*, Seattle, WA, pp.41-53.

Aglets Software Development Kit. IBM Tokyo Research Laboratory. <`http://www.trl.ibm.co.jp/aglets/`>.

Antonio Carzaniga, Gian Pietro Picco and Giovanni Vigna, 1997 Designing distributed applications with mobile code paradigms. In *Proceeding of the 1997 International Conference on Software Engineering*.

Carlo Ghezzi and Giovanni Vigna, 1997. Mobile code paradigms and technologies: a case study. *Mobile Agents: First International Workshop.*

Ernest H. Page, 1998. The rise of web-based simulation: implications for the high level architecture. In *Proceedings of the 1998 Winter Simulation Conference*.

Ewa Deelman, Boleslaw Szymanski and Thomas Caraco, 1996 Simlulating lyme disease using parallel discrete event simulation. In *Proceedings of the Winter Simulation Conference*.

Ewa Deelman and Boleslaw Szymanki, 1997. Breadth-first rollback in spatially explicit simulations. In *Proceedings of the Workshop on Parallel and Distributed Simulation*, 1997

Gilbert Chen, Boleslaw, 2000 K. Szymanski and Thomas Caraco. Multiparadigm simulations in modeling spread of lyme disease. In *2000 European Simulation Multi-Conference*.

Judith S. Dahmann, Richard M. Fujimoto and Richard M. Weatherly, 1998 The DoD High Level Architecture: an update. In *Proceedings of the 1998 Winter Simulation Conference.*

Linda Wilson, George Cybenko, David Lynch, Bruce Cushman-Roisin and Boleslaw K. Szymanski. KDI: next-generation agent-based distributed simulation, <`http://www-nml.dartmouth.edu/KDI`>.

Linda Wilson, George Cybenko and Daniel Burroughs, 1999 Mobile agents for distributed simulation. In *High Performance Computing Symposium*.

Richard M. Fujimoto, 1990. Parallel Discrete Event Simulation. *Communications of the ACM*, 33 (10) pp.31-53, 1990.

Robert Gray, David Kotz, Saurab Nog, Daniela Rus and George Cybenko, 1996. Mobile agents for mobile computing. *Technical Report PCS-TR96-285*, Dept. of Computer Science, Dartmouth College, May.

U.S. Department of Defense. *High Level Architecture*. Defense Modeling and Simulation Office <`http://hla.dmso.mil`>.

Wayne J. Davis and Gerald L. Moeller, 1999. The High Level Architecture: is there a better way?. In *Proceeding of the 1999 Winter Simulation Conference*.

William S. Murphy Jr. and Galen D, 1998l. Aswegan. High Level Architecture remote data filtering. In *Proceedings of the 1998 Winter Simulation Conference*.

AUTHOR BIOGRAPHIES

BOLESLAW K. SZYMANSKI is a Professor of Computer Science. He has been at Rensselaer since 1985. In the past he was also affiliated with the University of Pennsylvania, Aberdeen University (U.K.) and Warsaw Polytechnic (Poland). Dr. Szymanski received Ph.D. in Computer Science from National Academy of Sciences in Poland in 1976. He is an IEEE Fellow, ACM National Lecturer, Editor-in-Chief of Scientific Programming. He has edited and contributed chapters to several books and authored over 150 research papers in journals and conference proceedings. His research has been supported by NSF, DARPA, ONR, ARO, NASA and industry. His current research focuses on pro-active network management, distributed and parallel computing and computational models of evolution, epidemics and biomedical systems. He has been also involved in econometric modeling, algorithm design and information retrieval.

GILBERT CHEN is a graduate student in Computer Science department at Rensselaer Polytechnic Institute. He received BS and MS degrees in Electrical Engineering from Tsinghua University in 1995 and 1998 respectively. His email address is <`cheng3@cs.rpi.edu`>.