Proc. Computer Networks and Distributed Systems Modeling and Simulation, CNDS'03, Orlando, FL, January 2003, p. 37-44

Component-Based Simulation and Agent-Based Brokering: Towards Ad Hoc Simulations in Crisis and Emergency Management

Boleslaw Szymanski and Gilbert Chen
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180

Linda F. Wilson
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755-8000

Keywords: distributed simulation, simulation software tools, components, software agents, mobile agent systems

ABSTRACT

Crisis and emergency management systems often require the use of large simulations to predict future developments of the simulated phenomena. For example, floods and forest fires are dynamic phenomena with devastating consequences. Simulations for these disasters require large amounts of information, including current and future weather data and the topography of the local terrain. In both cases, the ability to create an efficient simulation to support responses to a crisis is useful only if the development time is short enough and the simulation is fast enough to be used in real-time crisis management. This paper introduces our work in developing software tools for integrating simulations and sensor data in support of crisis and emergency management. In particular, it presents three basic challenges for developing these tools and discusses our approaches for meeting these challenges.

INTRODUCTION

A large-scale simulation is often developed as a single monolithic system that integrates all of the desired functionality and uses predefined data resources. It is difficult to build such a complicated system from scratch, particularly when parallel processing is involved. Early in the design stage, the decision must be made about the type of model to be used. There are two fundamental types of simulation models. The first type uses partial differential equations to describe the simulated phenomena and the corresponding simulation solves numerically the appropriate set of partial differential equations. The other type of model is based on discrete event simulation. Either the first principles are applied to individual entities comprising the simulation or events are scheduled into the future of such entities based on an inter-event time distribution. The development of either type of system is very time-consuming and requires significant expertise in a diverse number of disciplines, from mathematics and statistics needed to formulate the correct model, to computer science to develop an efficient and reliable implementation of the model, to application-specific disciplines relevant to the simulated phenomena. The complexity is even higher when the model being developed is multi-disciplinary or involves interactions at different temporal and spatial scales, as is often the case for the modern simulations. As a result, such simulations cannot be developed rapidly to respond to quickly changing, unpredictable, and urgent needs arising in crisis and emergency management.

For example, consider floods resulting from the combined effects of spring thaws and heavy rains. Overflowing rivers damage or wash out bridges and roads, destroy property, and endanger human life. The damaged transportation system breaks the well-established paths of traffic, making it difficult to help victims and repair damage. The weather, topography of the terrain near and around the river beds, and the current transportation system topology, including roads and railroads, all influence which area will be endangered and all impact the possible remedies. The coordinated preventive and relief efforts require the ability to predict the development of the floods in response to different actions of the rescuers and to the ever-changing natural conditions.

A forest fire is another example of a highly dynamic and devastating phenomenon that changes and evolves depending on the weather, winds, topography of the local terrain, and responses of the firefighters. In those cases, and in many others, the ability to create an efficient simulation to support responses to a crisis is useful only if the development time is short enough and the simulation fast enough to be used in real-time crisis management. Furthermore, the simulation should be able to incorporate all available data from various sources, from satellite maps to ground crews to sensors and embedded systems. In short, such a simulation must be capable of linking to a variety of available simulation components and real-time data sources.

This paper describes our work developing software tools that enable the rapid creation of a spatially explicit simulation of evolving phenomena and the efficient execution of such a simulation so the result can be used in real-time to predict future developments and to evaluate alternative remedies and countermeasures. To achieve these goals, we are advocating component-based simulation that are created independently of each other and can be linked together into a simulation with the help of software agent systems, including mobile agents. In recent years, web-based simulation has exhibited explosive growth in the simulation research community. Our effort builds on and expands the recently developed techniques to create new capabilities beyond those found in conventional simulation technology and to provide integration with real-time data sources, such as wireless sensors.

CHALLENGES

Web-based simulations rely on the Internet in two fundamental ways. First, the Internet is a large repository of proven simulation components and simulations that can be reused by the new simulations. Second, the Internet can provide large amounts of computing resources for the execution of newly-created simulations. In this second role, the Internet environment is by definition parallel and distributed, yet compared to the typical parallel and distributed simulation (PADS) environment, the communication delays can be an order of magnitude larger and also more unpredictable (i.e., the distribution of the delays will have much higher variance than in the parallel architecture environment). In the first role, the development of the simulation components on the web is fully decentralized, and as such does not have any centralized coordinator or oracle with the knowledge of currently available resources. Hence, there is a need for providing information services and infrastructure that enables advertising the existence, searching for, and linking together simulations and data sources over the Internet. Consequently, to reach our goal of developing software tools for integrating existing simulations and data sources into an efficient real-time simulation, we need to meet three basic challenges, as described below.

- **Scalable Resource Brokering Techniques**: There is a need for building a framework for efficient registering, translating, searching, and linking of different components, simulations, and data resources over the Internet. The scalable brokering service must provide necessary interfaces that will make the differences between formats, naming, and descriptors of services transparent to the participants. The broker must also be able to dynamically adjust its knowledge based on availability of the resources, current status of the network, etc. Finally, the broker should be distributed itself to ensure high reliability and scalability of its services.
- **Standard Interfaces and Linkage Techniques:** Each of the participating resources, from a single data resource to the most complex simulation, needs to encapsulate its interface in a standard way to enable collaboration and linkage. An important challenge is to identify those parts of interfaces that are simulation specific (i.e., which express semantics of the simulation itself and therefore can be standardized) as opposed to the parts that are application specific and therefore must be defined as general objects consistent with requirements of the languages used by the participating simulations. Likewise, as discussed later in this paper, there are several different techniques that could be used to link the web-distributed resources into a coherent integrated simulation. These techniques must be evaluated from the point of view of their overhead, accuracy, and applicability to different classes of linked components.
- **Efficient Composition Techniques:** The methodology is needed for developing suitable inter-simulation synchronization techniques and for implementing simulation engines that are needed to run the entire integrated simulation together. This methodology must include techniques that will address scalability, load balance, and efficiency of the execution. The issues that need to be researched include (i) classes of simulation engines needed to run the entire integrated simulation, (ii) use of compilation or interpretation of the linkages between simulations, (iii) dynamic load distribution among available computational resources, and others.

The following sections describe the approaches we are using to address these challenges.

SCALABLE RESOURCE BROKERING TECHNIQUES

Crisis and emergency management simulations require access to dynamically changing data from other sources such as sensors, datasets, or even other simulations. This data may come from multiple sources or even dynamic sources with sporadic availability. For example, a simulation predicting the severity of a flood needs rainfall and weather predictions (from weather simulations), current water levels (from sensors), and information regarding the existing drainage infrastructure (from databases).

Ideally, a simulation would interact with a "cloud"' of dynamically changing data and computational resources available on a network. Simulations and other resources may join or leave the cloud at any point in time. For example, consider a scenario in which some of the communication within the cloud occurs via a wireless network. A forest fire simulation could communicate with various sensors out in the field as well as with a weather simulation running at a remote site. Naturally, sensors located in hostile environments may communicate sporadically with the rest of the network.

We are developing a software framework to support such a simulation and data cloud. A key feature is that simulations and other cloud participants are designed with no prior knowledge of the details of other simulations and data sources. Our framework does not rely on relationships or data formats that are precompiled into simulation codes, and simulations do not need be rewritten just because capabilities have been added to or removed from the cloud. Instead, the framework facilitates the translation of information to permit seamless integration of multiple, independently-developed software systems.

Central to our framework is the concept of a resource brokering system to register, match, and link the resources available in the cloud. The broker acts as a database for cloud participants and their advertised services. It also allows the transparent replacement of one service with another that provides similar functionality, without requiring the participants to be written to conform to a particular standard. When a resource joins the cloud, it advertises its services to the broker. Later, when a data consumer searches for a particular service, it queries the broker, who is responsible for linking the consumer with its corresponding data producer. Whether or not the desired service is found, the broker maintains a standing request for that service. The standing request is useful if a match is not found initially or an incomplete service is found. If an unavailable service or a better service becomes available at a later time, the broker notifies the requesting consumer of this. If a desired service becomes unavailable, the data consumer is automatically switched to the next-best service. Thus, the broker maintains up-to-date information concerning the cloud participants and their availabilities.

A key part of the brokering system is the ability to describe services accurately and unambiguously so that consumers and producers of data can be matched correctly. This difficult problem is fundamental to the desire to have various simulations and data resources interacting seamlessly through the cloud. In order to minimize the impact on existing systems and remain highly flexible, it is preferable to describe the data being transferred, rather than define a common format for the data. Furthermore, it is necessary to be able to describe relationships between formats.

The brokering system must provide the following capabilities:

- Resource discovery and matching: The system must support mechanisms that allow for the dynamic discovery of entities in the network and the dynamic matching of data consumers with suitable data producers. Since resources may join or leave the cloud at any time, and multiple producers may be candidates for a particular consumer, the system must rank producers using user-specified measures and be able to switch to alternative producers as necessary.
- Resource database: The system needs a database to store resource descriptions obtained when a resource joins the cloud.
- Resource description and identification: The system must allow the individual resources to describe and advertise their attributes and capabilities. The system must also provide a mechanism to uniquely identify each resource at all times.
- Data conversion: The system must be able to handle data format and unit conversions, since participating resources are not required to produce data in a specified format.
- Communication: The system must provide a communication mechanism that enables resources to communicate with each other and understand each other. In particular, data consumers must be able to connect to and execute functions of producer resources.
- Resource management: For robustness and efficiency, the system must manage various resources in the system. The system must monitor the cloud participants and perform periodic housekeeping. It must also determine (e.g., using leasing) when desired producers are no longer available and notify corresponding consumers.
- Security and authentication: The system must ensure that the resources within the cloud are authenticated with suitable mechanisms in order to ensure the integrity of the system. The system must also provide suitable privacy mechanisms to protect sensitive data. Furthermore, malicious agents must not be allowed to disrupt the system.

As discussed in [7, 9, 11, 13-14], we have developed a basic brokering system using Sun Microsystems' Jini technology [12]. The Agent-Based Environment for Linking Simulations (ABELS) framework uses software agents to create the cloud environment described previously. ABELS components include the participating data resources, the brokering system, and generic local agents (GLAs) which interface the data resources to the cloud. As described in [14], the brokering system is distributed in order to be scalable and robust. We are currently developing a security framework, described in [10], that provides mechanisms for authentication, authorization, privacy, and integrity. Future work includes the development of improved matchmaking algorithms and mobile helper agents to perform sophisticated data queries and resource validation.

STANDARD INTERFACES AND LINKAGE TECHNIQUES

We begin by defining each resource in terms of its two essential parts, as follows.

- The *component semantics* that are defined by the component internal structure and program. Hence, the semantics are not easily accessible to the integrated simulation. Therefore, the semantics of each resource will be defined in a natural language and will be accessed, analyzed, and approved by the creators of the integrated web-based simulation. We do not plan to subject it to any automated processing.

- The *syntax of the data* that the resource can produce or consume and their *relation to real and simulated time*. This information will be defined in the *component interface*. Since the time, real or simulated, is an essential parameter of the simulation proper, as opposed to the data that are associated with the particular application of the simulation, the interface parameters that refer to time will have predefined types associated with them. The presence or absence of these predefined time types will be an essential part of characterization of the resource and its interface.

Component Classification

Based on the relation of the produced and/or consumed data to the real and simulated time, we first need to make a distinction between two worlds, the real world and the simulation world. In the simulation world, we identify the following three basic types of components [1].

Type I: Time-independent components that produce values on their output ports in response to the data provided in the input ports. Such components cannot send out an event without having received one. Any event produced by the processing of an arrival event contains the same time-stamp as that of the latter. An example is a data conversion component that can change distance measurements from inches to meters.

Type II: Simulated-time driven components whose interfaces include timers. Timers are used to forward a request of advancing the simulated time to the simulation engine. Components are notified when such a request can be granted. As a result, this type of component is explicitly aware of the simulated time but does not operate directly on it. An example of such a component could be a server that delays any received events for a certain amount of time.

Type III: Simulated-time controlled components that are simulations maintaining their own simulation clocks. Consequently, such components can contain a simulation engine or the entire simulation that is used as a component for the integrated simulation. In its simplest form, these components can receive a message only if such a message does not cause causality errors. The easiest way to ensure the causality constraint is to accept only those messages whose time-stamp is equal to or greater than the value of the simulation clock. Simulations using this approach are called conservative because they always maintain causality constraints between parallel components [4]. By extending the processing capabilities of components to enable them to process backwards in the simulated time, the components become capable of processing messages with any time-stamp or even anti-messages that request removal of effects of previously received messages. A simulation that uses such components is often referred to as an optimistic simulation because it allows parallel components to advance their simulation clocks without considering clocks of other components [5]. If such eager processing leads to a causality error, the erroneous computation is simply reversed. Hence, the time in optimistic simulations can flow in both direction and therefore is often referred to as *virtual time*. One extreme way of implementing optimistic simulation is to store each state of the component and retrieve it when needed, but such a solution would require an enormous amount of memory and would involve a lot of copying operations. The complementary implementation uses reverse processing to undo the forward processing with the minimum data storage. Efficient implementations use a mixture of both techniques.

In the real world there is only one type of components, real-time driven components that have output ports that produce real time data and input ports that consume real time data. An example of such a component is a repository of real time data gathered by sensors. For a real-time component to cooperate with any simulation components, adaptors must be used to translate the real time into simulated time or vice versa.

An interesting feature of the classification scheme presented above is that it enables hierarchical composition of components, in which a group of components can be treated as a single higher-level component of a certain type. We plan to develop an interface description language that will be used to define interfaces of the components in such a way that their automatic classification will be possible. These interfaces will become instrumental in advertising, localizing, linking, and optimizing the collaboration between resources on the Internet in the process of building real-time simulations.

The key to a component is the declaration of both input and output ports. Input ports, or functions, define what functionalities the component can provide. Output ports, on the other hand, define what functionalities the component needs to fulfill its function. Such a distinction between functionality providers and functionality consumers separates the development phase of components from the linking phase, allowing for more flexibility and more composability in our approach.

Port Classification

In most cases, the size of simulations created by combining components will necessitate the use of multiple processors to produce results in time for use in emergency and crisis management. Hence, the techniques developed for PDES (Parallel Discrete Event Simulation) are helpful. They include conservative and optimistic protocols defined earlier. Recently, we discovered the third class of PDES protocols based on a *lookback* [3], the ability of a component to change its past without affecting other components and therefore dual to the well known lookahead [4]. A component with a certain amount of lookback is able to process out-of-time-stamp order events (or stragglers) falling into its lookback windows. Hence, lookback allows a component to advance its simulated time more aggressively. We have shown that a lookback is able to exploit the intra-component parallelism. It is also more commonly observed than lookahead, which is the ability to predict the future, upon which conservative protocols largely depend. For optimistic simulations, lookback can be used to reduce the frequency of rollbacks. It is also of importance for simulation theory that, as we proved in [2], lookback enables the conservative simulation to circumvent the speed limit imposed by the critical times of events, which was previously thought impossible by many researchers [6].

We observe that the three classes of PDES protocols arise from different ways of manipulating the simulated time. In conservative protocols, the simulated time is treated in the same way as the physical time. An analogy between the simulated time of distributed systems and the physical time was given by Lamport [8]. Later, Jefferson proposed the notion of virtual time, in which the simulated time could be reversed, leading to the optimistic protocols [2]. Now, lookback allows us to ignore to some extent the time-stamp order of events imposed by the simulated time. These three classes of protocols actually correspond to the following three types of communication ports in simulation components:

- regular ports which send and receive only regular messages (also called positive messages in the optimistic simulation community) that carry events in the future of the receiving component,
- virtual ports which send and receive regular messages and anti-messages, and
- lookback ports which send and receive regular messages and stragglers.

Different types of ports may coexist in the same component. For example, consider an FCFS (first-come-first-serve) server with a lookback input port. It can deal with stragglers without difficulty, because such stragglers can be correctly processed by inserting them into the internal list sorted in time-stamp order. The output port of the FCFS server, however, is always of the regular type. The semantics of the FCFS server guarantee that it never outputs events in out-of-time-stamp order. A regular port can therefore be connected to this output port, simplifying the simulation modeling of other components that receive events from this FCFS server.

It is self-explanatory that a regular input port can be linked to a regular output port, while a virtual input port can be linked either to a regular output port or a virtual output port. With lookback ports, however, the connection rules become more complicated. For instance, an input port connected to two regular output ports should be of lookback type if these two ports belong to different Type III components. If they reside on the same Type III component, a regular input port can be used. The rules presented above, and similar ones that we plan to derive by analyzing all feasible port interconnections, can be checked during the configuration processing, providing partial correctness checking of the combined simulation.

Efficient Composition Techniques

On the theoretical level, it is possible to show that different types of components, as defined in the previous section, differ in their ability to link with other components. Observing such limitations decreases the number of possible solutions that needs to be evaluated. Once the configuration of the collection of component simulation is established, the type of linkages needs to be selected to ensure efficient execution. For components that are not in a tightly-synchronized loop, the distributed processing is sufficiently efficient. Examples of such loosely-connected components include simulations in which one phenomenon is not significantly affected by the other. For example, the flow and volume of water in the river is not affected by the structural changes in the bridge, hence the river flood simulation can just feed the flow simulation into the bridge simulation without a feedback. On the other hand, in the forest fire case, there is a direct impact of the fire's progress on local winds and vice versa. Such a feedback between interacting phenomena creates a tight synchronization between them. For components that are tightly synchronized, the efficient composition of the simulations may require the mobile component approach described below.

Mobile Component Approach

The combination of the component-based approach and mobile agents is a promising direction in distributed simulation. These two approaches are complementary; while the component-based approach makes it possible to assemble programs that are developed independently and that are distributed geographically, mobile agents improve the efficiency of co-execution by exploiting the locality and by changing the component location dynamically.

Computer networks exhibit a characteristic similar to that of computer architecture: the closer, the faster. Existing distributed environments are usually slow due to the low bandwidth of the network. It is true that in the future this will change, but at the same time the processor power will also improve at a rate perhaps comparable to that of the network speed. Therefore, the network may still remain a bottleneck for years to come, and code mobility is an efficient solution to this problem. Even if it is impossible to move all collaborating programs to one host, it is always advantageous to allow the programs to dynamically change their location to take advantage of the multiplicity of the network nodes with different computing power and communication bandwidth.

We propose a mobile component approach to address the efficiency issue in the integrated simulation. It aims to enhance the reusability of existing simulations and to improve the efficiency of component-based simulations of complex systems. A basic element of the mobile component simulation is a simulation server with a communication interface exposed to mobile agents. In fact, the only difference between a simulation server and a mobile component is that the former is immobile while the latter is mobile.

The use of mobile agents is justified by the observation that in order to reduce the variance of results, a simulation must be run for a long time. 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 components together to a powerful multiprocessor instead of running them on separate hosts. 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 data sharing, 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 discuss later.

Mobile agents can also be used to link together simulations that are immobile and therefore better modeled as simulation servers. The main benefit of using agents in such a case is that the mobile agent can choose the host on which to execute. The communication flow between the agent and the simulation servers may be asymmetric or dynamic. Some simulation servers may have more intensive communication with the linking agent than the others. Moreover, the available resources on a host may vary greatly from time to time. Therefore, an efficient solution is to exploit the code mobility by allowing agents to move freely across the network, always executing on the host that is optimal according to certain criteria.

Jini is a distributed computing environment where components can be integrated in a “plug and play” style [12]. At first glance, Jini seems a good candidate for our purpose. However, it is not, for two reasons. First, Jini is not a truly component-based approach. The discovery of a service is done by the client program itself. The service-finding procedure is embedded in the client code which prevents changes during integration. Our proposed component approach tries to look for an appropriate service and to link with it at a higher level than the client level (the configuration level), during the integration. Secondly, Jini is not a real mobile system, because much of the service code still resides on the server side, and the client is more likely to use a proxy object to access the service. Even if it is fully mobile, it does not address the efficiency issue of the linkage, for the linkage is still in the form of function calls, unlike the direct access method in the reconfigurable components that we will discuss later.

Efficiency of Linkage

Our earlier investigation of the efficiency issue in the component linkage and of the usefulness of the mobile component approach is based on the spatially explicit simulation model of Lyme disease. A set of PDEs (Partial Differential Equations) simulate the spread of ticks and a discrete event simulation models the movements of mice. To link the PDE solver with the discrete event simulation according to the idea of mobile component approach, we evaluated several different solutions.

In the first implementation, we used the Aglets system to build a mobile agent. The communication between the agent and the interfaces was implemented in TCP/IP. The results were very disappointing. The simulation ran about 40 times slower than the same simulation with direct links between the component simulations. To eliminate the language efficiency effect, we rewrote the agent, initially written in Java, in C++ and as a result the simulation speed nearly doubled, still leaving it 25 times slower than the direct link version. This pointed out to TCP/IP communication as a source of the slowdown. Hence, we replaced the communication between the agent and the continuous simulation by co-locating both on an SGI Origin 2000 with shared-memory inter-process communication. The resulting simulation showed a great improvement; the execution time dropped nearly five times compared to the C++ agent version. The still remaining five time slowdown compared to the direct linkage was caused by the communication between the agent and the discrete event simulation that was still implemented in TCP/IP. The discrete event simulation ran on an IBM SP2 which is not a shared-memory computer. So, to test how fast this simulation can run, the program that contains the interface for the discrete event simulation was moved to the SGI Origin 2000. The agent used shared-memory message passing to access both interfaces. The execution time improved further, as shown in Table 1.

This experiment suggests that in the component-based approach, the communication among different components might become the bottleneck that degrades the performance considerably. Efficient communication is the key to an efficient implementation using the mobile component approach.

Reconfigurable Components

The experiments of linking two different simulations showed 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 that of multiparadigm approach in which two parts are directly connected. This performance gap is caused mainly by the communication overhead between the agent and the continuous simulation. While the multiparadigm approach accesses the tick density through memory references, the mobile component approach uses shared-memory based message passing. This reveals a fundamental problem of all component-based approaches. While it is convenient to decompose the complex system into smaller subsystems, the boundaries between the subsystems that are created by decomposition incur significant communication overhead. This problem is often ignored, yet it is a serious limitation of this approach.

We propose to address this problem by using reconfigurable components, whose communication mechanism is subject to change either prior to the execution or during the execution. If, for example, two reconfigurable components reside on different hosts, they need to utilize some network protocol such as TCP/IP. When they move to the same host, they can communicate by direct memory references. Note that replacing the TCP/IP with shared-memory message passing does not require reconfiguration of the system because the replacement can be done by simply switching to a different communication medium. Direct memory references become possible only when multiple components can be merged into one program so that they will share the same memory space.

There are two methods to implement reconfigurable components. The first is to design a component description language that is able to model simulations and mobile agents. If a simulation and an agent on the same host need to communicate efficiently, they can be recompiled to form a single program within which they can directly access the variables owned by the other. Such recompilation, however, is rarely feasible for web-based simulations. The more practical solution is to require that each participating simulation server exposes an interface through which the agent can interact with it. Instead of loading the interface directly, the mobile component system places a driver between the interface and the simulation server. This driver is responsible for loading the interface and the incoming agent, both of which are written in the same component description language. Before execution, the incoming agent and the interface are merged by the driver into a single executable, in the form of a dynamic library, with the connected ports being eliminated. The driver then loads the generated dynamic library which will execute under the same address space with the simulation server. Thus, the final result is that the agent can access the component simulation information using direct memory references.

CONCLUSIONS

In this paper, we outlined challenges for web-based simulations and described our approaches to some of them. In particular, we described the novel approach to building components that can be reused in many different simulations. We also developed an agent-based brokering system that enables the location and matching of components on the web. Finally, we developed the linkage techniques that enable efficient composition of the components into an integrated simulation.

The components that we propose use standard interfaces that describe the types of data expected on each but also allow for additional, simulation-specific types such as timers and clocks that deal with simulation time. The most important feature of components is their independence from the sources and targets of data that are provided by the interfaces. The binding of ports represented by the interfaces happens in a stage separate from component definition and compilation. This stage, called configuration, defines the interconnection between ports and also optimizes the linkages between components.

Our current work on novel synchronization protocol for parallel execution of simulation components focuses on lookback, its types and uses in both conservative and optimistic parallel discrete even simulations [15]. We identified four types of lookback: direct strong, direct weak, universal strong and universal weak. These types differ in the level to which they can avoid rollbacks and anti-messages. Their definitions are based on the *impact time* of events. We also showed that all four types of lookback exist in, for example, Portable Communication System (PCS) simulation and presented the performance gains that can be achieved by each one used individually or in combination with others. Using lookback improves the performance of Web-based simulation.

There are two important results from our current work. One is negative but expected. The Internet cannot provide large computation resources for truly interactive simulations. An overhead of an order of magnitude or more makes such interactions infeasible. The important part in this negative result is *interactiveness* which is limited by the latency of the Internet communication. This latency is the result of laws of physics (the speed of light) and no future technological advances can diminish this problem. (If anything, such advances can only exacerbate the problem. The latency of communication from the East Coast to the West Coast in the United States, for example, is bounded from below by the speed of light, but the amount of computation that an average computer can do at the time equal to this bound is increasing exponentially with improving technology, so the same time delay cost more in terms of computer cycles as computer technology evolves.) Fortunately, the bandwidth grows even faster than the computational power of processors, so simulations that do not require interactions (no cause-effect feedback) can be efficiently executed over the Internet.

The second conclusion from our current work is positive as it clearly indicates that the Internet could (and in our opinion should) be used as a convenient repository for simulation components. Our brokering technique can reliably and quickly provide information about the sources of components and match data producer components with consumer components. Furthermore, it allows the transparent replacement of one producer component with another that provides similar functionality. Yet, there are still unresolved issues to reap the full potential of such a solution. The most important among them relate to the component semantics and information assurance. How to describe the function of a component in an abstract way, how to make sure that the component does what it advertises (no less to have trustworthy results, but also no more, to avoid Trojan horse type of attacks) are the most important questions. We plan to investigate such issues in the future.

Acknowledgement

This work was partially supported by NSF Grant KDI-9873138 and DARPA NMS Contract F300602-00-2-0537. 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

[1] G. Chen and B. K. Szymanski, "Component-Oriented Simulation Architecture: Toward Interoperability and Interchangeability," *Proc. Winter Simulation Conference (WSC2001)*, SCS Press, pp. 495-501

[2] G. Chen and B.K. Szymanski, "Lookback: A New Way of Exploiting Parallelism in Discrete Event Simulation", *Proceedings of the 2002 Workshop on Parallel and Distributed Simulation*, May 2002, IEEE Press, Los Alamitos, CA, pp. 89-96.

[3] G. Chen and B.K. Szymanski, "Lookahead, Rollback and Lookback, Searching for Parallelism in Discrete Event Simulation" , *Proc. SCS 2002 Summer Computer Simulation Conference*, July 2002.

[4] R. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, October 1990.

[5] D. Jefferson, "Virtual time", *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404-425, July 1985.

[6] D. Jefferson and P. Reiher, ``Supercritical speedup'', Proceedings of the *24th Annual Simulation Symposium*, pp. 159-168, April 1991.

[7] A. Kumar, L. F. Wilson, T. B. Stephens, and Sucharitaves, "The ABELS Brokering System", *Proceedings of the 35th Annual Simulation Symposium*, 63-71, April 2002.

[8] L. Lamport, "Time, clocks, and the ordering of events in a distributed system", *Communication of the ACM*, vol. 21, no. 7, pp. 558-565, July 1978.

[9] G. A. Mills-Tettey, G. Johnston, L. F. Wilson, J. M. Kimpel, and B. Xie, "The ABELS System: Designing an Adaptable Interface for Linking Simulations," *Proceedings of the 2002 Winter Simulation Conference*, to appear, December 2002.

[10] G. A. Mills-Tetty and L. F. Wilson, "Security Issues in the ABELS System for Linking Distributed Simulations", *Proceedings of the 36th Annual Simulation Symposium*, to appear, April 2003.

[11] J. T. Sucharitaves, L. F. Wilson, and A. Kumar, "The Generic Local Agent: Gateway to the ABELS System," *Proceedings of the 2002 High Performance Computing Symposium*, 147-154, April 2002.

[12] Sun Microsystems, web site, http://www.sun.com/jini.

[13] L. F. Wilson, D. Burroughs, A. Kumar, and J. Sucharitaves, "A framework for linking distributed simulations using software agents", *Proceedings of the IEEE*, vol. 89, no. 2, pp. 186-200, February 2001.

[14] L. F. Wilson, B. Xie, J. M. Kimpel, G. A. Mills-Tettey, and G. Johnston, "The Design of the Distributed ABELS Brokering System", *Proceedings of the Sixth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT)*, 151-158, October 2002.

[15] G. Chen and B.K. Szymanski, "Lookback Types and Uses," *Proceedings of the 2002 Workshop on Parallel and Distributed Simulation*, submitted in December 2002, available also as *Technical Report, Department of Computer Science, Rensselaer Polytechnic Institute*, web site, http://www.cs.rpi.edu/~szymansk/papers.html.

Table 1. Comparison of different implementations of the synchronization agent. Note that no agent exists in the multiparadigm approach; instead, an extra communicating thread in the continuous simulation is responsible for cooperating with the discrete event simulation. This thread uses memory references to access continuous simulation 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)
Multi-paradigm Approach	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