Integrating Web, Desktop, Enterprise and Military Simulation Technologies To Enable World-Wide Scalable Televirtual (TVR) Environments.

G.C. Fox, W. Furmanski, B. Natarajan, H. T. Ozdemir,
Z. Odcikin Ozdemir, S. Pallickara and T. Pulikal
Northeast Parallel Architectures Center(NPAC) at Syracuse University
{gfc, furm, balaji, timucin, zeynep, shrideep, tapulika}@npac.syr.edu

February 27, 1998

Abstract

We present an approach to the next generation televirtual (TVR) environments that integrate collaboration with distributed computing and modern modeling and simulation technologies. We follow the 3-tier architecture with the Web Object (Java/CORBA) based middleware, VRML/Java3D/DirectX based front-ends and JDBC/PSS/OLEDB based back-ends and we are testing our design and the integration concepts by prototyping a multi-user authoring and runtime environment to support WebHLA based distributed military simulations. We present first our taxonomy of collaboratory frameworks and our integration paradigm, based on the WebFlow system at NPAC. We then list the critical enabling technologies that are being integrated and finally we summarize the current status of our prototyping experiments.

1. Introduction

Support for collaboration was one of the main driving forces for the original Web, created by CERN physicists to share data across high energy physics labs. Over the last decade, the Web technologies rapidly evolved and so did the collaboration capabilities of the whole framework. Natural initial focus was on asynchronous collaboratory models (shared databases) which were more recently augmented by the synchronous components (shared displays) and are now further evolving towards the ultimate televirtual (TVR) (shared worlds) environments. Some essential Web technology thresholds towards TVR included: VRML for 3D interactive front-ends and Java that enabled real-time synchronous connections.

First generation systems were represented by custom Java based collaboratory server technologies such as NCSA Habanero or JavaSoft JSDA. The recent onset of distributed object and/or component technologies opens new interesting avenues for the standards based collaboratory infrastructure. However, selecting a specific direction in the exploding field of distributed object and component technologies is not an easy task. CORBA offers one promising approach, especially from the large scale computing perspective but there are alternatives such as Java RMI or Microsoft DCOM. Finally, the World-Wide Web Consortium is developing a set of new standards such as XML, DOM, RDF and HTTP-NG which, when combined, can be viewed as yet another, new emergent distributed object model (sometimes referred as WOM [1]) which is likely easier to be adopted by simple to medium complex distributed object/component applications.

Recent OMG/DARPA workshop on compositional software architectures [1] illustrated very well both the growing momentum and the multitude of options and the uncertainty of the overall direction in the field. A closer inspection of the distributed object/component standard candidates indicates that, while each of the approaches claims to offer the complete solution, each of them in fact excels only in specific selected aspects of the required master framework. Indeed, it seems that WOM is the easiest, DCOM the fastest, pure Java the most elegant and CORBA the most complete solution.

In our approach towards TVR, we adopt the integrative methodology i.e. we setup a multiple-standards based framework in which the best assets of various approaches cumulate and cooperate rather than compete. We start the design from the middleware which offers a core or a 'bus' of modern 3-tier systems and we adopt Java as the most efficient implementation language for the complex control required by the multi-server middleware. We adopt CORBA as the base distributed object model at the Intranet level, and the (evolving) Web as the world-wide distributed (object) model. System scalability requires fuzzy, transparent boundaries between Intranet and Internet domains which therefore translates into the request of integrating the CORBA and Web technologies. We implement it by building a Java server which handles multiple network protototcols and includes support both for HTTP and IIOP. On top of such Object Web software bus, we implement specific computational and collaboratory services.

2. Taxonomy Of Collaboratory Frameworks

The most straightforward is an *asynchronous* collaboratory framework, typically offered by Web linked relational databases. Participants interact in such a shared space at their own pace, using some suitable connection identifiers to maintain the session identity across subsequent client-server connections. The atomicity, concurrency, security and other transactional capabilities of the database ensures consistency of the shared space. Our two projects, one in healthcare domain (Careweb) and another one in distance education (Language Connect University - LCU) used this type of collaboratory system based on CGI-extended web servers with tools for creating and editing shared documents in the database. In Careweb, we used a centralized Oracle database of student health records to bring various members of the school health community (school nurses, nurse practioners, pediatricians, parents etc.) together. LCU also used CGI-based approach for bringing a virtual classroom of students and instructors together over the web. It used a hypermedia database where students can take lessons, quizzes, exams etc. and instructors can grade these submissions and respond back to the students in various different ways. Support was also there for a Web/Oracle based customized mailing facility wherein various members of the virtual university can interact with others in a group or on a individual basis.

The onset of Java and ActiveX brought full dynamic/remote programmability to the client/Web browser side and hence the support for the socket based active connections between clients and collaboratory servers. This enabled *real-time synchronous* collaboratories, typically implemented as a collection of Java applets, maintaining active connections to the common server-side shared/collaboratory object and updating their displays within some synchronous communication model such as chat, whiteboard, webcast etc. The current first generation of such systems offers various API solutions, ranging from simple generic collaboratory Channel model of JSDA by JavaSoft, to the full collaboratory applet (hablet) model of Habanero by NCSA. In our collaboratory experiments, we explored both JSDA and Habanero models from the perspective of their relevance for TVR collaboratory environments.

Televirtual (TVR) frameworks are based on the shared virtual world metaphor, popular in the networked virtual reality community. Hence, TVR can be viewed as a natural merger of asynchronous and synchronous collaboratory technologies discussed above. Participants get engaged in synchronous collaboration within their individual 'reality windows' whereas the rest of their worlds follows the asynchronous update and access model. Naively, one could implement TVR simply as a central server based synchronous collaboratory with clipping but such solution clearly does not scale towards truly large multi-user worlds. A natural scalable solution is given by a peer-to-peer or a multi-server model in which each participant joins the world with her/his own 'personal' server. Distributing an active multi-user virtual world over a set of servers raises in turn several non-trivial synchronization issues. Such parallel or distributed simulation environments were actively investigated during the last decade by the DoD. A set of promising solutions and standards is now available, including DIS, SPEEDES and most recently HLA/RTI. These systems offer sophisticated algorithms for managing logical simulation time and handling the distributed synchronization such as dead reckoning in DIS, Global Virtual Time or Breathing Time Warp in SPEEDES, and Data Distribution Management in HLA/RTI.

3. Towards Non-Trivial Collaboratory Applications

Simple Web based collaboratory applications are already being used by specific comunities such as online gaming, chat rooms, or distance learning. Here, we are trying to address a more complex question of what comprises a useful collaboratory application for an enterprise or a research lab. Such institutions often use computers to address and solve some complex problems in their domain. Hence, for collaboration to succeed, it needs to be made part of the mainstream computational framework used by an organization, rather then a standalone or add-on capability, which is typical to the current generation systems.

Such integration of collaboration and computation can be most naturally addressed within the current models of distributed objects such as CORBA, Java RMI or DCOM. Indeed, a shared entitity such as a CORBA object can act as a natural collaboratory channel, while at the same time it can be associated with some server side processing or it can interact with other computational CORBA objects. One example of such approach is given by the WebFlow [2][3] environment under development at NPAC.

WebFlow is a 3-tier distributed visual dataflow model. WebFlow front-end is given by a Java applet that offers a visual interactive tool for dataflow authoring. WebFlow middleware is given by a mesh of Java Web Servers that offer servlets based management of the distributed computation: Session Manager that interacts with the editor applet, Module Mamanger that instantiates computational modules represented as visual dataflow nodes, and Connection Manager that communicates with other WebFlow servers to form distributed computational meshes of WebFlow modules. WebFlow Module is a Java Object which implements webflow.Module interface with three methods: init(), run() destroy(). WebFlow backend is currently left open for experimentation with various computational (e.g. HPF, RDBMS) and collaboratory (e.g. JSDA) paradigms. This allows us to put computational and collaboratory components of a complex computational environments into the common framework of visual dataflow authoring. Some natural collaboratory applications under development within this model include: collaboratory visual software engineering (collaboratory UML front-ends); simulation based design (collaboratory VRML authoring), visual HLA simulation tools (collaborative authoring of federations or federate objects for wargaming simulations).

4. Enabling Technologies

We follow the 3-tier architecture with the distributed object/componentware based middleware, visual front-ends and suitable simulation, computation or information/database objects in the back-ends.

In the front-end, the most promising collaboratory technologies include: Java3D, VRML, DirectX, XML. Is seems that Java3D and VRML communities will join their efforts, but it isn't clear yet how these technologies are to be integrated with the Microsoft approach based on XML application languages on top of DirectX engines.

Virtual Reality Modeling Language (VRML) is a file format that defines the layout and content of a 3D world with links to more information. The Java 3D API is an application programming interface used for writing stand-alone three-dimensional graphics applications or Web-based 3D applets. DirectX is a group of technologies designed by Microsoft to make Windows-based computers an ideal platform for running and displaying applications rich in multimedia elements such as full-color graphics, video, 3D animation, and surround sound.

In the middleware, we are exploring CORBA, Java/RMI and DCOM and we are addressing their integration towards a uniform Object Web software bus. We are also augmenting this generic layer of distributed objects and components by specialized DoD technologies such as SPEEDES, DIS and HLA/RTI in the distributed simulation domain. These techologies are being discussed in the following sections in details.

In the back-end, we are analyzing simulation object technologies such as HLA/FOM or HLS/SOM, the generic CORBAservices and CORBAfacilities, and the transparent persistence technologies such as Java JDBC, CORBA PSS or Microsoft OLEDB which are still evolving but essential for building large and heterogeneous and yet scalable worlds.

JDBC API from JavaSoft, is a standard SQL database access interface for accessing heterogeneous databases from Java programs. It encapsulates the various DBMS vendor proprietary protocols and database operations and enables applications to use a single high level API for homogeneous data access. With the integration of Java and databases, many visually challenging, collaborative applications can be developed with ease.

One of OMG's Corba Object Services, Persistent State Service (PSS), addresses the issues of making persistent CORBA objects across machines, platforms and datastores. PSS provides the platform for storing and managing distributed business objects over heterogeneous datastores in a reliable and scalable manner for general and common shared use. PSS makes use of other CORBA services like Transaction Service and features like portable object adapter, objects-by-value etc. to give a location transparent, language and datastore -independent access to the objects.

OLEDB, which is the core of Microsoft's Universal Data Access strategy, defines a set of COM interfaces by which data providers, consumers and service components can interact with ease, for developing multitier enterprise applications. Applications or service components like query processor, cursor engine etc. can access the underlying diverse data in a unique way. The ActiveX Data Objects built on top of OLEDB gives a language and data provider -neutral, extensible and easy to use way for manipulating the data.Thus applications can use the same interface to access various heterogeneous datastores like mail stores, project management tools, ODBC databases etc. While OLEDB addresses persistency of COM objects and PSS that of CORBA objects, we are trying to integrate these two technologies to develop a location, language, operating system and datastore -independent, transparent way of persistent object storage.

5. Technology Integration: A Case Study

Our early TVR experiments [4] were based or JSDA collaboratory server and VRML multi-user front-ends, linked to the JSDA channels via the EAI or the Java scripting. The associated computational experiments within the WebFlow model were using Java Web server middleware and Java applet authoring front-ends.

We are currently building the new version of an Object Web based collaboratory WebFlow environment which includes and integrates several enabling technologies listed above. Our current application domain that drives the system design and prototyping is given by the WebHLA based military modeling and simulation where are are developing the Object Web based RTI and the WebFlow based visual authoring tools for HLA simulations. The software bus of our system is given by JWORB - Java Web Object Request Broker that integrates Java, Web and CORBA middleware technologies.

JWORB is a multi-protocol extensible server written in Java. The base server has HTTP and IIOP protocol support. It can serve documents as an HTTP Web Server and it handles the IIOP connections as an Object Request Broker. As an HTTP server, JWORB supports Servlet and CGI mechanism. Any servlet developed with Java Servlet API can run with JWORB.

Since JWORB design is Object Oriented, it is very easy to add other protocols. As JWORB starts up, it looks at configuration files to figure out which protocols it is capable of handling and it loads the necessary protocol classes for each protocol. If we want to add a new protocol, we need to implement a few abstract classes defined for the protocol object and to register this protocol implementation in the configuration file. This mechanism allowed us to define HTTP and IIOP protocols in the current prototype and we are investigating to include the DCOM Protocol or CORBA-to-DCOM bridge to be able to communicate with the DCOM objects.

On top of JWORB, we are developing dedicated services such as CORBA collaboratory.

5.1 Collaboration based on CORBA We include below the two most significant IDL definitions in our proposed CORBA Collaboration Service. The IDL definitions signify the operations a Client could invoke on a remote instance of these objects. Invocation of any of these aforementioned operations should be preceeded by a successful reception of a remote handle to these objects.

```
interface Coordinator {
   boolean setMaxClients(in long arg0);
   long getMaxClients();
   long numberOfMembers();
   typedef sequence string sequence_of_string;
   MultiCoordinator::Coordinator::sequence_of_string
                                  getClientNames();
   boolean isEmpty();
   long register(in long arg0, in string arg1,
                    in Client::ClientControl arg2);
   boolean deregister(in long arg0);
   boolean broadcast(in string arg0);
   boolean whisper(in string arg0, in long arg1);
};

interface PartyScheduler {
   boolean createParty(in string arg0);
   long getPartyID(in string arg0);
   MultiCoordinator::Coordinator
                  getPartyHandle(in long arg0);

};
```

The PartyScheduler is the one which schedules the appropriate instance of the Coordinator Object to coordinate Clients logged onto a specific session (Party) comprising of possible different applications. In the event that there is a Distributed Directory service and the Active Object server is in place, the Client is ready to invoke the IDL-defined operations.

- It starts with the createParty(String partyName) function which would return a true in the event that a new Coordinator Object has been instantiated or a false to signify the prior existence of the desired party.
- The Client gets a handle to the Coordinator Object by invoking long getPartyID(in string arg0); MultiCoordinator::Coordinator getPartyHandle(in long arg0); in succession.
- Once Steps I and II are over and done with, the Client is now in a Distributed Collaboration mode, and can invoke operations specified in IDL definitions for the Coordinator.

5.2 RMI-Based Collaboratory The abstractions provided by the CORBA Collaboratory are maintained in the RMI version, except that instead of IDL definitions as the starting point Java Interfaces perform the same function.

```
import java.rmi.*;

public interface PartyScheduler extends Remote {
  boolean createParty(String PartyName)
```

```
                throws java.rmi.RemoteException;;
  int getPartyID(String PartyName)
                throws java.rmi.RemoteException;;
  MultiCoordinator.Coordinator getPartyHandle(int partyID)
                throws java.rmi.RemoteException;;
}

package MultiCoordinator;
import java.rmi.*;
public interface Coordinator extends Remote {
  boolean setMaxClients(int _maxClients)
           throws java.rmi.RemoteException;
  int getMaxClients()throws java.rmi.RemoteException;
  int numberOfMembers()
           throws java.rmi.RemoteException;
  String[] getClientNames()
           throws java.rmi.RemoteException;
  boolean isEmpty() throws java.rmi.RemoteException;
  int register( int clientHashCode, String ClientName,
            String clientObjRef)
            throws java.rmi.RemoteException;
  boolean deregister(int clientID)
            throws java.rmi.RemoteException ;
  boolean broadcast(String Message)
            throws java.rmi.RemoteException;
  boolean whisper(String Message, int clientID)
            throws java.rmi.RemoteException;
}
```

The RMI based Collaboration is about 30-40% faster than the IIOP solution, however the advantage RMI holds is blunted by the fact that its a pure Java Solution. RMI-Collab is platform independent, albeit expressed through Java. Nevertheless, CORBA is a platform independent and language independent solution. With CORBA one could have Java Helper classes accessing a C++ implementation of the Party Scheduler. The choice is clear in case of pure Java solutions write to RMI else write to CORBA.

5.3. Logical Time and Data Distribution Management Communication locality is the most important concept which allows us to build large scale scalable collaborative environments. This essential feature is enabled via event filtering in terms of JSDA channels, CORBA Event Service and RTI routing spaces.

CORBA's Event Service tries to address the Subscribe/Publish pattern for information exchange between objects. By relying on this technology, it is possible that each participant keeps its Event Channel as its broadcasting channel in his/her machine and publishes its Channel address in the public directory so that all interested parties can get the address of this Event Channel and subscribe this channel.

In HLA/RTI[6], DDM[5] allows simulations to define a routing spaces so that the communication layer delivers the interaction and attribute updates to the appropriate simulation unlike broadcasting to everybody[7] and consuming computation time and network bandwidth[8] defined by Distributed Interactive Simulation(DIS)[9].

Time Management[10] service of RTI is being developed to provide appropriate time stamps for messages while allowing a simulation to pick the appropriate message order (FIFO, Priority Based, Total Order, and Causal Order). Handling messages in the correct order is a challenging problem since participants are distributed and network introduces latency. Because of this delays participants can receive messages which has happened in the past but arrived late[11]. The solution for this problem is to provide a synchronization mechanism between participants.

One of the approaches is to synchronize everybody conservatively. Conservative approach spends a lot of time for synchronization and wastes network bandwidth and computation resources. The second alternative known as Optimistic approach allows participants to proceed into the future(for a small time window); if they receive an event from past then they rollback. The last synchronized time is defined as a Global Virtual Time[12]. The determination of Global Virtual Time (GVT) consumes a network traffic and time; therefore we wish to calculate GVT as less frequently as possible[13]. To find the GVT, the minimum time stamped event(message) has to be determined while taking care of the events (messages) in transit. SPEEDES[14] handles this problem by counting the sent and received events globally in the system. Whenever it finds out they are equal, then it starts the process related to determination of GVT.

5.4 Visual Authoring Tools Front-end of our WebHLA prototype will include visual authoring tools for the HLA simulation objects. HLA Simulation Tools are being designed to provide automated support for development of HLA Object Models(OM), generation of RTI federation execution data and exchanging OMs with the Object Model Library. Currently available tools include Aegis Object Model Development Tool(OMDT),OSim from OriginalSim Inc

Currently, our efforts in this area include development of a Simulation tool that conforms to the OMDT look-and-feel and it is included as a module/component in the WebFlow framework. Also, experiments are being performed to explore the feasibility of exploiting useful components like Microsoft Excel'97 spreadsheet through its COM interfaces for the purpose of tree-like graphical representation of the simulation object model.

5.5 DirectX meets HLA The DirectX is a common infrastructure from Microsoft Inc., in the implementation of high performance real-time applications such as computer games and multimedia applications. The component of DirectX that supports multiplayer, networked applications is called DirectPlay. The functions provided by DirectPlay are similar to those provided by the HLA RTI with a few significant differences. DirectPlay provides some features specific to gaming. However, it does not provide any time synchronization features; it is built to support a DIS-like, loose causality model. DirectPlay also differs from the RTI in that it does not provide any support for determining data routing. In DirectPlay, there is no concept of a common object or data definition.

DirectPlay is possibly the more suitable technology for multiuser PC games. However combining DirectPlay with concepts of DIS and HLA might create a hybrid technology that could be useful for both military simulations and the entertainment industry.

We are currently designing the runtime display support using VRML and DirectX technologies, linked to the Java middleware via suitable interfaces (EAI, Java scripting, COM/CORBA bridge). We are also analyzing a set of advanced DoD simulation kernels including RTI, SPEEDES and ModSAF and we intend to experiment with their underlying time management algorithms within our JWORB framework via suitable CORBA interfaces. This way, the JWORB middleware acts as a universal software bus linking desktop/commodity visualization front-end with the DoD M&S backend.

6. Summary

We believe that the current suite of Web/Commodity technologies provides us with a critical mass sufficient to build the next generation world-wide scalable televirtual environments. We outlined here several enabling technologies that need to be integrated and we summarized the current status of our prototyping experiments. At the moment, our Java Web Server based WebFlow prototype is operational and we are also testing the early prototype of our new JWORB middleware. We are also experimenting with and evaluating the front-end display (VRML/Java3D/DirectX) and the back-end persistent store (JDBC/PSS/OLEDB) technologies as they emerge and evolve. In parallel with this core technology R&D, we are also exploring a suite of advanced military simulation environments such as SPEEDES, ModSAF, HLA/RTI. We expect early integration demos for multi-user WebHLA authoring and runtime to be available by summer '98 and to be presented during this Workshop.

References

[1] Craig Thompson, OMG/DARPA Workshop on Compositional Software Architectures, Monterey, CA January 6-8 1998, http://www.objs.com/workshops/ws9801/

[2] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski and G. Premchandran, "WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing", February '97, in the special issue of "Concurrency: Practice and Experience" on Java for Scientific Computing.

[3] G. Fox, W. Furmanski and T. Haupt, "ASCI WebFlow: High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing", http://www.npac.syr.edu/projects/asci-webflow

[4] D. Dias, G. Fox, W. Furmanski, V. Mehra, B. Natarajan, H. T. Ozdemir, S. Pallickara and Z. Ozdemir, "Exploring JSDA, CORBA and HLA based MuTech's for Scalable Televirtual (TVR) Environments", presented at VRML98, February '98, Monterey CA.

[5] Danny Cohen and Andreas Kemkes, Using DDM - an Application Perspective, 1997 Spring Simulation Interoperability Workshop (SIW), 97S-SIW-014

[6] High Level Architecture (HLA) by the Defence Modelling and Simulation Office (DMSO), at http://www.dmso.mil/hla/

[7] Katherine L. Morse, Interest Management in Large-Scale Distributed Simulations , UC Irvine, Information and Computer Science Technical Report, ICS-TR-96-27

[8] Duncan C. Miller, The DOD High Level Architecture and The Next Generation of DIS , March 1996, 14th DIS Wokshop, 96-14-115

[9] IEEE Standard for Distributed Interactive Simulation - Application Protocols, IEEE 1278.1-1995

[10] HLA Time Management Design Document, August 16, 1996, at http://www.dmso.mil/hla/tech/ifspec/

[11] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, Communications of the ACM, 21(3), pp:558-565, July 1978

[12] D.R. Jefferson, Virtual Time, ACM Transactions on Programming Languages and Systems, 7(3), pp:404-425, July 1985

[13] J. Steinman, C.A. Lee, L.F. Wilson and D.M. Nicol, Global Virtual Time and Distributed Synchronization,

[14] J. Steinman, 1991, SPEEDES: Synchronous Parallel Environment for Emulation and Discrete-Event Simulation, In the proceedings of the SCS Western Multiconference on Advances in Parallel and Distributed Simulation(PADS91), Vol. 23, No. 1, pp:95-103