Experimental Interest Management Architecture for DCEE

Bill Helfinstine, Mark Torpey
Lockheed Martin Information Systems
Burlington, MA
Bill.Helfinstine@lmco.com, Mark.Torpey@lmco.com

Gene Wagenbreth
Information Sciences Institute
Marina Del Rey, CA
genew@isi.edu

ABSTRACT

The Distributed Continuous Experimentation Environment (DCEE) is a permanent simulation system and facility that is being designed and assembled by the US Joint Forces Command (USJFCOM) to provide a capability to do simulation-backed experimentation without incurring heavy integration and ramp-up costs. Among the several thrusts of the DCEE system is the capability to do large-scale human-in-the-loop experiments in the spirit of the Millennium Challenge 2002 experiment, as well as very detailed representations of joint urban operations scenarios. Additionally, the DCEE system will be used in support of a number of smaller-scale experiments and training events, such as Limited Objective and Multinational Experiments.

In order to provide a system that can scale to a richer and more expansive world, we need to increase the computational power available to produce the environment. However, this leads to a classical problem of parallel computation, where the communications requirements of the system become the bottleneck, and additional computation adds no additional capacity to the system.

This paper describes the architecture that we have prototyped to address some of the problems of data communications scalability. It discusses the interest management techniques that have been used in the past, and how those experiences influenced the prototype design. It talks about the technology that provides finer resolution interest management than simulations have had in the past while allowing better scalability. It explains the limitations of the prototype system and discusses some possible approaches to addressing them. Finally, it describes some likely future requirements of the DCEE system, and talks about how the architecture would have to change in response.

ABOUT THE AUTHORS

BILL HELFINSTINE is a federation developer for the USJFCOM J9 Experiment Engineering Department and a developer and integrator of JSAF (Joint Semi-Automated Forces), as well as primary maintainer and developer of the RTI-s experimental RTI. He has worked in M&S for 9 years, with the last several in support of JFCOM-sponsored exercises, culminating in Millennium Challenge 2002. He is a Staff Software Engineer at Lockheed Martin Information Systems Advanced Simulation Center (LMIS-ASC) in Burlington MA. He received his B.S. in Computer Science and Engineering at the Massachusetts Institute of Technology.

MARK TORPEY is a federation developer for the USJFCOM J9 Experiment Engineering Department and the lead developer and integrator of JSAF (Joint Semi-Automated Forces). His 8 years of M&S experience have been largely in support of JFCOM-sponsored exercises including Millennium Challenge 2002, Unified Vision 2001, and Attack Operations 2000, as well as the DARPA STOW program. He is a Staff Software Engineer at Lockheed Martin Information Systems Advanced Simulation Center (LMIS-ASC) in Burlington MA. He received his M.S. and B.S in Computer Science at the University of Massachusetts.

GENE WAGENBRETH is a parallel processing systems analyst with Information Sciences Institute in Marina Del Rey CA. He has 30 years experience with a range of applications on parallel processors and supercomputers. He received his B.S. in Mathematics and Computer Science at the University of Illinois at Urbana-Champaign.

Experimental Interest Management Architecture for DCEE

Bill Helfinstine, Mark Torpey
Lockheed Martin Information Systems
Burlington, MA
Bill.Helfinstine@lmco.com, Mark.Torpey@lmco.com

Gene Wagenbreth
Information Sciences Institute
Marina Del Rey, CA
genew@isi.edu

THE DCEE

The Distributed Continuous Experimentation Environment (DCEE) is a facility and a capability being designed and assembled by the US Joint Forces Command (USJFCOM) for exploration of concepts in joint warfighting. (Ceranowicz et al 2003) One major component of the DCEE is a permanent simulation installation to provide the capability to do simulation-backed experimentation without incurring the major integration and ramp-up costs that previous experiments have incurred.

The simulation capability that will be provided by the DCEE is based on the Joint Experimentation Federation (JEF) that was assembled for the Millennium Challenge 2002 (MC02) experiment. (Ceranowicz et al 2002) This federation provides a framework for the individual services to bring simulation capabilities to a joint virtual world. It provides concept developers the ability to experiment with large-scale battles and situations in a platform-level human-in-the-loop style.

However, the DCEE is not simply designed to be a snapshot of the MC02 version of the JEF; it is designed to evolve and expand to encompass new capabilities and fulfill new requirements as they arise. Therefore, we must keep pushing the technology in advance of the requirements, or the DCEE won't be used. The whole point of the DCEE simulation system is to make it easy and quick to set up a situation and simulate it, in a brainstorming style, in order to bring Joint Experimentation to its full potential.

The simulation component of the DCEE is implemented as a High Level Architecture (HLA) federation. (Dahmann et al 1997) It is an aggregation of a number of simulation systems, each of which has a particular focus on a different facet of the battlefield. However, since many of the simulations that make up the DCEE were originally designed to interoperate using the DIS protocols, (IEEE 1998) they were designed to support scenarios that are in the size range that is supported by DIS—which typically has an upper bound on the number of simulated platforms in the low thousands. This leads to another issue for the DCEE, that of providing a simulation capability that can handle the progressively larger scenarios that the DCEE is designed to handle.

SCALABILITY

One major thrust in the world of Joint Experimentation is that of scalability. The simulated world of MC02, while larger than any others created previously, is not big enough or detailed enough to play out the situations that JFCOM wants to examine. The DCEE must be able to provide a larger, more detailed environment, both in terms of numbers of simulated actors, and the simulated natural environment they interact within.

Table 1. Scalability Achievements Over Time

Event	Object Count	Max Objs Produced PerFederate	Max Objs Consumed PerFederate
STOW97	7000	400	500
J9901	40,000	5000	5000
AO00	160,000	20,000	50,000
MC02	50,000	30,000	30,000
SPP	1,500,000	15,000	70,000

As is shown in Table 1, the number of simulated objects that make up large federations has been steadily increasing, with the exception of MC02, in which it was more important to integrate a large number of new models. The trend towards larger numbers of objects will continue as we move forward, simply because the simulations are not yet capable of portraying a full-scale situation at full accuracy. With the increasing capabilities of computers and networks, we believe that we will be able to produce such a full-scale scenario, but there are still a large number of open issues remaining, both in how to properly control such a simulation, and in how to usefully use and observe such a simulation.

There are a number of factors that limit the size of the simulation that we are able to produce in a system like the DCEE. The computation cost of simulating the objects that make up the world and the interactions between them is the most straightforward of these factors. The communications overhead of sharing these objects between the various simulators introduces another major cost. The final and most complex factor is the effect of the objects simulated by other systems on the local simulated objects.

The computational factor is a straightforward problem to solve, simply by adding additional hardware to the system. However, this solution is constrained due to space restrictions and financial restrictions, and therefore the system is limited in how large and how quickly it can grow.

The communications factor is also constrained by the amount of funding available to support the networking hardware and services to run an exercise. Additionally, there is a significant lead time requirement in order to get networks provisioned and security requirements fulfilled.

The cost of incoming data is still a major subject of research. Each remote vehicle that is received by a simulator adds load to that system. One technology that is often used to reduce the load on a system is interest management. This lets each simulator describe what data might affect its simulation, and the networking infrastructure filters the data that the system needs to consider. While interest management helps an enormous amount, there are cases where it fails simply due to the amount of data requested. (Brunett & Gottschalk 1997a)

SCALABLE PARALLEL PROCESSORS

In searching for a possible solution to the first two pieces of the scalability dilemma, we turned to another area of research that has been exploring the areas of scalability and parallelism. The scientific supercomputing community has been exploring the limits of scalability for many years. Furthermore, this community has led to the creation of government-owned and operated High Performance Computing (HPC) centers, many of which are available for use with little lead time.

The HPC centers provide a variety of types of systems, most of which fall into the general category known as Scalable Parallel Processors (SPPs). These systems are defined by their large number of individual CPUs that are connected by a high speed network.

One of the major types of systems run by the HPC centers is the Beowulf cluster. (Sterling et al 1995) Beowulf clusters have become popular systems in the world of HPC systems because of their low cost for the amount of power they provide. A typical configuration of a Beowulf cluster is several hundred commodity PCs running Linux connected with a multi-gigabit network, with custom resource allocation and parallel machine software running on them. Since these clusters are similar to the systems the DCEE uses, we decided to concentrate on using these systems to provide a huge amount of computation and communication resources, and thereby address the time, money, and space restrictions on scalability in the DCEE.

INTEREST MANAGEMENT

The third piece of the scalability question is how to handle the large quantities of data that we can now generate using the capabilities of the SPP systems. We have spent quite a bit of time optimizing the simulation systems to reduce the load imposed by incoming data, but there is an inherent polynomial factor in all simulation systems, simply because vehicles interact with nearby other vehicles, and therefore as vehicle density increases, processing per vehicle increases as well.

In particular, sensor processing is typically an $O(n^2)$ operation on vehicles in a local area. There have been several attempts to mitigate this load through alternative sensor approaches (McGarry & Torpey 1999) (Lorenzo et al 2000) (Kwak & Andrew 2002) but these approaches require additional simulation changes to support them. Due to the legacy nature of many of the DCEE simulation models, these approaches are difficult to implement, since they require modifications to all the different models that are used. This also limits the flexibility of the system, since new simulations have additional requirements over their existing capabilities in order to interoperate with the rest of DCEE.

So, it is still necessary to reduce the quantity of data coming into each simulation to the minimum that they need in order to operate correctly. In general, only the individual simulator can determine what data is interesting, and only at runtime, since the information needed is based on the situation that the simulator is modeling. Therefore, interest management is a dynamic problem, and needs to react and adapt to

changing data requirements by all the components of the system.

This dynamic interest specification and handling is performed by the Data Distribution Management (DDM) functionality of the HLA. The HLA Run Time Infrastructure (RTI) software provides an API to the simulations that allows them to specify what data is interesting in a dynamic fashion. The HLA provides a generalized, abstract way to specify data and interest, by representing data domains as multidimensional spaces, and interest and data specifications as regions within those spaces. (Morse & Steinman 1997) See Figure 1 for a two-dimensional example of how DDM represents interests and data based on overlap of regions.

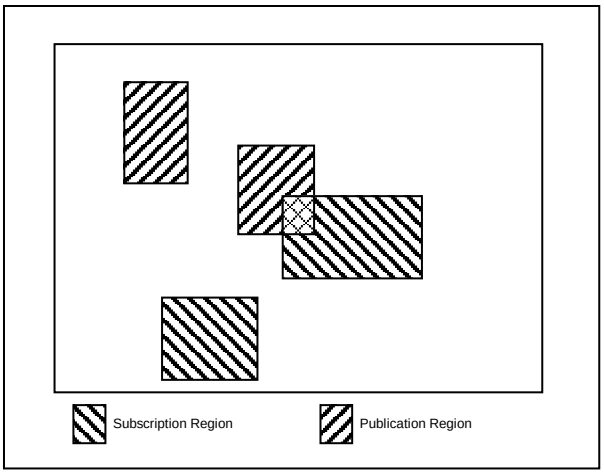

Figure 1. Subscription and Publication Overlap

There have been a number of different DDM designs in several RTI implementations, each of which represents spaces and regions in different ways. The most scalable implementation we have found so far is a statically-assigned grid representation that represents spaces as multidimensional grids, and regions into subsets of the grid. This leads to a fast mapping of interest to grids without any communications and with a simple algorithm. (Helfinstine et al 2001) See Figure 2 for an example of how Figure 1 would be represented in a fixed-gridded implementation.

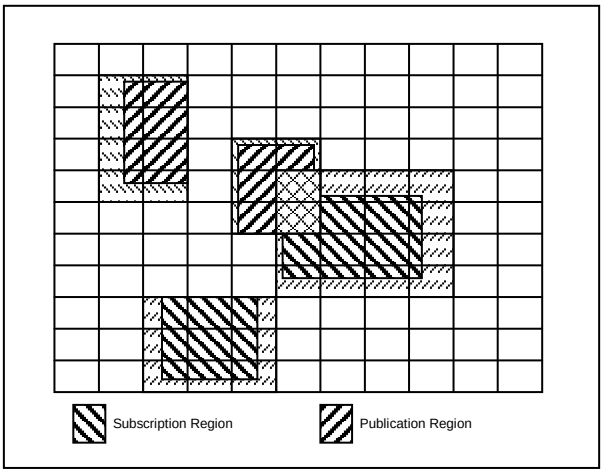

Figure 2. Regions Snapped to a Grid

MULTICASTING

The main communications capability that is provided by the HLA is a publish/subscribe capability that delivers each message to multiple receivers. This capability is often implemented using Internet Protocol (IP) multicast, (Deering 1989) which provides support for point-to-multipoint communications with dynamic subscription changes, over a range of different networking technologies.

However, this presents a problem when trying to run on SPPs, which typically do not support IP multicast. Depending on the type of SPP, it may not support IP at all, since there are many ways to interconnect processors that do not look like a traditional network.

However, SPPs do support message-passing communications, either using IP, as Beowulf systems do, or with some other technology. In order to provide a standardized means of doing message-based communications, the HPC community has standardized on the Message Passing Interface (MPI) as a common API for building parallel programs that express their parallelism in terms of messages. (MPI Forum 1995)

So, it became clear that we would need to build a mechanism that would provide the many-to-many semantics of multicasting while using a communications technology that only supports point-to-point. Furthermore, we also need to maintain our existing capability to run the simulation in a Local Area Network (LAN) environment, since user interfaces and other DCEE federates would not be supported by the SPP.

This led us to examine the work that was done for the Synthetic Forces Express (SF Express) program which investigated running the ModSAF simulation on SPP systems in the 1997 timeframe. (Brunett & Gottschalk 1997b) This work demonstrated a fairly straightforward way to provide an emulation of the capabilities of multicast using router processes running on SPP nodes to arbitrate the communications and do data duplication and forwarding to appropriate receivers. (Brunett & Gottschalk 1997a)

Another interesting body of work that uses a similar networking architecture is being pursued by the DARPA Active Networks program. (Dorsch et al 2002) In their system, software router processes perform data routing to the appropriate recipients in a similar fashion to the SF Express router nodes. This project uses direct region matching to do filtering, which is more precise, but less scalable as the number of regions increases.

RTI-S

In order to construct a system that runs on an SPP and supports HLA federations, we needed an RTI implementation that would use router processes to communicate within the federation. The particular implementation that we used to form the basis of this system is the RTI-s subset RTI implementation. (Calvin et al 1997)

We chose this implementation for several reasons. It was available to us with source code, and is familiar to us from its use in previous experiments, so it was easily modifiable to use the new communications system we were building. It has much less code than a full RTI implementation, which makes it much easier to understand and extend. It scales well, and has a fairly small memory footprint. Finally, it has a very flexible implementation of DDM, providing multiple static inset grids that allow detailed tuning of interest specifications. (Rak et al 1997)

COMMUNICATIONS ARCHITECTURE

We put together a design for the communications architecture based on the concept of stackable protocol modules. We analyzed the existing RTI-s communications code and refactored the functionality it provided into several pieces.

The original RTI-s network interface is composed of the stream manager classes, which provide single-sender to multiple-receiver message sending, receiving, and subscription, and the message buffer class, which provides an interface to messages. Below this interface, the infrastructure provides message bundling, to reduce the packet count by aggregating multiple small messages into each packet, and fragmentation, to split large messages into multiple packets and reassemble them on receive. Finally, it sends and receives the actual packets using IP multicast.

Then, these main components of the communications infrastructure were separated out into chained protocol modules, and given a standardized interface to ease extension and flexibility. We then added additional modules that send and receive packets using point-to-point TCP and point-to-point UDP. Finally, we added a module that translated generalized subscription requests into a message that states the current list of subscriptions, which is sent across the point-to-point connection and remembered by the receiver. Figure 3 shows three possible configurations for an RTI communications structure, with the three columns of protocol modules below the stream manager.

In order to operate on SPP systems that use MPI as their connectivity basis, we built an MPI send and receive module. However, we were worried about the fault-tolerance effects of MPI, and since we were running on Beowulf clusters, which support IP connectivity, we ended up using TCP for our prototype events.

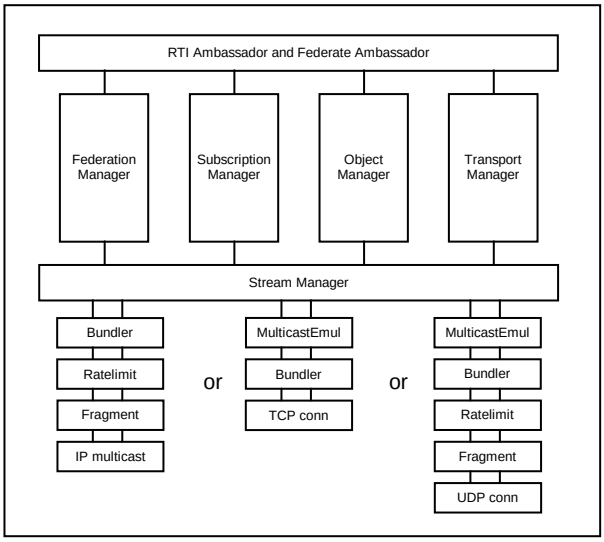

Figure 3. Three Example Configurations of the RTI-s Communications Infrastructure

This general model of protocol modules has allowed us to experiment with additional message transformations depending on our needs. In this vein, we built a module that compresses the data across a connection, when our bandwidth is low and we have available CPU time. For testing purposes, we also built a module that simulates a lossy network, and which randomly drops incoming or outgoing data with a specified loss rate. We see this as a very convenient way of integrating future data transformations as they become necessary.

ROUTER DESIGN

Once we had a way for the RTI to send and receive data in a point-to-point fashion, we needed a router implementation that would receive the data from each federate and forward it to the clients that subscribed to it. As an initial implementation, we built a simple router process that reuses the RTI's flexible connection code, receiving the data and processing it in the same fashion as the RTI. Figure 4 provides a diagram of a router that is routing between three connections.

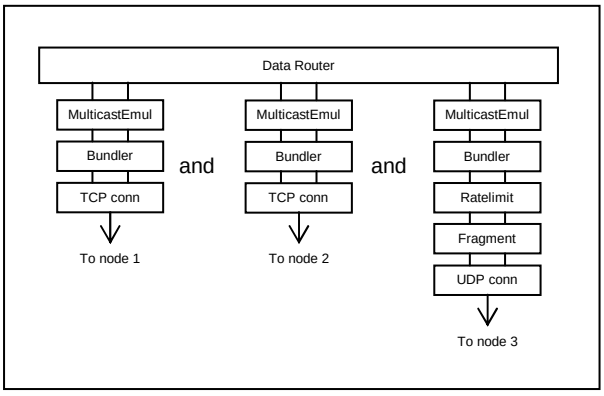

Figure 4. Simple Router Design Routing Between Connections to Three Federates or Other Routers

In order to send messages to only those receivers that want to receive it, each connection tracks what the receiver's subscriptions are. Therefore, since each connection has knowledge of what the receiver wants to hear, it can filter outgoing data before it makes it through the protocol chain. Since each side of each connection knows this information, if no listeners in the system want to hear a particular piece of data, it won't be sent out of the originating machine. This aggressive source-side squelching of data is a very nice side-effect of the router design.

However, in order to accomplish this, we need to send subscription information across each link in both directions. In the case of the router, it turns out to be fairly simple-- a router's interest is the union of all its connections' interests. Therefore, the two major things that a router must do are to forward incoming messages to all other connections, and update interest information on all other connections when one connection changes.

TOPOLOGY

This simple router architecture is quite functional, but it does have some significant problems. In particular, it does not handle cycles in the graph of routers. Each router expects to be able to forward all incoming messages to all receivers. If one of those receivers is a router that forwards a message to a router that has already forwarded it once, the routers will send data in a loop forever and overload the system. However, this implies that these routers can only be set up in a tree structure if we have more load than a single router can handle. This is obviously not a scalable design.

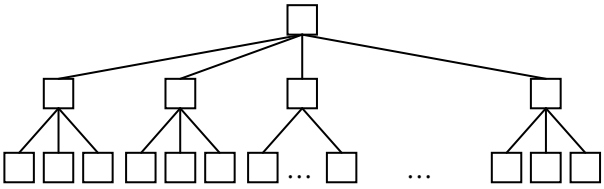

Figure 5. Simple Tree-Based Router Topology

Because of this, we also built a second router implementation based on the up/down fully-connected mesh topology that was explored by the SF Express project. Unfortunately due to schedule pressure, we have not yet been able to test this design fully.

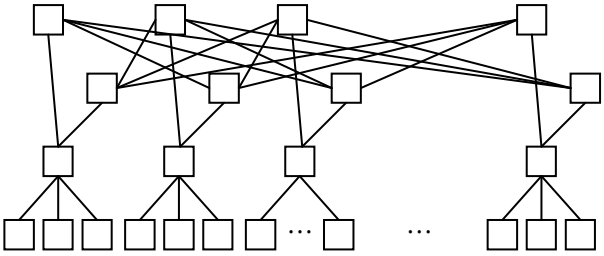

Figure 6. Triplet-Connected Mesh Topology

We believe that we need to investigate the topic of topology more, and look into new ways to organize the communications between the various components of the federation. In particular, when Wide Area Network (WAN) connections between multiple SPP systems are introduced, being constrained to a tree structure can result in very heavy data loads to one of the sites, which

is a very expensive solution to a software limitation. Further, WAN connections have a much lower bandwidth than SPP interconnects or LAN connections, and therefore it makes sense to investigate specialized connection methods across WANs, and different tradeoffs in the design of the communications setup.

INTEREST MANAGEMENT IMPROVEMENTS

One of the results of this new messaging architecture was that we began to run into limitations of the RTI-s interest management infrastructure. In particular, we wanted to expand the number of interest states from the previous maximum of 3000 to 100000 or more. Previously we were limited by the capabilities of IP multicast routers, which begin to fail after roughly 3000 multicast groups, but with our own router implementation, we no longer are subject to these limits. Since the efficacy of the static grid interest management scheme is determined by the number of interest states, the more states that are available, the smaller the grid cells are. As the grid cells become smaller, less unwanted data will be delivered to the federate.

However, the existing code began to perform poorly, due to the use of arrays of integers to represent the list of interests of a particular subscription. In order to scale the number of interest states up, we had to refit a number of internal data types in RTI-s to be more efficient, both in storage usage and in access time. In particular, the list of interest states was changed to be represented as a sparse bit vector implementation, with a fixed-block-size representation. This provided a way to quickly determine interest overlap as well as a fast means of calculating the union of interests in the router. Further, it resulted in a compact representation that could easily be sent over connections with a fairly small overhead.

Similar changes were made throughout the RTI code, in many places where the assumption was that an array of values with an entry for each interest state would be acceptable, we had to change to a tree representation or a hash table in order to not consume large amounts of memory. Additional changes were required to provide a means of associating objects with their interests in an efficient fashion.

Finally, a centralized means of recording statistics about data amounts and counts was added, in order to be able to pinpoint pieces of the system and what was causing slowdowns. With the existing RTI-s capability to examine internal information, this allows a remote, distributed debugging capability that was extremely useful in monitoring the system as it ran.

PROTOTYPE EVENTS

We ran two prototype events, in which we demonstrated that it is possible to generate enormous numbers of vehicles in a very large virtual environment, using SPP systems. Both events were run using a subset of the DCEE federation, composed of the JSAF simulation GUIs, the JSAF simulator running aircraft, ships, and ground combatants, and the JSAF clutter simulator providing background and civilian traffic.

In December 2002, we were able to generate over 1,000,000 vehicles, using a terrain that covered the entire Pacific Rim. The simulation ran on the University of Southern California's Beowulf cluster, and operators and observers were located at Joint Forces Command in Suffolk, Virginia, as well as at Information Sciences Institute in Los Angeles. We were able to use 50,000 interest states to provide a fairly precise specification of interests, in several geographically disparate simulated locations.

In March 2003, we ran an even larger event, generating over 1,500,000 vehicles on the same terrain database, but located in different areas with more terrain detail. We ran on the Huinalu Beowulf cluster at the Maui High Performance Computing Center and the ASC Beowulf cluster at Wright-Patterson AFB, with observers in Suffolk and Los Angeles again. We also increased the number of interest states to 100,000 without adverse effect.

Both of these events were focused on testing the functionality of the new system, and showed that we can indeed generate a very large simulated environment. They also demonstrated that we have quite a bit of additional work that we can do, in order to make the system viable for the end users. In particular, WAN latencies and inefficiencies in the simulation's control protocols combine to make the user interfaces very sluggish. The tree nature of the routers also became a point of failure when the system was under its heaviest loads. We suffered a number of router failures due to data overload, and we are still working to address these.

FUTURE REQUIREMENTS

We still face a number of issues that we need to resolve in order to make the use of SPP systems possible for

DCEE. The events that we have run so far show major promise, but have not yet demonstrated that we are able to fulfill the DCEE's flexibility and ease-of-use requirements yet.

The first major requirement is that we need to make it much easier for non-experts to acquire time on SPPs and execute the system on them. It currently is a fairly involved process that takes several people to accomplish. This is a major project that is already underway. (Williams & Tran 2003) An initial version of the MARCI launch and control system was tested at the March event, and it is undergoing further development and refinement.

Another major requirement is that we must have a way for simulations running on SPP systems to participate in the DCEE federation. The primary reason we cannot simply plug the SPP systems into the DCEE is that the DCEE uses the enhanced version of RTI-NG developed for Millennium Challenge 2002, (Hyett & Wuerfel 2003) and the SPP uses RTI-s with point-to-point routers. Since they use different RTI implementations, they run in two separate HLA federations, and we need to build a federation gateway that will allow us to bridge data back and forth between the two federations. This is not an easy task (Granowetter 2003) but we believe that we can build such a gateway as long as its scope is restricted to the DCEE and similar federations. This is another ongoing major project.

An additional issue that we are beginning to investigate is how the SPP will help analysts do After Action Review of the huge amounts of data that can be produced by simulations running on an SPP. A distributed logging and query system is currently being designed to attempt to address this requirement.

One of the most important areas that we need to investigate is the issue of control. As we scale up scenarios to the desired sizes, it becomes more and more difficult to control the simulation and make sure it behaves in a proper fashion. We need to look into schemes that reduce the amount of operator control that is required to run a simulation. This would have an additional benefit for DCEE as well, since any technique that reduces the number of personnel involved will be of incredible utility.

CONCLUSIONS

The use of Scalable Parallel Processor systems has a great deal of promise in building larger and more detailed virtual environments, both for experimentation and for many other uses of simulation. We are integrating the use of SPP systems into the DCEE, and we believe that it will provide an extremely valuable asset in the DCEE environment.

The use of software interest management routers to provide data distribution gives us a great deal of flexibility in building a scalable system and providing the building blocks to more detailed dataflow control and management.

There are many additional dimensions that are worth exploring, both in better integration into DCEE, and additional technical exploration to discover new ways to apply the SPP assets to the problems of DCEE and similar human-in-the-loop simulation systems.

REFERENCES

Brunett, S. & Gottschalk, T. (1997a). An Architecture for Large ModSAF Simulations Using Scalable Parallel Processors, *Center for Advanced Computing Research Technical Report* CACR-155.

Brunett, S. & Gottschalk, T. (1997b). Scalable ModSAF Simulations With More Than 50,000 Vehicles Using Multiple Scalable Parallel Processors, *Center for Advanced Computing Research Technical Report* CACR-156.

Calvin, J., Chiang, C., McGarry, S., Rak, S., Van Hook, D., & Salisbury, M. (1997). Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s), *Proceedings of the Spring 1997 Simulation Interoperability Workshop*, Paper 97S-SIW-019.

Ceranowicz, A., Torpey, M., Helfinstine, B., Evans, J., & Hines, J. (2002). Reflections on Building the Joint Experimental Federation, *Proceedings of the 2002 Interservice/Industry Training, Simulation and Education Conference*.

Ceranowicz, A., Dehncke, R., Cerri, T., & Blank, J., (2003). Moving toward a Distributed Continuous Experimentation, *Submitted to the 2003 Interservice/Industry Training, Simulation and Education Conference*.

Dahmann, J., Fujimoto, R., & Weatherly, R. (1997). The Department of Defense High Level Architecture, *Proceedings of the 1997 Winter Simulation Conference*.

Deering, S. (1989). *Host Extensions for IP Multicasting*, IETF Network Working Group, RFC 1112.

Dorsch, M., Kostas, T., & Skowronski, V. (2002). Reducing Bandwidth Requirements of Distributed Simulations, *Proceedings of the Fall 2002 Simulation Interoperability Workshop*, Paper 02F-SIW-118.

Granowetter, L. (2003). RTI Interoperability Issues - API Standards, Wire Standards, and RTI Bridges, *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Paper 03S-SIW-063.

Helfinstine, B., Wilbert, D., Torpey, M., & Civinskas, W. (2001). Experiences with Data Distribution Management in Large-Scale Federations, *Proceedings of the Fall 2001 Simulation Interoperability Workshop*, Paper 01F-SIW-032.

Hyett, M. & Wuerfel, R. (2003). Connectionless Mode and User Defined DDM in RTI-NG V6, *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Paper 03S-SIW-102.

IEEE (1998). *IEEE Standard for Distributed Interactive Simulation - Application Protocols*, IEEE Std 1278.1A-1998.

Kwak, D. & Andrew, E. (2002). Technical Challenges for Joint Synthetic Battlespace (JSB), *Proceedings of the Fall 2002 Simulation Interoperability Workshop*, Paper 02F-SIW-075.

Lorenzo, M., Morse, K., Riggs, B., & Rizik, P. (2000). Sensor Simulation Scalability Using Composable Component Federates, *Proceedings of the Fall 2000 Simulation Interoperability Workshop*, Paper 00F-SIW-090.

McGarry, S. & Torpey, M. (1999). Back to Basics: Balancing Computation and Bandwidth, *Proceedings of the Fall 1999 Simulation Interoperability Workshop*, Paper 99F-SIW-188.

Morse, K. & Steinman, J. (1997). Data Distribution Management in the HLA: Multidimensional Regions and Physically Correct Filtering, *Proceedings of the Spring 1997 Simulation Interoperability Workshop*, Paper 97S-SIW-052.

MPI Forum (1995). *MPI: A Message Passing Interface Standard*, Version 1.1.

Rak, S., Salisbury, M., MacDonald, R. (1997). HLA/RTI Data Distribution Management in the Synthetic Theater of War, *Proceedings of the Fall 1997 Simulation Interoperability Workshop*, Paper 97F-SIW-119.

Sterling, T., Becker, D., Savarese, D., Dorband, J., Ranawake, U., & Packer, C. (1995). Beowulf: A Parallel Workstation for Scientific Computation, *Proceedings of the 24th International Conference on Parallel Processing*.

Williams, R. & Tran, J. (2003). Supporting Distributed Simulation on Scalable Parallel Processor Systems, *Submitted to the 2003 Interservice/Industry Training, Simulation and Education Conference.*