A Framework for Linking Distributed Simulations Using Software Agents

LINDA F. WILSON, MEMBER, IEEE, DANIEL J. BURROUGHS, STUDENT MEMBER, IEEE, ANUSH KUMAR, AND JEANNE SUCHARITAVES

Invited Paper

This paper presents the basic ideas behind the use of software agent technology for distributed simulation and data assimilation. A software agent is an autonomous computer program that operates on behalf of someone or something. A mobile agent has the ability to migrate during execution from machine to machine in a heterogeneous network, while a stationary agent executes only on the system on which it began execution.

To understand the role of agents in distributed simulation, note that simulations often operate on static datasets and data sources. Many simulations would produce more accurate results if they could access dynamically changing data from other sources, such as sensors or even other simulations. From the perspective of one simulation, other simulations are data resources, producing information possibly relevant to the past, present, or future of the system being modeled.

Software agents allow dynamic linking between distributed simulations and efficient monitoring of and access to remote data resources. Specifically, they conserve bandwidth, provide custom operations without precompiling or preloading, and adapt to support disconnected operations.

This paper describes our work developing a software agent-based framework for dynamically linking distributed simulations and other remote data resources. The framework called ABELS (Agent-Based Environment for Linking Simulations) allows independently designed simulations to communicate seamlessly with no a priori knowledge of the details of other simulations and data sources. We discuss our architecture and current implementation developed using the D'Agents mobile agent system.

Keywords**—***Data assimilation, distributed simulation, mobile agent systems, software agents.*

I. Introduction

Operational simulations are scientific, environmental, public infrastructure, and other computer simulations that operate continuously and online, estimating the current states of a system and predicting future states. Distributed data assimilation is the ability of a simulation to incorporate new sources of data dynamically while the simulation is operational. Examples include air traffic control, financial systems, and weather and ocean forecasting.

Many simulations would produce more accurate results if they could access dynamically changing data from other sources. Furthermore, many interactive simulations require dynamic data, possibly from multiple sources. From the perspective of one simulation, other simulations are data resources, producing information possibly relevant to the past, present, or future of the system being modeled.

This paper presents the fundamental framework for using software agent technology for distributed simulation and data assimilation, with a discussion of the D'Agents system [3], [7], [17]–[22], [29]. A software agent is an autonomous computer program that operates on behalf of someone or something. A stationary agent remains on the system on which it began execution, while a mobile agent has the ability to migrate under its own control within a heterogeneous network. Our framework uses stationary and mobile software agents to coordinate distributed operational simulations, efficiently communicate simulation data between the simulations, and retrieve data from other sources such as sensors and datasets. Such simulation agents will allow simulations to enter and exit a global simulation "cloud" (Fig. 1) asynchronously without requiring recompilation and constant recoordination among all participating sites and datasets. The networked data and simulation cloud consists of dynamically changing data and computational resources available on a network to a specific operational simulation. Note that the networked resources may consist of other simulations, datasets, active probes, and sensors.

As discussed later in this paper, software agents provide the required flexibility and efficiency to coordinate communication within the data and simulation cloud. Some existing technologies (e.g., GLOBUS [12]–[14], [16], CORBA [4],

Manuscript received February 20, 2000; revised October 15, 2000. This work was supported by the National Science Foundation under KDI Grant 9873138. This paper is based on work reported in [42].

The authors are with the Thayer School of Engineering, Dartmouth College, Hanover, NH 03755-8000 USA.

Publisher Item Identifier S 0018-9219(01)01745-5.

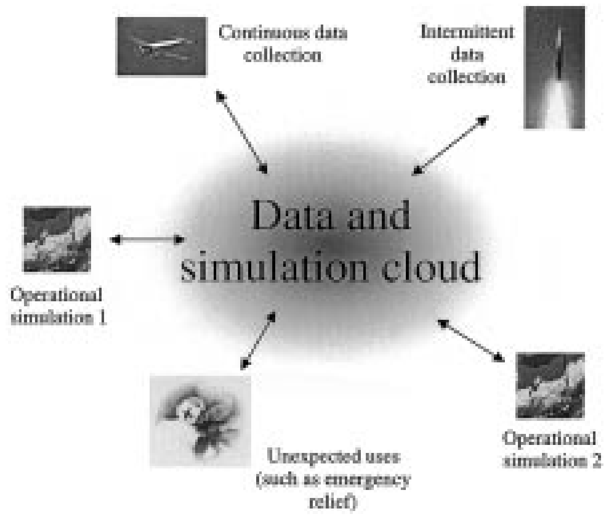
Continuous data collection
Intermittent data collection
Data and simulation cloud
Operational simulation 1
Operational simulation 2
Unexpected uses (such as emergency relief)

Fig. 1. This work is part of a project to research and develop simulation technology that allows distributed simulations, both discrete-event and continuous, to asynchronously and dynamically locate, assimilate, and generate data resources.

[5], [9], and RMI [10], [23]) rely on relationships, function names, and/or data formats that are precompiled into the simulation codes, but software agents allow such specifics to be moved from the simulations to the agent system. In the agent-based system, a simulation makes a general request to an agent, which is responsible for finding and extracting the needed information. Note that simulations do not have to be rewritten just because capabilities have been added to or removed from the cloud.

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.

Note that the simulations and other resources in the cloud do not have to be perfectly synchronized in time. However, the resources needed by a particular simulation must generally cover the same time frame or have previously generated data for the needed time period. In some cases, it may be possible for a simulation to wait for a data resource to reach the time specified for the desired information.

The next section describes background material related to this work. Section III discusses the use of software agents to communicate with distributed data resources. Section IV introduces our framework architecture while Section V discusses our current implementation. Section VI presents a search-and-rescue example that demonstrates the use of our system for a real-world scenario. The last section presents our conclusion and areas of future work.

II. Background

A. *Distributed Simulations and Data Resources*

Generally speaking, simulations operate by temporally and spatially propagating state variables according to the dynamics of the system being modeled. Those dynamics can be discrete or continuous, deterministic or stochastic. At any given time, the computed values of the state variables are estimates based on initial conditions, data updates, and evolution over time using the dynamics. Because of observation error, modeling error, discretization error, and environmental unobservables, the state variable values are only estimates of the true values. Viewed in this way, the propagated values are actually random variables with some underlying probability distribution. The probability distributions can in principle be propagated using the dynamics as well, but this is computationally feasible only in restricted model classes such as Gaussian and Poisson families, for example. The propagated state variable distributions capture the uncertainty of the current simulation estimate.

Once the uncertainty exceeds a certain tolerance (which is itself static or dynamic), external data must be retrieved and assimilated to reduce uncertainty. We are developing a general simulation infrastructure that can monitor this uncertainty in a computationally feasible way, trigger requests when the uncertainty becomes too high, and locate, retrieve, and ultimately assimilate appropriate data that reduces uncertainty to acceptable levels until the next such sequence of events is triggered. Such on-line, asynchronous assimilation of data is necessary for any class of real-world simulations that aspire to become operational in the sense of monitoring and predicting a real-world subsystem in real time.

New system software is required to support such functionality. Local interface agents are installed at each simulation site and data resource. This local interface allows the simulation or data resource to interact with other objects in the simulation cloud. The interface agent may send commands to other objects in the cloud or receive commands from the cloud and execute them locally. Simulations execute data request commands periodically, in response to data starvation, or when the simulation state estimate uncertainty is too high. The success of the simulation framework requires novel investigations into asynchronous simulation, self-defining data structures, security, distributed directory services, and efficient mobile coding for numerical computation. Fig. 2 presents a high-level view of this project.

There are several example applications that would benefit from this technology. Consider the National Airspace System (NAS), which includes all airports, airfields, airspaces, and connecting air routes in the United States. A typical day in the NAS consists of 35 000 commercial flights and 10 000 to 16 000 general aviation flights. This complex system is modeled by a discrete-event simulation. Such a model is useful in determining how various factors affect the overall system. For example, what happens if a snowstorm closes Chicago's O'Hare Airport? As many passengers discovered two years ago, a storm in Chicago can cause the cancellation or delay of flights around the country.

While the NAS model is very useful, it requires significant amounts of external information. In an existing simulation model [41], the Official Airline Guide provides information concerning scheduled commercial flights; sudden schedule changes are rarely included in the model because that infor-

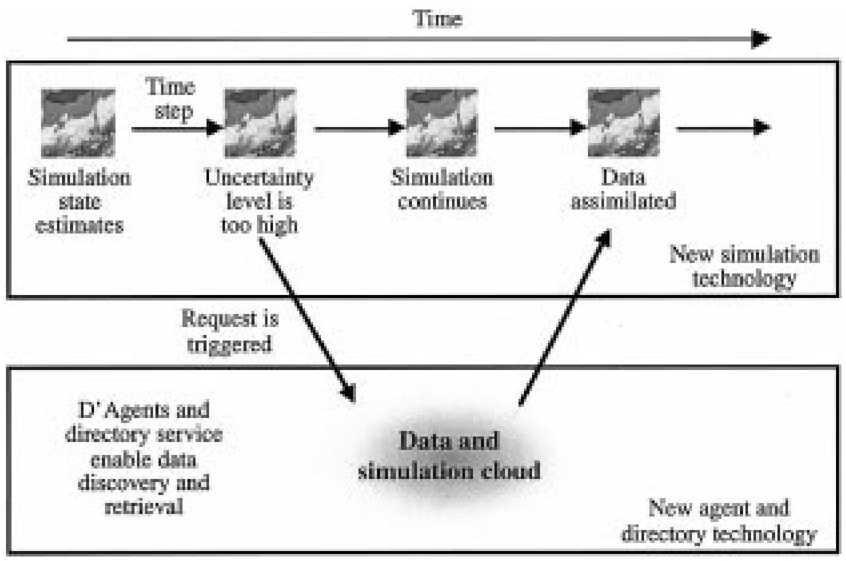
Time
Time step
Simulation state estimates
Uncertainty level is too high
Simulation continues
Data assimilated
New simulation technology
Request is triggered
D'Agents and directory service enable data discovery and retrieval
Data and simulation cloud
New agent and directory technology

Fig. 2. When a simulation reaches a certain level of uncertainty, it can spawn an agent to retrieve updated information from the data and simulation cloud.

mation would have to be manually inserted into the simulation. Thus, a simulation run today may be based on information that is out of date. Similarly, since weather effects must be inserted manually, only large-scale weather events (such as the Chicago snowstorm) can be simulated. The NAS simulation could more accurately include weather effects if it could communicate with a weather simulation running concurrently.

Modeling the spread of forest fires in real time is another area where such a technology might be applied. A fire is a highly dynamic, rapidly changing system, and as such the strategies used to control it must be just as dynamic. When a widespread forest fire occurs, simulations are utilized to determine the likely progress of the fire and the best course of action to take. However, these simulations are not able to accurately predict the nature of the fire over a long period of time. Periodically, current data must be fed back into the simulation to keep it on track. One method of providing such data is through the use of small, easily deployable sensors such as the ones described by Corr and Okino [6]. These sensors provide real-time measurements of the area of interest. Through a communication and interaction framework, this data may be used as input to a simulation that is providing a picture of the fire to those attempting to control it.

For another example, consider the operational use of coastal ocean models, in which real-time, mission-oriented computations utilize simulations and observations concurrently. Observational platforms include satellites, autonomous and remote-controlled vehicles, and advanced acoustic and optical instruments. It has been widely discussed in the ocean science community that data-assimilation techniques are needed to support real-time monitoring and prediction [32]–[34], [37]. The vision of this community is expressed in Fig. 3, which shows a network-integrated system of simulators, instruments, and data processors. Notice that the various components in Fig. 3 must be linked by a wireless communications network.

In order for a simulation to effectively use dynamically-evolving external data resources, including other simulations, several technical advances are required. Among them are the following.

- *Agent-Based Systems*: Developing technologies for indexing, searching, and efficiently retrieving data from the cloud. Methods for indexing and searching text-based information on the World Wide Web are still relatively immature but at least minimally effective. By contrast, methods for indexing, searching, and retrieving scientific data are relatively unexplored in spite of the fact that the Internet was initially designed to foster high energy physics research. While software agents do not solve the problem of indexing, they provide a powerful platform for implementing search and data retrieval functionalities.
- *Distributed Discrete-Event Simulation*: Developing the concept that continuous time and space simulations (that is, based on discretizations of partial differential equations governing a continuously evolving continuum, for example) must be treated as distributed discrete-event simulations. The discrete-event nature arises from the fact that external data request and assimilation steps are discrete events that might require rollbacks and other types of synchronization procedures.
- *Uncertainty in Distributed Discrete-Event Simulations*: Studying the mathematical properties and representations of uncertainty evolution in simulations. As a simulation progresses forward in time from a known initial state, the error in the state variables of the simulation will change and evolve. The conceptual principles behind these evolving distributions of random variables are straightforward and may be described through standard stochastic systems and probability theory. However, the specific computational steps required to make

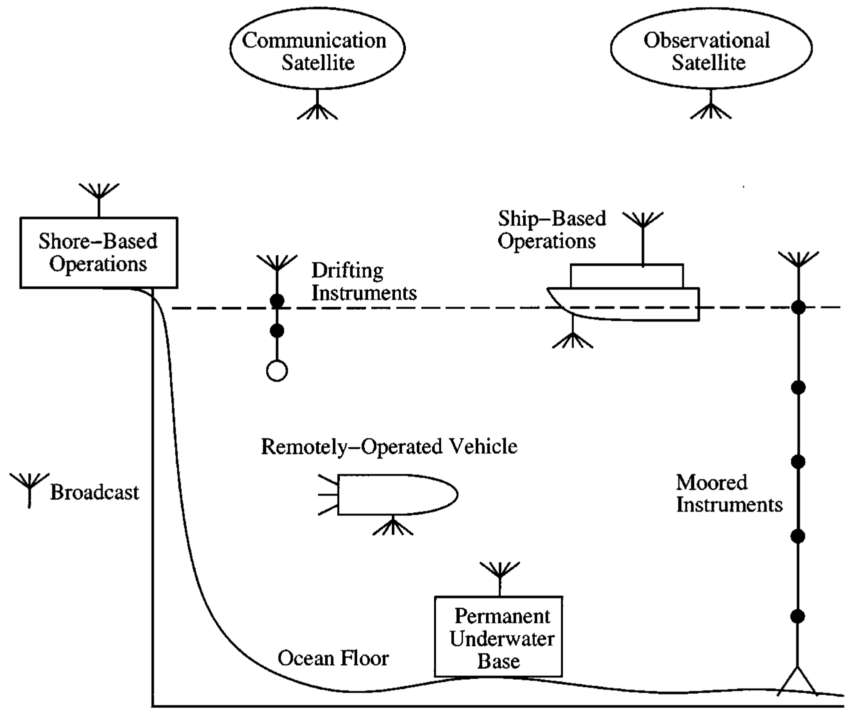

Fig. 3. This figure demonstrates the future of ocean modeling in which data will be obtained from a variety of monitoring instruments and integrated seamlessly into simulations running concurrently.

this feasible are largely unexplored in simulations governed by discretized partial differential equations.

This paper discusses the first area and presents our initial agent-based framework, called ABELS (Agent-Based Environment for Linking Simulations), for linking distributed simulations and data resources. The other two areas will be discussed in future papers.

B. Software Agents

A software agent is an autonomous computer program that operates on behalf of someone or something. A mobile agent has the ability to migrate under its own control within a heterogeneous network, while a stationary agent executes only on the system on which it began execution. Most research on software agent systems focuses on mobile agents because they require additional capabilities.

It is well known that mobile agents have several advantages in distributed information-retrieval applications [2], [3], [26], [29], [30], [35], [38]. By migrating to an information resource, a mobile agent eliminates the intermediate data transfer and can access the resource efficiently. This is particularly useful when large data transfers over a low-bandwidth network would be infeasible or undesirable. By migrating to the other side of an unreliable network link, an agent can continue executing even if the network link goes down. Since network computing is inherently heterogeneous, mobile agents are designed to be system-independent. A mobile agent can also adapt to changing conditions. For example, it can choose migration strategies depending on its task and current network conditions, and then change its strategies as network conditions changes. Finally, the mobile-agent model is naturally suited for distributed applications since an agent can migrate through a set of machines, send out child agents to visit machines in parallel, or remain stationary and interact with resources remotely.

These advantages are not unique to mobile agents. In fact, a specific application can typically be implemented just as efficiently with traditional techniques such as message passing, process migration, or remote procedure calls. However, different applications require different techniques for optimal performance. Mobile agents are useful not because they make distributed information-retrieval applications possible, but because they allow a wide range of applications to be implemented efficiently, robustly, and easily within a single, general framework.

Mobile agents are particularly well suited for the scenario present in our simulation and data cloud. In [26], Johansen noted that mobile agents provide good performance for data-intensive applications that are remotely located and have specialized needs. These are the characteristics of many cloud participants. Agents are also useful in detecting when a resource changes. For example, an agent may discover that a resource is unavailable or does not contain the desired information. Depending on the simulation, the agent might report the failure, move to an alternative resource, or wait for the desired resource to become available. Code mobility is valuable in wireless networks that experience frequent disconnects and variations in link transmission properties. While it is unlikely that an entire simulation would migrate, the fact that the agent migrates to the remote location allows the processing of data to continue even when a link is broken. That is, even though the agent may be unable to send data back to the simulation due to a broken link, the agent can continue to process the data locally, anticipating that the link will eventually return. Finally, mobile agents allow flexible remote data operations. That is, they do not require preinstalling specialized remote procedures for remote procedure invocation, although a general-purpose agent server must be preinstalled on all computing nodes that will support mobile agents.

In conjunction with the data and simulation cloud, mobile agents can be used to:

- report to a server that a new data resource is ready for use;
- obtain new data products from instruments and deliver subsets of the full dataset to simulations or other consumers;
- implement standing remote requests and report on the status of a simulation (e.g., plot the current estimate of velocity and its estimated error);
- ask sensors or probes for observational support of a certain kind due to unacceptable error growth in a forecast;
- efficiently get and prepare a data product for a specific assimilation purpose.

To summarize the generic advantages of using mobile agents for linking distributed simulations and data resources, we list the key points made above.

1) *Efficiency*: Mobile agents can move to a large data resource as an alternative to moving large datasets to a client. The mobile agent performs a remote computation on the data and sends back only the relevant data products.
2) *Flexibility*: Mobile agents allow flexible remote data operations. They do not require preinstalling specialized remote procedures for remote procedure invocation although a general-purpose agent server must be preinstalled on all computing nodes that will support mobile agents.
3) *Adaptability for Disconnected Operations*: Mobile agents are persistent in space and time. They can migrate within a network, docking on remote nodes should network conditions be unstable. This functionality is most valuable in volatile environments such as wireless networks.

Our framework for linking distributed simulations is implemented using Dartmouth's D'Agents system, which began as the Agent Tcl system in 1995 [17], [21]. It initially supported mobility and communications functionality for Tcl-based programs. Since that time, support for other languages such as Java, Scheme, and Python has been added within the same general architecture. Additional modules for debugging, visual programming, and security among others have been added as well. With the introduction of multiple language support, the system is now called D'Agents (pronounced "dee-agents"). More complete descriptions and a public D'Agents distribution can be found at http://agent.cs.dartmouth.edu and http://actcomm.dartmouth.edu.

Related research on mobile agent systems is conducted at several locations, including Mitsubishi Electric [31], IBM [25], and the University of Tromso (Norway) [26]. A comprehensive web site dedicated to mobile agent systems research and development is http://www.cetus-links.org/oo_mobile_agents.html.

C. Related Work on Simulation

Most large-scale distributed simulation systems focus on military applications such as battlefield simulations. For example, the Aggregate Level Simulation Protocol (ALSP) [1] is used by the United States military to connect analytic and training simulations. Similarly, the Department of Defense's high-level architecture (HLA) [4], [8], [24] is designed to support interoperability and synchronize the execution of multiple simulations. In both systems, simulations are designed using strict rules and connected into federations. Furthermore, significant rewriting of a simulation is often needed to meet those specifications.

Other systems such as GLOBUS [12]–[14], [16] and the Common Object Request Broker Architecture (CORBA) [4], [5], [9] facilitate the creation and distribution of a large distributed program among multiple heterogeneous nodes. While these systems are useful for general computing, they do not have built-in mechanisms to handle the special needs (e.g., checkpointing and rollbacks) of distributed simulations.

III. Using Agents to Communicate with Data Sources

A. Discussion

Using an agent-based system for communication within the simulation cloud allows the responsibility of establishing and maintaining links for data transfer to be removed from the simulation and placed with the agent-based system. A stationary agent is colocated with each simulation in the cloud. Once a simulation has established a connection with its agent, it is the agent's duty to fulfill requests made by the simulation. The simulation may make a request for data corresponding to a particular entity. This simulation is not directly concerned with the source of the data, but it is concerned that the data is the correct quantity and is as accurate as possible. Different sources that provide the same quantity may have varying degrees of accuracy, but it should not be the responsibility of the simulation to decide between these. The simulation provides constraints that detail exactly what data is needed and what format it should be in, how accurate that data must be, the priority at which it must be found, etc. The agent then attempts to discover a source for this data and provide it to the simulation.

One of the features of the simulation cloud is to allow simulations and other objects to enter and leave the cloud dynamically. For example, some members of the cloud may be linked via wireless networks. In order to support this, the agent system must be able to support operations even when disconnections occur. The data source needed to fulfill a request may not always be available. The simplest solution is to inform the simulation that the data is not available and let the simulation handle the problem. Since the responsibility for establishing links between data producers and data consumers has been moved from the simulations to the agent system, this is not the best approach. If there are other sources for the same data, perhaps not as accurate or desirable as the primary source, the agent can retrieve data from one of these secondary sources and provide it to the simulation. Without any action on the part of the simulation, the system is allowed to progress forward despite a loss of communication with the primary data source. It is important to verify that the secondary sources still fall within the original constraints of the data request, or alternatively to inform the simulation that the request cannot be fulfilled in its current form.

When the agent first begins to search for the needed data, it may discover numerous potential sources. For example, assume the simulation needs to know the water temperature at a particular location in the ocean. The agent may find a temperature sensor at that location, a computational model of that section of the ocean, or even a tool that can extrapolate data from past records. In this situation, the sensor is the most accurate and thereby the most desirable data source. However, it is possible that the communications link to the sensor is unreliable. The agent should go directly to the sensor for the data whenever possible, but move to the next most reliable data source as necessary.

B. An Ocean/Shipping Simulation Example

A simple prototype utilizing two simulations was constructed to demonstrate the concepts of simulation interaction through the use of agents. The first simulation is a very simple model involving ports along the Gulf of Maine and vessels that travel between them. This simulation was implemented using the SPEEDES framework for parallel discrete-event simulation [37]. The second simulation is an oceanographic simulation of the body of water on which these ports lie. This circulation model for the Gulf of Maine is a continuous ocean circulation simulation model based on the numerical solution of partial differential equations. As shown in Fig. 4, when the port/vessel simulation needs information such as the surface currents in a given area, it uses agents to obtain this data from the oceanographic simulation.

The agent software was built using Agent Java, which is part of the D'Agents system. The use of agents has two purposes, the first being that it allows simulations running on different platforms to be linked together. Second, it improves the transfer of data across the network. In this demonstration, the oceanographic simulation provides vast amounts of data (e.g., current, temperature, salinity) about a three-dimensional (3-D) body of water, whereas only the surface currents are needed by the shipping simulation. Agents, with their ability to travel to the data source and extract only the necessary information on a case-by-case basis, can reduce the network traffic significantly.

This prototype problem demonstrates three points.

1) It shows that a simulation can be designed to obtain needed data from sources such as other simulations.
2) The use of agents will both optimize the network traffic and allow the system to be platform independent.

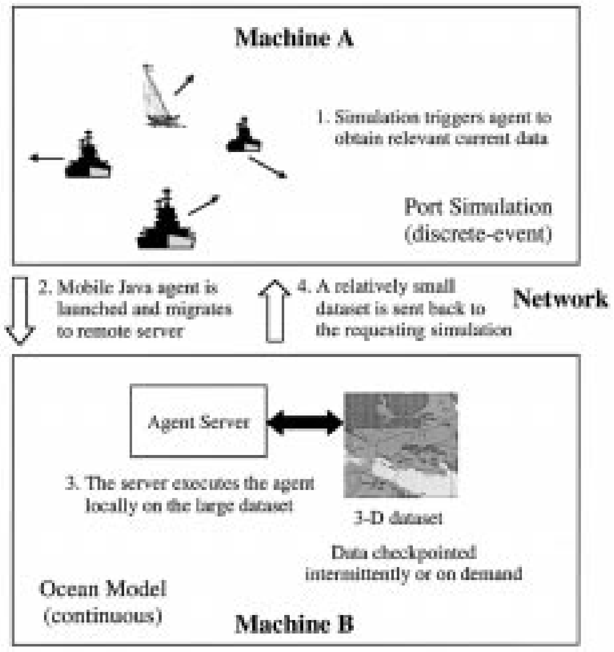
Machine A
1. Simulation triggers agent to obtain relevant current data
Port Simulation (discrete-event)
2. Mobile Java agent is launched and migrates to remote server
4. A relatively small dataset is sent back to the requesting simulation
Network
Agent Server
3. The server executes the agent locally on the large dataset
3-D dataset
Data checkpointed intermittently or on demand
Ocean Model (continuous)
Machine B

Fig. 4. A graphical depiction of the agent's role in the prototype example described in this section.

3) This system demonstrates the ability to link together two very different types of simulations (both discrete-event and continuous).

It is important to note that neither the Gulf of Maine circulation model nor the shipping simulation was developed with any knowledge of the operation or data format of the other. The ability to make two or more simulations interactive when they were not designed to work together is an important demonstration in this case study. While it is necessary for the agent system to understand the formats and interfaces of each simulation, the simulations do not require knowledge of each other. For purposes of this demonstration, this knowledge was hard-coded into the agent system. In Section IV, methods for generalizing this task are described.

As a ship in the simulation begins to travel from one port to another, the simulation needs external data to be able to calculate the transit time between these two ports. Specifically, the transit time between two ports is affected by how much of the ship's speed is used to maintain its position against the current. When the ship requests this information, the simulation makes a call to a mobile agent that travels across the network to the location of the data source. Once there, the agent obtains the needed data and returns across the network to the location of the simulation. Finally, the agent informs the simulation that it has completed its task and makes the data available to the simulation.

In this demonstration system, the data source (Gulf of Maine simulation) has vast amounts of information that it records into data files. Only a tiny portion of the information in these files is needed. Since data transfer across a network is very time consuming, the efficiency of the system can be improved greatly if only the necessary information is transferred across the network. This has been accomplished through the use of mobile agents. In situations such as this, where a small amount of needed data is located inside a large dataset, mobile agents provide an excellent method for minimizing the amount of data transferred across the network. In this system, the data files contain all of the surface current data for the entire Gulf of Maine, but only the currents along the ship's path are needed. Given information about the ship's path, the agent can obtain only the necessary data for that ship.

This demonstration has been expanded to show other concepts of the interaction framework. The next section describes the framework architecture, and Section V describes the current implementation.

IV. ABELS Framework Architecture

A. Overview

We have developed an agent-based software framework to facilitate the dynamic exchange of data between distributed simulations and other data resources. The framework is called ABELS. The goals of this framework are to allow existing simulations to join the cloud with a minimal amount of code modification and to provide a system where simulations can interact without any *a priori* knowledge of each other's interfaces. The users of the simulations and other data resources are able to define custom interface agents to their particular systems, and then use these interface agents to communicate and interact with other participants.

The framework not only provides the functionality needed to connect multiple simulations together but also allows for the translation of information. This is necessary to allow seamless integration of multiple, independently developed software systems. The approach that we are pursuing is to describe the meaning of the data rather than the format of the data.

A standard that describes only the structure and format of the data is not sufficient for strong interaction between participants. The meaning of the data must also be described. Also, it is necessary to be able to describe relationships between formats. For example, assume that there is an oceanographic simulation that has *depth* as one of its variables. To properly describe this variable, it is necessary to know that it represents the depth in meters, measured from the surface with a positive-down Z-axis. An atmospheric simulation may have *altitude* as one of its variables. To properly describe this variable, it is necessary to know that it represents the altitude in feet, measured from the surface with a positive-up Z-axis. If there is to be any interaction between these variables, then it is also necessary to know the relationships between feet and meters and between a positive-up and positive-down axis. Once these relationships are defined, data described in one method may be translated to the other.

While this may seem like much more effort than simply defining standard data formats, the end result is a very powerful data description and interaction system. As described by Staniford-Chen, *et al.* [36], such a description language must have the following attributes.

1) *Precision*: Multiple readers of the same description must not draw contradictory conclusions from it.
2) *Layering*: Specific concepts must be able to be described in terms of general ones.
3) *Self-Defining*: It should be self-evident from the description how each datum should be interpreted.
4) *Extensible*: There should be a mechanism through which a user or group of users may build a personal vocabulary and indicate this to other users. Additionally, it must be possible to describe relationships between the various vocabularies.

It is not our intent to build such a system. There are technologies, in various stages of development, described below that we intend to leverage to solve these issues. Our area of interest lies within the framework and operation of the simulation communication system.

The architecture of our system consists of four basic components: user objects (e.g., simulation entities), generic local agents, mobile helper agents, and a broker agent. The user objects are often described as simulations but can actually be any producers or consumers of data. For example, a sensor that generates data used as input to a simulation would be a data producer. Visualization tools that are used to collect and display output of various simulations would be consumers. Simulations, of course, can be both producers and consumers of data. The generic local agent (GLA) is a user object's interface to the simulation cloud, and all communications and commands are routed through the object's GLA. Thus, the user object needs to communicate directly only with its GLA. Within the simulation cloud, the broker establishes the necessary links between the user objects by connecting their GLAs. Finally, the helper agents (HAs) are mobile agents that are used to minimize the network traffic by performing computations at the data source. The various components in this system are shown in Fig. 5 and described in detail in the following subsections.

B. User Objects and GLAs

In order to participate in the simulation cloud, a user object (such as a simulation) needs some basic functionality. At a minimum, the user object must be able to connect to and disconnect from the simulation cloud. Data producers will also need to advertise their services to the outside world. Data consumers will need to be able to discover advertised services and invoke those services. Other functionality includes the ability to generate and use helper agents and respond to changes in the availability of other user objects in the simulation cloud.

The basic functionality is provided to the user object through its GLA. The GLA is the user object's sole connection to the simulation cloud, and there is a one-to-one relationship between a user object and a GLA. The user object, through commands sent to and from the GLA, has the ability to connect to the cloud, disconnect from it, advertise its services, and look up available services. Essentially, the GLA is the front end of the user object. However, the user object must be able to communicate with its GLA. This communication capability may be built into the simulation or provided via a separate interface. Our prototype system described in Section V discusses the use of such an interface.

The architecture of the GLA can be described with three main components: the agent communication system, the command interpreter, and the dynamic command table. (These components are shown in Fig. 6, which is discussed in Section VI.) The communication system controls all communication between the GLA and other objects in the simulation cloud, including the local simulation, other GLAs, and the broker. We use sockets for communication since they provide a platform- and language-independent method for communication.

Any incoming messages are passed to the command interpreter, which processes the messages and carries out their instructions as appropriate. Messages fall into two general categories: broker interaction commands and execution commands. Broker commands cause the GLA to interact with the broker, either to advertise or discover services. Execution commands cause the GLA to execute functions from the local simulation or invoke commands on a remote GLA. Broker interaction functions are built into the GLA, are static, and are identical across all GLAs. Execution commands, on the other hand, are dynamic in nature.

The abilities available to a GLA are determined by what services it has advertised to the simulation cloud or discovered from the broker. That is, the GLA knows what services or functions are provided by the local user object (simulation) and what remote services have been requested by the local simulation.

The dynamic command table provides a data structure for maintaining the necessary information regarding these abilities. This information includes the name of the service, a detailed description of the service, a description of the service's input and output parameters, and the location and execution method of the service. When a service is either advertised or discovered, its specifics are written into the GLA's dynamic command table. Once the service is listed in the GLA's table, the GLA can access that service without going through the broker. Fig. 5 shows direct GLA-to-GLA connections between simulations S1 and S6, S2 and S4, and S4 and S8.

For an advertised service (one provided by the local user object), the execution request will come from a remote GLA that needs information from the local service. For a discovered service (provided by a remote user object), the execution request will originate from the local simulation.

Finally, the GLA provides the ability for a user object to disconnect itself from the simulation cloud. The GLA of the disconnecting object reports this to the broker (who reports the disconnection to other remote GLAs) and then terminates itself. Note that the simulation may continue to run even though it no longer participates in the cloud.

In our current system, the GLAs support only blocking requests from a simulation. When a simulation passes an execution command to its GLA, the simulation must wait until this command has been completed before continuing. Alternatively, it must provide its own mechanism for generating the request in a separate thread and polling for the completion of the request. In future implementations, nonblocking requests will be allowed, and the GLA will be able to interrupt or otherwise inform the simulation when a request is completed. In certain situations, the GLA will also be able to prefetch data before it is needed by the simulation.

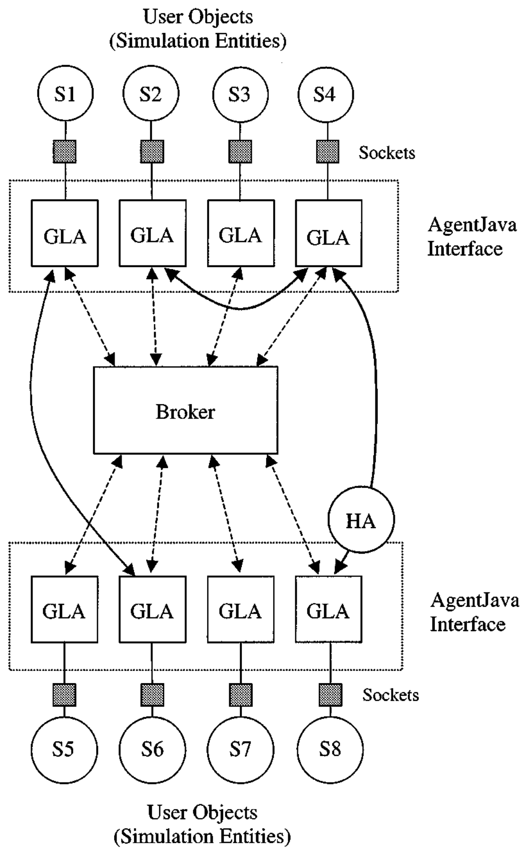

Fig. 5. Basic framework connecting the components in the cloud.

The GLA is the portal through which objects in the simulation cloud communicate with each other. It provides a standard interface through which objects may interact and it provides all necessary functionality to be part of the cloud. The user objects in the simulation cloud should not need to directly interface or communicate with any object other than the GLA. From the simulation's point of view, the GLA is what provides all functionality and responds to all requests. Likewise, a data producer will see all requests for data coming through its GLA. That is, the GLA is used to make the details of the framework invisible to the end user and thereby simplify the task of connecting to the framework. Each object in the cloud needs only a single GLA. One producer can satisfy the requests of many consumers through its GLA. Likewise, a single consumer can use its GLA to obtain information from many remote producers scattered through the cloud. The identities and locations of the other objects are hidden from the user, and thus, so is the need to maintain and update that information.

C. The Broker

The broker acts as a database for cloud participants and their advertised services. When a user object advertises a service to its GLA, the GLA reports this advertisement to the broker. Later, when a remote user object searches for a particular service, its GLA will query the broker. Upon finding a match for the query, the broker will inform the remote GLA of the name, location, parameter list, and execution method of the service. This information will be transferred to the remote GLA, who will write it into its dynamic command table. Once the remote GLA has this entry, it can directly access the local GLA of the service without any further communication through the broker.

In the case where a desired service is not found, it will be possible for a user object to tell its GLA to leave a standing request at the broker. If at some future time the service does become available, the broker will inform the appropriate GLA of this, and the GLA will in turn inform the simulation.

A similar approach is used when an incomplete service is found. If the broker later finds a better match, it will inform the GLA of this change in the state of the simulation cloud, and the GLA can begin using the new service.

If multiple matches are found for a query, the broker will inform the GLA of all possible matches so the GLA can determine which source is most appropriate. Furthermore, the GLA will be able to use a second source if the connection to the primary source is lost.

The interaction between the GLA and the broker is invisible to the end user or simulation object. Thus, in a dynamic system, changes should be handled without involving the user. If a data source stops responding, for example, the GLA is able to inform the broker of this, and the GLA or the broker will attempt to replace that data source with another. This may be done without ever informing the user (consumer) of the problem. Since the consumer does not request the data source by identity but rather by functionality, any data source with equivalent functionality will be able to satisfy the request. Of course, the consumer will be notified in the event of a nonrecoverable failure (e.g., no suitable replacement is available).

A central 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 meaning of the data being transferred, rather than define a common format for the data.

In describing a framework for communications between intrusion detection systems, Kahn *et al.* [27] state three conditions that must be met for strong interactivity between independently developed systems.

1) Configuration interoperability, which refers to the ability of two systems to discover one another and communicate data back and forth.
2) The ability to parse the data being transferred (e.g., agree on data types, byte ordering, etc.).
3) Intercomprehension, or agreement on the meaning and definition of the data descriptors.

When all of these conditions are met, it is possible for two independently developed systems to interact closely even though they were designed independently.

The GLA interface between data producers and data consumers provides the functionality to meet the first of these requirements. The GLA provides the ability to issue commands to remote systems and, through communication with the broker, to discover the existence and location of these systems. The second requirement may be met with a minimal amount of description of the data being transferred. The third requirement, however, is significantly more challenging. In order to solve this, we are currently looking at a number of existing research efforts.

Knowledge Interchange Format (KIF) [28], a development of the Logic Group at Stanford as part of the ARPA Knowledge Sharing Effort, is an attempt to develop this. KIF is based on first order logic with extensions to support nonmonotonic reason and definitions [11]. As described in the draft proposal [28], KIF is designed for the interchange of knowledge in disparate computer systems. It is not designed as a human interaction language, nor is it designed to be the internal representation of data within a system. Instead, KIF is designed to facilitate the independent development of software that will eventually communicate by providing a mechanism through which the meaning of data may be described. Additionally, the relationships between various meanings and definitions may also be structured and described. The usage of this system provides the ability to meet the third requirement for interactivity, intercomprehension.

Another product of the ARPA Knowledge Sharing Effort is KQML [11], [40]. This is described as a "language that is designed to support interactions among intelligent software agents" [28]. KQML is concerned with knowing whom to talk to and how to maintain a conversation. This includes the ability to initiate a conversation. It is designed to control the structure on the conversation while tools such as KIF define the language of the conversation. It enables programs to identify, connect to, and exchange information with other programs. This allows for a highly interactive communication system between two or more independently designed systems. The role filled by KQML falls under the first requirement for interactivity, configuration interoperability. While the GLA and broker system meet this requirement to a degree, the inclusion of KQML could greatly expand the performance and abilities of the communication system.

The Foundation for Intelligent Physical Agents (FIPA) [15], [39] is promoting a set of standards to promote interoperability within and across agent systems. These standards define agent to agent, agent to human, and agent to external system interactions. Included in the specifications set forth by FIPA is an ontology server interaction standard. This may prove to be quite useful in developing the data description system used within the framework. By adhering to FIPA standards, some of the difficulties in meeting the first two requirements, configuration interoperability and data parsing, may be alleviated. Generally, the areas of the framework governed by these two requirements are internal and ideally will not be seen by the end user. The user will be required to describe the meaning and structure of the data, but not the communication and interaction methods. Thus, the use of FIPA standards will apply more to the internal interactions of the framework as opposed to the interface to user objects. Additionally, the use of FIPA compliant tools and services may help address issues such as security and interoperability with other systems.

In order to be scalable and robust, the broker should be designed as a distributed system rather than as a large central database. For example, a group of user objects located at one facility could have their own broker. This broker would know the locations of other brokers, and upon receiving a request that it could not fulfill, it would simply pass this request to another broker. This system is similar to domain name server (DNS) lookup, in which a higher level DNS is queried only when a query to the local DNS fails.

D. Helper Agents

The final component of the simulation cloud is the helper agent, which is used to extract or generate the desired data while minimizing network traffic. Since a producer does not necessarily know the exact needs of its consumers, it is unrealistic to expect the producer to provide all possible forms of its data. Thus, data may need to be collected from the producer and then manipulated to fulfill the needs of the local consumer.

For example, suppose one simulation needs the average value of a set of data provided by a remote producer. One possibility is to send all of the data from the remote source to the simulation and then compute the average. This is inefficient in terms of network usage since it requires a large amount of data to be transferred when only a small amount (i.e., the average) is required. A better approach would be to perform the averaging calculation at the data source. In order to do this, we introduce the helper agent.

In our software architecture, a helper agent is a mobile agent capable of traveling to a remote location and executing there. Through its GLA, a simulation may create a helper agent that travels to the remote GLA in order to preprocess the data before transferring it over the network. From the remote GLA's viewpoint, the fact that the helper agent has moved itself across the network is transparent. The helper agent and the GLA communicate over a socket just as if they were still on remote machines. In fact, the helper agent issues the same commands to the remote GLA that would have been sent from the GLA that launched the helper agent. Once the helper agent collects the results, it processes them and then returns the final result across the network. Fig. 5 shows a helper agent processing and transferring data between the GLAs for simulations S4 and S8.

It is possible that given a specific request, a broker may find multiple producers capable of satisfying the requirements. When this occurs, helper agents may be sent to each viable producer to dynamically decide upon the best source of data. This evaluation may be based on the speed of the response, transfer rates between the data sources and the GLA, or even the accuracy of the data being provided. If the producer's advertisement of its services did not provide detailed enough accuracy information, or its advertised accuracy was not trusted, the helper agents could perform a functional validation of the data source. Of course, this is highly dependent on the functionality being provided and the consumer's knowledge of that data. The main advantage of using helper agents to perform this task are that the agents are able to localize the testing at the source, thus conserving bandwidth. When used in this fashion, the local GLA communicates with the helper agent as if it were another GLA. Requests that would normally be sent to a remote GLA are sent to the helper agent, and the helper agent is then responsible for handling these requests, possibly forwarding them to appropriate remote GLAs. This makes the use of the helper agents transparent to the GLAs involved.

In some cases it is desirable to have the evaluation of the data sources be a continuing process rather than a one-time analysis. In cases similar to the example described in Section III, the quality of the data sources may vary with time. The helper agents can be designed to constantly evaluate the quality of their sources and determine on a continuous basis which source is most desirable.

The two primary goals of this architecture are to allow existing simulations to join the cloud with a minimal amount of code modification and to provide a system where user objects can interact without any *a priori* knowledge of each other's interfaces. By using a standard, platform and language-independent protocol, sockets, we are able to provide an interface to which it is relatively easy to connect. Furthermore, by decoupling the user objects from one another, there is a great deal of flexibility in the system. A user object is not required to know anything about the other user objects; it needs only to talk with its local GLA, and all further communication is handled through the agent system.

V. Current Implementation

We have developed an initial implementation to demonstrate the concept of using an agent-based system to exchange data dynamically between simulations. This section describes the basics of our implementation, while the next section discusses a search-and-rescue prototype that demonstrates the operation of our system.

We are using Agent Java from the D'Agents system [3], [7], [17]–[22], [29]. Our current implementation provides basic functionality for the generic local agents (GLAs). Implementation of the broker system is part of our future work.

As discussed earlier, every simulation has its own GLA. Each GLA is developed as a Java application providing end-to-end connectivity with certain built-in functionality, and the GLAs communicate with each other through sockets. Note that sockets shield the programmer from the low-level details of the network, like media types, packet sizes, packet retransmission, network addresses, and many other lower level implementation details. Most importantly, sockets allow platform-independent communication.

At a minimum, each GLA provides five standard functions that are invoked through commands sent from a simulation to its GLA. The commands are as follows.

- *Lookup*: With this command, the GLA takes the function name and function description (which can possibly include expected input and output parameters) from the simulation and sends this information to the broker. The broker searches the simulation cloud to find a match and then returns the contact information for the entity (simulation, sensor, etc.) that provides the requested service.

- *Advertise*: This command is used by an entity in the simulation cloud to advertise its producer capabilities. When an entity invokes this command, it supplies the name of the function or service it can execute along with a description of the service.
- *Execute*: A consumer in the simulation cloud invokes the execute command to initiate a request for data. The consumer provides the local function name and description to its GLA. If a match is found in the GLA's table, the request will be processed by the appropriate producer and the results will be channeled back to the consumer via the respective GLAs. If a match is not found, the GLA contacts the broker to find a match. If no match is found by the broker, the GLA tells the consumer that it will not be able to satisfy the request.
- *Table*: The table command is used to access the contents of the dynamic table data structure maintained by each GLA. This table is an integral part of the system because it is involved in the execution of each command. When a user object or entity first advertises its capabilities to the simulation cloud, the capabilities are recorded in the table by its GLA. The table entry includes the function name and location, where the location is "local" for an advertisement. When a lookup command is invoked by an entity, the broker checks the tables of the other GLAs in the cloud, finds a match, and sends the appropriate information to the requesting GLA. In this case, the table entry includes the local function name, the location of the service, and the remote function name of the service. Upon encountering the execute command, the GLA checks the table to see if a match already exists; if it does then the request is channeled to the location specified in the table. Otherwise, the broker tries to find a match.
- *Disconnect*: This command is used by an entity to exit from the simulation cloud. The GLA informs the broker of the disconnection and then terminates itself. Note that the simulation may continue to execute after it leaves the simulation cloud.

The agent system has two communication interfaces: the internal interface between each GLA and the broker, and the external interface between the GLA and its simulation or simulation interface. Java sockets are used to implement both the internal and external communication interfaces. With different simulations entering the cloud from different platforms and using various programming languages, the platform-independent Java socket provides the communication flexibility for interoperability.

Communication error control is handled through the use of acknowledgments. When the simulation sends the first command to the GLA, the GLA validates the command and sends back a "valid" acknowledgment. Once the simulation receives an acknowledgment, it sends other information such as the function name and function description. All of the commands and descriptions are acknowledged individually.

VI. Search-and-Rescue Example

A. Description

Suppose that a Coast Guard search-and-rescue unit receives calls for help on an unpredictable (random) basis. Each call indicates the estimated location and time of an accident. The Coast Guard needs to obtain a forecast of the trajectory of the survivors. The information must be sufficient to dispatch a vessel from one or more locations and deploy a search pattern in and around the predicted trajectory. Furthermore, the temperature history along that trajectory is needed to determine the likelihood of survival. Since the probability of survival depends on the number of accumulated degree–hours, survivors must be intercepted before they accumulate too much exposure to cold.

Coincidentally, Dartmouth's Numerical Methods Laboratory (NML) provides an ocean forecasting service that archives the latest forecasts for ocean velocity and temperature. Given a suitable request in terms of estimated location and time, the service can compute a trajectory (i.e., location and temperature versus time). This is exactly the service needed by the Coast Guard unit.

To demonstrate the basic idea of using a software agent-based system to exchange data between remote simulations, we developed a simple prototype for this search-and-rescue scenario. The Coast Guard simulation can be visualized as a client or consumer that is making requests, and the ocean forecasting service is the server or producer that is servicing these requests dynamically. The communication is facilitated by our agent-based system and is completely transparent to both simulations.

B. Implementation

For this prototype, we developed GLAs with basic capabilities and basic communication interfaces for the Coast Guard and ocean simulations. However, the broker that is responsible for linking the simulations has not yet been developed; thus, the broker's functionality is hard-coded in the prototype.

Our prototype consists of four basic components: the Coast Guard simulation, the ocean forecasting simulation, the NML server interface for the ocean simulation, and the individual GLAs. The components are shown in Fig. 6 and described in the following paragraphs.

The Coast Guard implementation in this prototype is a Java application that enables the user to enter the commands to the GLA as if they were sent from a simulation. This application is similar to the interface that a simulation needs to implement to be able to join the simulation cloud and talk to its GLA. The Coast Guard simulation and its GLA are shown in the left side of Fig. 6.

The ocean forecasting simulation was developed by NML without any knowledge of our simulation cloud or GLAs. By itself, the simulation knows only to take an input file containing the required parameters and generate the corresponding trajectory. Thus, we created a simple interface called the *NML server* to handle communication between the ocean simulation and its GLA. The NML

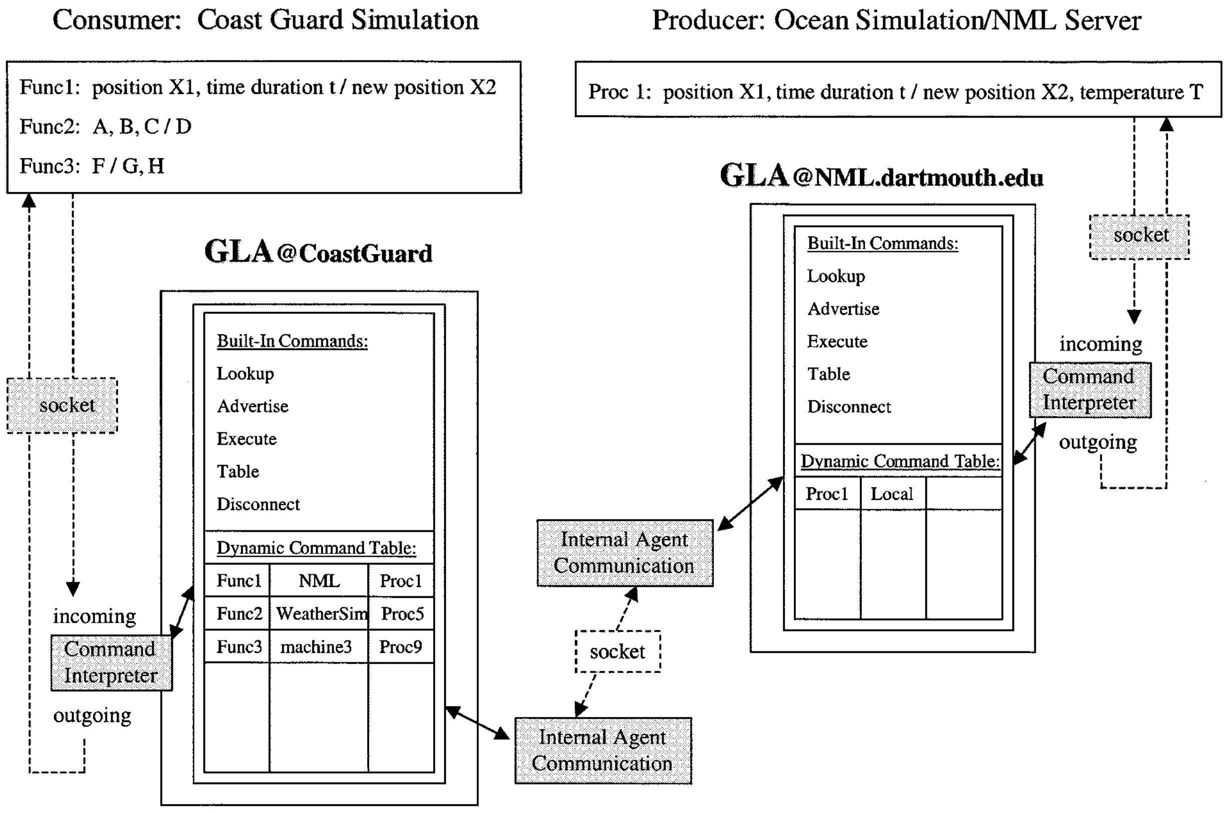

Fig. 6. Search-and-rescue prototype implementation.

server interface receives the request from the GLA, creates the formatted input file needed by the ocean simulation, and sends the file to the simulation to execute. The interface then waits for the result and sends it back to the GLA. Since the NML server is necessary for the ocean simulation's participation in the simulation cloud, we consider the "user object" to be the ocean simulation and its NML server interface. The right side of Fig. 6 shows the ocean simulation/NML server combination and the corresponding GLA.

In our prototype, we can visualize the GLA acting as a client or a server, depending on the command issued, using sockets to listen to port 1610. The GLA listens to this port for incoming requests from consumers and also for processed results from producer entities. A single listening socket is used to handle requests from the local simulation object and from other objects internal to our system (e.g., other GLAs, brokers, etc.). The socket can therefore be viewed as a pipe transporting streams of data between various GLAs and the entities. The NML server listens to port 2812 for incoming requests from its GLA, communicates with the simulation, and sends the required result set back to the GLA, who then sends it to the remote GLA that needs the data. The port numbers are chosen arbitrarily.

During an advertise operation, the GLA takes the function name and description of the service the simulation would like to provide and stores it in the dynamic function table with a "local" service location. In Fig. 6, the ocean simulation advertises that its service named Proc1 takes parameters $X1$ and t and returns outputs $X2$ and T. Proc1 could be the ocean simulation service that takes parameters such as the current position $X1$ of an object and the time duration t after which the result is required. Proc1 will then return the new position $X2$ to which the object has drifted after time t and the water temperature T at the current position of the object. As mentioned earlier, an ontology-based system could be used to describe these parameters so that the broker could perform the appropriate match. For example, we could describe the result temperature T as a water temperature in Celsius of a location in the Gulf of Maine calculated from a nature (ocean) simulation. In future implementations containing a functional broker, the advertise function will be responsible for sending this information to the broker to store in the broker's database for further lookups.

During a typical lookup operation, the consumer gives its GLA the description and local function name (to be used in the GLA's table), and the GLA calls the broker's lookup method to find a match. In this prototype, the broker's lookup method returns hard-coded information on the producer's location (i.e., we have simulated the broker's actions). Given the producer's information, the GLA enters the local function name, remote service location, and remote function name into its dynamic function table. In Fig. 6, the Coast Guard simulation requests information for three services that it labels Func1, Func2, and Func3. Matches for these services were apparently found since all three are listed in the GLA table. Note that the Coast Guard GLA's table shows the local name for each service along with the remote location and remote name for each service. Thus, the first entry confirms that Func1 is executed by calling Proc1 on the NML machine.

The entries for Func2 and Func3 are for demonstration purposes only; they were not used in the prototype.

The GLA has the ability to filter results and choose only those requested by the client. In Fig. 6, the Coast Guard simulation requests a service that gives the result $X2$ for the new position. The ocean simulation computes both the new position $X2$ and the temperature T. When it receives the results, the Coast Guard GLA discards the temperature T and send only the new position $X2$ to the Coast Guard simulation. If bandwidth is a concern, the ocean simulation GLA can be informed that it should do the filtering. In either case, the GLA table will have the information concerning which results are needed.

When an execute operation occurs, the GLA takes the function name and description, looks it up in the dynamic function table, and continues only if the function exists in the table. If the function is a local service, the GLA will open a connection with its local simulation or interface, send the information, wait for the result, and send the result back to the enquirer. In Fig. 6, a request to the ocean simulation GLA to execute Proc1 will result in execution of the local Proc1 service. If the function is a remote service, the GLA will open a connection with the appropriate remote GLA object and ask that GLA to execute this function. In Fig. 6, a request to the Coast Guard GLA for Func1 will cause a request to the ocean simulation GLA for Proc1.

A sample run of our prototype can be visualized as follows.

1) The Coast Guard simulation and the ocean simulation/NML server enter the simulation cloud. The GLAs are activated and wait for incoming requests.
2) The NML server performs an advertise operation to inform its GLA of its capabilities. Information on Proc1 is stored in the GLA's table.
3) The Coast Guard simulation performs a lookup operation for the function that it will need to execute when a rescue call occurs. The broker returns the name and location of the required procedure that matches the criteria, and this information (i.e., the NML location, etc.) is entered into the Coast Guard GLA's table. Note that Func1 is the Coast Guard's name for the service named Proc1 located at the NML machine.
4) The Coast Guard simulation then issues an execute operation whenever it needs information from the ocean forecasting simulation. The GLA verifies that the required functionality is present, and the Coast Guard simulation gives its GLA the necessary parameters to be passed to the ocean simulation.
5) The Coast Guard's GLA then sends the data provided by the Coast Guard simulation to the ocean simulation's GLA.
6) The ocean simulation's GLA receives the execute command and parameters from the Coast Guard's GLA and sends the information to the NML server, which acts as an interface between the GLA and the ocean simulation.
7) The ocean simulation receives this information from the NML server and runs the required executable to produce the output data file.
8) This output file is then sent to the ocean simulation's local GLA, which then sends it to the Coast Guard's GLA, which in turn sends it to the Coast Guard simulation.
9) The Coast Guard simulation uses the trajectory information to find and rescue the survivors.

Using a real-world scenario, our prototype demonstrates a transparent, dynamic, and real-time integration of simulations using an agent-based framework. The following points summarize the results of this prototype.

1) The simulations were linked in real time using our software agent-based framework. Specifically, the Coast Guard simulation was able to request data dynamically without prior knowledge of the location of the ocean simulation or the methods of the ocean simulation.
2) The GLAs hide the framework implementation details from the participating simulations.
3) The broker was hard-coded for the purposes of this demonstration. When fully developed, the broker will not only transparently connect the entities in the cloud but will also match the producers and consumers using a language description format.

Although helper agents were not used in this example, it is easy to see how they could be used. For example, the search-and-rescue team is interested in two pieces of data: the path taken by a drifting object and the point of maximum survivability along that path. The survivability point is measured as a function of the time spent in the water and the temperature of the water along the path. While the ocean forecasting model does not provide the survivability point directly, it does provide the data needed to calculate this value. In the current implementation, it is necessary to retrieve this data from the ocean forecasting model and then compute the survivability point locally. The use of helper agents could reduce the total data transmission by calculating this value on the server (ocean simulation) side rather than the client (Coast Guard) side. The helper agent would interface to the NML GLA in the same fashion as the Coast Guard GLA currently does, retrieve the needed information, calculate the desired values, and then return this information to the Coast Guard GLA. From the client's viewpoint, the helper agent extends the functionality of the server. The net effect allows for selective and efficient data transfer between server and client.

VII. Conclusion and Future Work

We have presented the framework and initial implementation of our agent-based system for linking distributed simulations. Our simple prototype, though incomplete with respect to the development of all the components, demonstrated the effectiveness of using an agent framework to link simulations together dynamically at run time. In addition, it showed that an existing simulation written with no knowledge of our system can be added to it.

In order to enter the simulation cloud, an entity must have an interface to the GLA. This interface is the single component that must be customized at the end-user level. We have kept this interface as simple as possible so that minimal effort will be required on the entity's part to use our system. For the ocean forecasting simulation, we added a simple server interface to enable communication between the ocean simulation and its GLA.

In the traditional approach involving interaction between different entities, any simulation wishing to communicate with another simulation or data provider must know in advance where and when it will be required to do so. In many cases, such communication requires that the participating simulations be written to meet a given standard. In addition, a considerable amount of network bandwidth is typically used in order to transfer data from one entity to another.

Our approach allows an entity to request data dynamically at any time through its local GLA. The broker will facilitate the matching and the resulting data will be transferred to the requesting entity. Although our system requires some overhead in the form of the broker and local GLAs, this overhead places little burden on the CPU and memory systems. The network bandwidth used will be comparable to the traditional approach, and the performance of our system will increase when a mobile helper agent is used. The fact that the simulations can request information on the fly, without being aware of the location of the provider, overrides the minimal overhead incurred by the simulation cloud.

When it is developed, the broker will play the central role in making the system dynamic at run time. We are currently investigating various approaches to develop this module and also the use of a universal description language by which every entity can describe what it produces and consumes without any ambiguity.

Our agent-based framework for linking distributed simulations opens new avenues of possibilities to dynamically integrate simulation systems and data providers at run time. We look forward to developing more-advanced implementations that include a full-fledged brokering system, sensors as data producers, etc.

While the basic functionality required of agents for distributed simulation systems is already present in the D'Agents system, much work remains to be done. Specifically, we list several topics of ongoing research.

- *Security*: There are many issues at stake in allowing foreign programs to migrate to a machine and execute there. Mechanisms for authenticating agents and servers can be built on top of public key encryption systems [18], but much work remains to be done on dealing with multiple administrative domains and related topics.
- *Resource Control*: This is closely related to the issue of security but deals with the problem of how many resources a mobile agent is allowed to use on a remote agent server. For example, how much memory, disk space, and CPU cycles can an agent use, even if it is authenticated properly. This is an area of ongoing research.
- *Functional Validation*: Once a simulation or data product has been located, some form of validation would be appropriate. This is distinct from *certification*, which is still ontology and keyword based. How does a simulation commit to using the results of another simulation or data resource? Should some sort of functional validation be required first? What form does that validation take? For example, if one simulation requires ocean current data and another simulation offers ocean circulation data, how can we validate beyond the symbolic level of keywords and ontologies, at the functional level?

Acknowledgment

The authors would like to thank G. Cybenko, R. S. Gray, D. R. Lynch, M. Kiczo, and B. Cushman-Roisin for their valuable feedback and assistance in various aspects of this work. The authors would also like to thank the reviewers for their productive comments.

References

[1] Aggregate Level Simulation Protocol (ALSP) [Online]. Available: http://alsp.ie.org/alsp/alsp.html

[2] M. Baldi and G. P. Picco, "Evaluating the tradeoffs of mobile code design paradigms in network management applications," in *Proc. 1998 Int. Conf. Software Engineering*, pp. 146–155.

[3] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus, "Mobile agents in distributed information-retrieval," in *Intelligent Information Agents*, M. Klusch, Ed. New York: Springer-Verlag, 1999.

[4] A. Buss and L. Jackson, "Distributed simulation modeling: A comparison of HLA, CORBA, and RMI," in *Proc. 1998 Winter Simulation Conf.*, pp. 819–825.

[5] CORBA [Online]. Available: http://www.corba.org

[6] M. G. Corr and C. Okino, "Networking reconfigurable smart sensors," in *Proc. SPIE: Enabling Technologies for Law Enforcement and Security*, Nov. 2000, vol. 4232.

[7] D'Agents [Online]. Available: http://agent.cs.dartmouth.edu

[8] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The DoD high level architecture: An update," in *Proc. 1998 Winter Simulation Conf.*, pp. 797–804.

[9] W. Emmerich, "An overview of OMG/CORBA," in *Proc. Inst. Elec. Eng. Colloq. Distributed Object Technology Application*, 1997, pp. 1/1–1/6.

[10] J. Farley, *Java Distributed Computing*. Cambridge, MA: O'Reilly, 1998.

[11] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an agent communication language," in *Proc. 3rd Int. Conf. Information Knowledge Management*, 1994, pp. 456–463.

[12] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A directory service for configuring high-performance distributed computations," in *Proc. 6th IEEE Int. Symp. High Performance Distributed Computing*, 1997, pp. 365–375.

[13] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit," *Int. J. Supercomput. Applicat.*, vol. 12, no. 2, pp. 115–128, 1997.

[14] ——, "The Globus project: A status report," in *Proc. 7th Heterogeneous Computing Workshop*, 1998, pp. 4–18.

[15] Foundation of Intelligent Physical Agents (FIPA) [Online]. Available: http://www.fipa.org

[16] Globus [Online]. Available: http://www.globus.org

[17] R. S. Gray, "Agent Tcl: A transportable agent system," in *Proc. CIKM Workshop Intelligent Information Agents, 4th Int. Conf. Information Knowledge Management (CIKM 95)*, Baltimore, MD, Dec. 1995.

[18] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, "Agent Tcl," in *Itinerant Agents: Explanations and Examples with CD-ROM*, W. Cockayne and M. Zypa, Eds. Greenwich, CT: Manning, 1997.

[19] R. S. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko, "Mobile agents: The next generation in distributed computing," in *Proc. 2nd Aizu Int. Symp. Parallel Algorithms/Architectures Synthesis (pAs '97)*, Fukushima, Japan, Mar. 1997, pp. 8–24.

[20] R. S. Gray, D. Kotz, G. Cybenko, and D. Rus, “D’Agents: Security in a multiple-language, mobile-agent system,” in *Mobile Agents and Security*, G. Vigna, Ed. New York: Springer-Verlag, 1998, pp. 154–187.
[21] R. S. Gray, “Agent Tcl: A flexible and secure mobile-agent system,” Ph.D. dissertation, Dept. Comput. Sci., Dartmouth College, Hanover, NH, June 1997.
[22] R. S. Gray, G. Cybenko, D. Kotz, and D. Rus, “Mobile agents: Motivation and state-of-the-art,” in *Handbook of Agent Technology*, J. Bradshaw, Ed. Cambridge, MA: AAAI/MIT Press, 2000.
[23] E. R. Harold, “Java network programming,” O’Reilly, Cambridge, MA, 1997.
[24] High Level Architecture (HLA) [Online]. Available: http://hla.dmso.mil
[25] IBM Aglets [Online]. Available: http://www.trl.ibm.co.jp/aglets
[26] D. Johansen, “Mobile agent applicability,” in *Mobile Agents ’98*, K. Rothermel and F. Hohl, Eds. New York: Springer-Verlag, 1998, pp. 80–98.
[27] C. Kahn, P. Porras, S. Staniford-Chen, and B. Tung, “A common intrusion detection framework,”, submitted for publication.
[28] *Knowledge Interchange Format, American National Standard (dpANS) NCITS.T2/98-004* [Online]. Available: http://logic.stanford.edu/kif/dpans.html
[29] D. Kotz and R. S. Gray, “Mobile agents and the future of the Internet,” *Oper. Syst. Rev.*, vol. 33, pp. 7–13, Aug. 1999.
[30] D. B. Lange and M. Oshima, “Seven good reasons for mobile agents,” *Commun. ACM*, vol. 42, pp. 88–90, Mar. 1999.
[31] Mitsubishi Electric’s Concordia [Online]. Available: http://www.meitca.com/HSL/Projects/Concordia
[32] W. D. Nowlin, Jr., “US ocean science needs for modeling and data synthesis: Status of a common community assessment,” *Oceanography*, vol. 10, no. 3, pp. 135–140, 1997.
[33] “NSF: Assessing ocean modeling and data assimilation requirements 1,” U.S. WOCE Office Rep., College Station, TX, Feb. 1997.
[34] “NSF: Assessing ocean modeling and data assimilation requirements 2,” U.S. WOCE Office Rep., College Station, TX, Apr. 1997.
[35] D. Rus, R. Gray, and D. Kotz, “Autonomous and adaptive agents that gather information,” in *Proc. AAAI ‘96 Int. Workshop Intelligent Adaptive Agents*.
[36] S. Staniford-Chen, B. Tung, and D. Schnackenberg, “The common intrusion detection framework (CIDF),” in *Information Survivability Workshop*, Orlando, FL, Oct. 1998.
[37] J. Steinman, “SPEEDES: A multiple-synchronization environment for parallel discrete-event simulation,” *Int. J. Comput. Simul.*, vol. 2, no. 3, pp. 251–286, 1992.
[38] M. Strasser and M. Schwehm, “A performance model for mobile agent systems,” in *Int. Conf. Parallel Distributed Processing Techniques Applications*, vol. 2, 1997, pp. 1132–1140.
[39] H. Suguri, “A standardization effort for agent technologies: The foundation for intelligent physical agents and its activities,” in *Proc. 32nd Hawaii Int. Conf. Syst. Sciences*, 1999.
[40] University of Maryland Baltimore County KQML [Online]. Available: http://www.csee.umbc.edu/kqml
[41] F. Wieland, E. Blair, and T. Zukas, “Parallel discrete-event simulation (PDES): A case study in design, development, and performance using SPEEDES,” in *Proc. 9th Workshop Parallel Distributed Simulation (PADS ’95)*, June 1995, pp. 103–110.
[42] L. F. Wilson, D. J. Burroughs, J. Sucharitaves, and A. Kumar, “An agent-based framework for linking distributed simulations,” in *Proc. 2000 Winter Simulation Conf.*, Dec. 2000, pp. 1713–1721.

Linda F. Wilson (Member, IEEE) received the B.S. degree in mathematics from Duke University, Durham, NC, in 1988, and the M.S.E. and Ph.D. degrees in electrical and computer engineering from the University of Texas, Austin, in 1990 and 1994, respectively.

From 1994 to 1996, she was a Staff Scientist with ICASE at the NASA Langley Research Center, where she conducted research in parallel discrete-event simulation. Her research interests include distributed simulation, software agent systems, and computer security. She is currently the Clare Boothe Luce Assistant Professor of Engineering at the Thayer School of Engineering, Dartmouth College, Hanover, NH.

Dr. Wilson is a member of the IEEE Computer Society, ACM, and SWE.

Daniel J. Burroughs (Student Member, IEEE) received the B.S. degree in computer engineering from the University of Central Florida (UCF) in 1995. He is currently a Ph.D. candidate at the Thayer School of Engineering, Dartmouth College, Hanover, NH.

While at UCF, he worked on virtual reality systems for military training at the Institute for Simulation and Training. After leaving UCF, he worked for American Laserware, developing high-speed motion control software and user/programmer interfaces for laser engraving systems. His current research is in distributed network intrusion detection and the application of Bayesian hypothesis tracking to identify distributed, coordinated attack efforts.

Anush Kumar received the B.E. degree in computer science and engineering from Venkateshwara College at the University of Madras in 1999. He is currently a M.S. candidate at the Thayer School of Engineering, Dartmouth College, Hanover, NH. His research interests include distributed object systems, computer networks, and mobile agent systems.

Jeanne Sucharitaves received the A.B. degree in engineering sciences modified with computer science from Dartmouth College, Hanover, NH, in 1999. She is currently a M.S. candidate at Dartmouth College’s Thayer School of Engineering. Her research focuses on mobile agent systems and distributed simulation.