

Specification, Control, and Analysis of Collaboration in Distributed Heterogeneous Interoperating Virtual Environments

David Stotts*

Department of Computer Science Collaboratory
University of North Carolina
CB 3175, Sitterson Hall, Chapel Hill, NC 27599-3175
stotts@cs.unc.edu

Michael Capps
MIT

ABSTRACT

Virtual environments (VEs) are currently built one at a time and no systematic approach exists for combining several existing VEs into a new interoperating VE. We have been researching interoperable VEs for the past two years and have produced a technique for interconnecting existing systems using the Polyolith software bus (J. Purtilo, University of Maryland). This proposal will extend out techniques into the area of multi-user collaborative VEs. We call the technology *Distributed Heterogeneous Interoperating Virtual Environments*, or DHIVE.

We will first produce a specification method for defining the rules of collaborative interaction in a DHIVE, and will produce analysis algorithms for verifying the correctness of these rules. We call these "collaboration protocols." To enforce the collaboration protocol in a DHIVE, we will modify the LambdaMOO system from Xerox PARC. The MOO currently is popular worldwide for multi-user text-based game playing, but we find the underlying OODB infrastructure to be useful for serious systems as well. We will modify the MOO to operate from an externally specified (and verified) collaboration protocol rather than from its hard-coded internal rules. We will also modify the MOO to operate as a component of a distributed system over the Polyolith toolbus.

The modified MOO will provide generic OODB capabilities to a DHIVE as well as controlling the collaboration of the many users in the DHIVE. We will be able to add object semantics to our early experiments in distributed VEs. The semantic problem is this: when an orange sphere is picked up in one VE and carried to another, what happens when the sphere is dropped? Does it bounce? Break? Thud to the floor? Float to the ceiling? The modified MOO and Java objects will allow the behavior of an object to be passed from one virtual world to another and still execute properly.

Our methodology will allow construction of new VEs from existing systems, distributed over a wide-area

network. For example, consider a single-user architectural walk-through VE built for visualizing new ship designs. A user dons a head-mount display and virtually navigates the interior space of a ship model. Consider also another VE to allow group conferencing for individuals physically remote from one another. The system allows them to have a virtual meeting, communicating visually and through audio streams. Why can't two such systems be "combined" to interoperate so that a new group walkthrough VE results? The group walkthrough VE would use the communications and floor control capabilities of the original conferencing VE and the navigation capabilities of the single-user walkthrough VE to provide a multi-user system. Several users could walk through a ship design "together" discussing their experiences as they go. The new system has aspects of each of its two building blocks, but is clearly not a simple modification of either.

After creation of the infrastructure for a DHIVE we will construct several example systems and experiment with them over the Internet between UNC and other universities.

Keywords: verification, formal methods, software reuse, reengineering, temporal logic, model checking, petri nets, collaboration protocols, groupware, virtual reality environments, distributed heterogeneous systems, interoperability

1 COLLABORATION SEMANTICS IN DHIVE

We propose to study the specification, control, and analysis of collaboration protocols in the context of distributed heterogeneous interoperating virtual environments (DHIVE). This work will build on our earlier experiments [7, 2] with creating interoperable VEs with the Polyolith toolbus [15], and will greatly expand the capabilities of VE technology specifically in the area of defining and managing multi-user interactions. We propose to extend DHIVE in several ways: addition of dynamically interpreted collaboration protocols (using lessons learned from Trellis research [18, ?, 10, 14]); incorporation of collaborative OODB technology [5]; and addition of a method for exchange of object behavioral semantics among individual VEs in DHIVE.

In the following sections we first describe our initial experiments in defining DHIVE with the Polyolith toolbus. After that we present the technical details of our proposed extensions for collaboration semantics, control and analysis. We close with a summary of our research goals and methods.

BACKGROUND: EARLY EXPERIMENTS WITH DHIVE

In earlier experiments we have studied how to interconnect existing graphical virtual environments so that they can interoperate. The goal was a software architecture whereby existing VEs could be used as building blocks for new ones. Consider, for example, a single-user architectural walk-through VE built for visualizing new ship designs. A user dons a head-mount display and virtually navigates the interior space of a ship model. Consider also a another VE to allow group conferencing for individuals physically remote from one another. The system allows them to have a virtual meeting, communicating visually and through audio streams. Why can't two such systems be "combined" to interoperate so that a new group walkthrough VE results? The group walkthrough VE would use the communications and floor control capabilities of the original conferencing VE and the navigation capabilities of the single-user walkthrough VE to provide a multi-user system. Several users could walk through a ship design "together" discussing

their experiences as they go. The new system has aspects of each of its two building blocks, but is clearly not a simple modification of either.

Interoperable VEs require the capabilities and resource needs of a VE to be specified abstractly, and for new VEs to be generated at least semi-automatically from these specs. Our early experiments did this with existing distributed program generation technology called Polyolith [15]. Using the Polyolith toolbus, we converted several existing single-user VE applications written previously at UNC into multi-user distributed systems [7, 2]. The interaction among users were very limited in these experiments, as we concentrated on the technical problems of sharing different data formats and execution environments across a heterogeneous network of machines.

Our initial experiments have been promising but there is much work left to do to get to a consistent methodology for building interoperable VEs. The DHIVE conceptual framework we are developing is an effective combination of several technologies:

- **Distributed:** DHIVE systems are composed of multiple processes executing on separate hosts and communicating across a network.
- **Heterogeneous:** The distributed processes in a DHIVE system may be implemented in different programming languages, compiled for different hardware platforms and different operating systems, and may even use different forms of interprocess communication (IPC). Our early experiments interconnected systems that has different user interface paradigms (combining 3D immersive in some processes and 2D WIMP in others).
- **Interoperating:** The ultimate goal of the DHIVE technology is to allow one DHIVE to exist and operate “within” or “among” others. An object in one VE will be able to be carried to another and retain its correct appearance and behavior. We have a start on this aspect from earlier experiments, but we propose to extend this aspect considerably in the current project.

Development costs could be reduced by using software engineering techniques to build reusable virtual components. Each virtual component may operate as a stand-alone virtual reality, but when interconnected with other components, it becomes part of a larger virtual environment (VE). Thus each component of the VE will need to have features that allow interconnection, while still maintaining hardware-specific software that gives high performance. As a result of this work, VE software engineers will have a tool that facilitates reuse of virtual components, and further, they will be able to interconnect components in novel and creative ways that heretofore would have required a completely new system.

- **Virtual Environment:** DHIVE allows the rapid construction of immersive, direct-manipulation graphical systems using existing systems as building blocks.

We are proposing to extend the basic DHIVE structure to encode and manage the semantics of multiple collaborating users. We are also proposing to extend the DHIVE to function more automatically than it currently does, and to encode and exchange objects among VEs so their behavior is consistent across VEs. First we present more details about our existing

Interconnecting VEs with Polyolith Figure 1 shows the components of the Polyolith bus and their relationship to two processes in a heterogeneous distributed VE. The communications interface between the two processes is generated by the bus from libraries of modules specific to different communication methods, hardware architectures, operating systems, etc. The spec files contain information about the

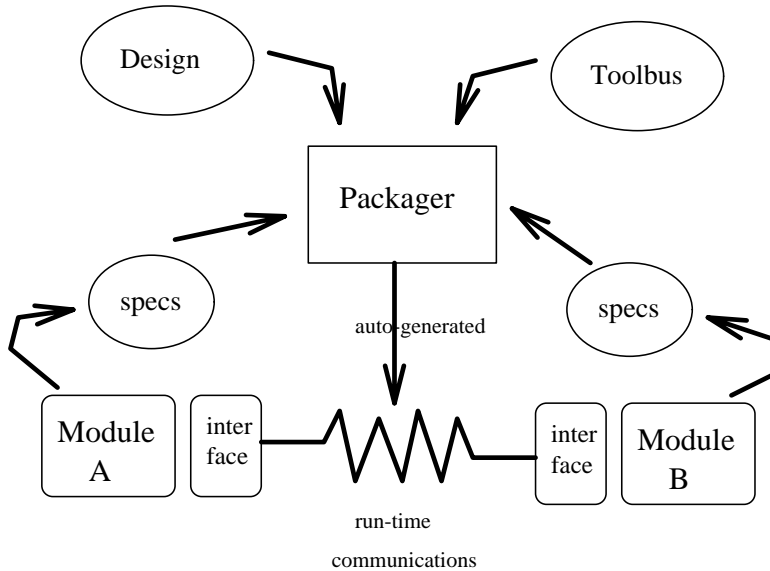


Figure 1: Polyolith VE Interconnection

abstract interfaces of the modules to be connected. The system design is a specification detailing the network of processes desired. The toolbus is a library of functions written to implement the details of various communication methods: sockets, TCP/IP, RPC, shared memory, etc.

We think this technology is especially well suited to interconnection of VEs, and that it provides benefits for engineering new VEs from existing ones. It requires development of appropriate specs files (see the figure) that capture the abstractions commonly found in different VEs. It also requires encoding somehow the various behaviors objects exhibit when manipulated in a VE. If I pass you a sphere from my VE into yours, what will happen when you drop it? Will it bounce, like a basketball? Or will it "thud" to the floor, like a bowling ball?

We now present summaries of three interconnection experiments we have done with Polyolith and VEs. We have learned enough about doing this to propose a more thorough exploration. There are more fully described in [7, 2].

Xfront-to-Xfront The first virtual environment application chosen for interconnection was Xfront, a joystick-navigated 3-D walkthrough with generic hi-resolution mono output and an X-interface. Xfront was chosen for the primary experiment for a host of reasons related to suitability and availability. First and foremost, the code is stable and in full release, and many members of the original programming team were still available at UNC for consultation. Second, it requires limited resources or experience to use, as it does not use the more complicated head-mounted display arrays and is intended to be a simple, easy-to-use walkthrough. Lastly, the code has a very tight event loop (approximately 15 lines) that made understanding the code structure, and modifying it, a simple task.

Our goal for the Xfront modification and interconnection was to examine the capabilities of the Polyolith bus system, so our modifications were chosen with simplicity of implementation in mind. The project decided upon was a shared-walkthrough in which two separate Xfront processes would navigate identical virtual worlds, and each user would be able to see the other user's position as marked by a stick-figure

object. Running more than two processes was assumed to be a trivial exercise, excepting only that the Xfront software requires a specific configuration which UNC can only provide twice simultaneously.

This model embodies a simple communication model in which each of the two Xfront processes has a data source and sink connected to the other process. For the sake of simplicity, an elementary blocking read was used.

The shared-walkthrough model necessitated only two changes to the original code: the transformation matrix of the user needed to be calculated and sent to the other process, and the other user's eye matrix was needed to move the stick-figure object correctly in the local process. We found the proper location and types for bus calls to be easy to identify, so the existing system was relatively easy to modify for distributed execution.

Xfront-to-Vixen The second VE program chosen for interconnection was Vixen, an immersive walkthrough application that makes use of a head-mounted display with head and hand tracking devices. Motion is handled by buttons on a 3-D mouse, rather than by joysticks or keyboard input. Regardless of the different equipment for both input and output, Vixen is built upon many of the same libraries as Xfront, so it is similar in function and file-formats but still satisfied our goal to connect heterogeneous VE applications.

The task to create a shared walkthrough with the two disparate programs was much less difficult than the previous effort. The modified version of Xfront required absolutely no modification for this project, which showed the value of using the Polyolith system for inter-process communication. Once a walkthrough has been modified with Polyolith message-handlers, it can be easily plugged through the orchestration files to connect to any other similarly-modified application. Since Vixen and Xfront share file formats and many libraries, the specialized object-moving and position-grabbing functions were able to be re-used. The major difficulty in sharing a world between different walkthroughs was scaling and relative positioning of objects in the world at startup; this was corrected by keeping standard object starting locations in a file available to both applications, and using the absolute object-move function to adjust the initial virtual world accordingly. Even with heterogeneous applications, and significant differences between their design, this project took only a fourth the time of the original dual-Xfront connection. A subsequent dual-Vixen version took only the time to change the name of a process in the Bus orchestration file; the interoperability of the communication method shone here.

Gvixen and MEMUD Our most recent experiment with the Polyolith system and interconnecting VEs was much more complex as it involved multiple walkthroughs and many heterogeneous non-VE processes. The project goal was to implement a Multi-Environment Multi-User Dungeon (MEMUD), in which many users share a dynamically-generated maze populated with robots. Each robot figure can be any of three types: a 2-D Walker, a 3-D Walker, or an Automaton; however, there is no way for any robot to tell the type of another robot.

The 3-D Walkers are actually users in the immersive Vixen program; a modified version called Gvixen reads a textual description from the bus and generates a 3-D maze complete with stone textured walls and dirt floors. All other robots, no matter the type, are represented as stick-figures moving about the corridors of the maze. The 2-D Walkers are users of an interactive UNIX X-windows display; cursor movements cause a small icon to move about the same maze, which is rendered from the text description but in 2-D graphics. Other robots are represented as unique icons, but no differentiation is made between the different robot flavors. The Automatons have the same display as 2-D Walkers, but their motions are generated by computing engines rather than by humans.

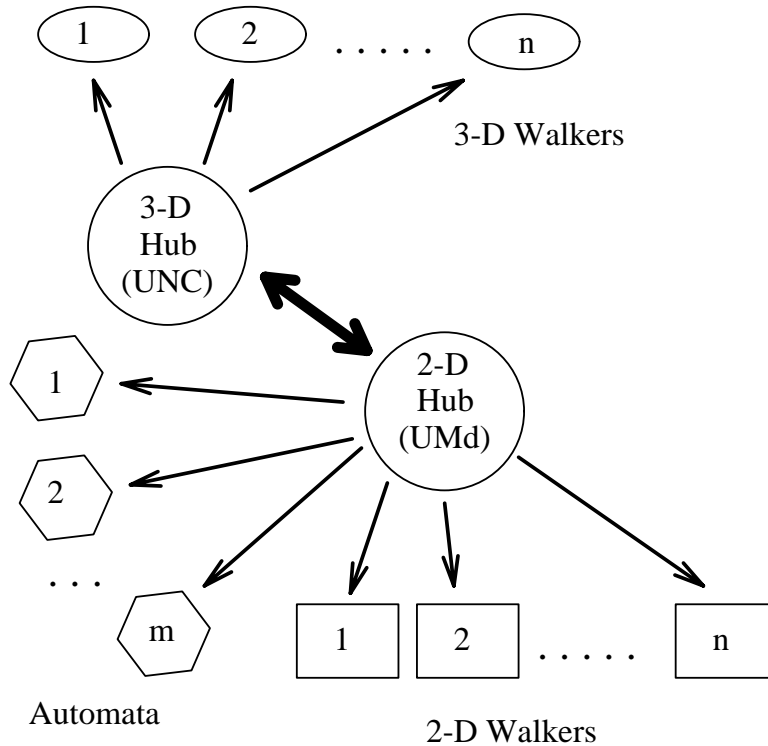


Figure 2: MEMUD DHIVE mixing two interface paradigms

This project was planned in order to make more use of the distributed capabilities of the Polyolith system, as well as to further cooperation between the Maryland and North Carolina sites. Therefore, the MEMUD system was developed with two main hubs, one at each of the universities. At UNC, the 3Dhub process accepts the text-descriptor of the maze and distributes it to the copies of Gvixen running locally (currently limited to 2 by availability of rendering hardware). It also collects position information for the 3-D Walkers and exchanges that information with 2Dhub running at Maryland. The Maryland hub holds the maze description and collects location information from the multiple 2-D Walkers and Automata. Therefore, all position information is communicated between universities on only one channel, which proved to be the more efficient method. The positions sent are only the map x,y coordinates, and each process translates the map coordinates into 3-D coordinates, X-display pixels, etc., as appropriate.

Figure 2 demonstrates the configuration of the different processes and shows the communications connections and patterns between them. The maze description propagates directly outward in all directions from process 2Dhub. The resulting system had some users at UNC immersed in a 3-D maze, some users at UMD sitting at X consoles navigating the maze in 2-D with the keyboard, and some maze walkers being controlled by background programs.

MEMUD is a truly distributed and scalable application that connects Virtual Environment walkthroughs with two-dimensional graphical interfaces. It demonstrates the flexibility of the Polyolith system, especially in that the GVixen program was ready to use only after the maze-generating modifications were made. The 3Dhub translates the GVixen user transformation into maze coordinates, so the generic communication already in Vixen was still usable. The scalability in number of processes is another benefit of Polyolith. Rather than needing additional programming, making the hubs scalable was a trivial task, and starting

additional instances of identical processes with the bus is exactly what Polyolith was designed to do.

Discussion and Relation to other Research A number of groups have been studying distributed virtual environments, and have built prototype environments and tools to help build those systems [12, 19, 16, 3]. Each of these systems depends on a homogeneous data and control environment to ease the VE development process. While they all provide a distributed system, they are each still stand-alone in the sense that they operate in a particular language and control domain. In the long term, it is unlikely that the VE community will settle on one particular language for control structures. Thus a system such as proposed here will be necessary, should a software engineer decide to reuse VEs built using different control paradigms.

One system we know of, BrickNet [16], is concerned with sharing object semantics in a distributed VE. They accomplish this with a custom-designed scripting language for executing object actions. These scripts are shipped around with the objects across the network. This will work for a homogeneous network, but fails to be scalable without a great deal of effort to a network of heterogeneous platforms. Our proposed approach using Java will accommodate the heterogeneity BrickNet cannot.

Finally, Funkhouser at UC Berkeley has been working on multi-user virtual environments [8, 9]. This work is primarily looking at networking issues of data delivery, performance guarantees, etc. Our work will concentrate on rules of collaboration and presume we have useful technology like Funkhouser's for good network performance.

In our previous experiments, the Polyolith system helped greatly in abstracting the communications modifications, but unfortunately progress was slowed because the design of Xfront and its libraries was not particularly conducive to interconnection. It was determined that availability of certain functions is likely quite necessary for the smooth interconnection of VE applications such as Xfront. For example, a function for determining the absolute location of the user's eyepoint is needed, as is a method for transforming an object to such an absolute location by replacing its matrices rather than incrementally multiplying. The doubly-connected Xfront project progressed more slowly than expected because it was necessary for our software engineers to become intimately familiar with the code and libraries, as well as the graphics theory, to add this functionality.

None of the other efforts in distributed VEs is working with heterogeneity, and none is examining control of collaboration with an externally-defined and analyzable group interaction protocol. Only BrickNet is dealing with object sharing. Our collaborative DHIVE work will center on these areas of investigation

DHIVE EXTENSIONS: COLLABORATION AND OBJECT SEMANTICS

While the previous experiments are promising, we are a long way from having achieved the goal of a systematic methodology for constructing and interconnection interoperable VEs. Our further research will identify a full methodology to help future VE libraries and systems be conducive to interconnection and interoperation.

The interconnections we have achieved so far have dealt primarily with syntactic issues: mismatched data formats, mismatched run-time structures, object appearances, etc. Though we created DHIVEs with multiple users in them, we have no provision at this time for managing the activities and interactions of these users. Essentially, each user sees the others, but cannot affect the behavior of the others directly. This would require semantic information about objects and about rules for collaboration.

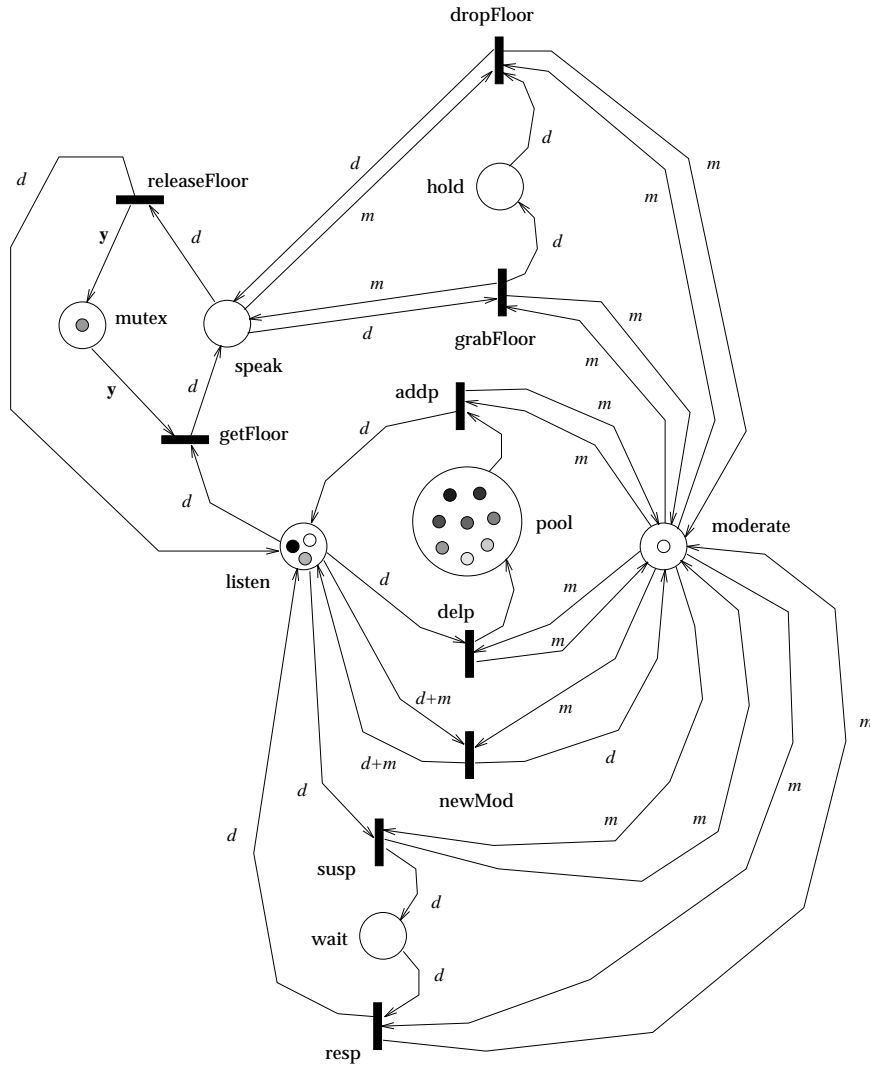


Figure 3: Trellis net for a simple meeting protocol.

Our proposal is to continue the DHIVE research in more depth, adding in specifically semantic issues of object behavior and multi-user collaboration control. While we remain open to pursuing whatever avenues look promising as the research progresses, we think the following are logical places to continue our study of collaborative DHIVES.

Collaboration protocols We will investigate use of what we call collaboration protocols to define and manage the interactions of multiple users in DHIVE. The technical issues to be examined include the specification notation to use and the mechanism for enacting and enforcing the rules across distributed VEs. In our current experiments, the users in a DHIVE are aware of each other but may not affect each other. There are no rules, for example, disallowing two users to occupy the same floor space, or no facilities for allowing two users to grab an object simultaneously.

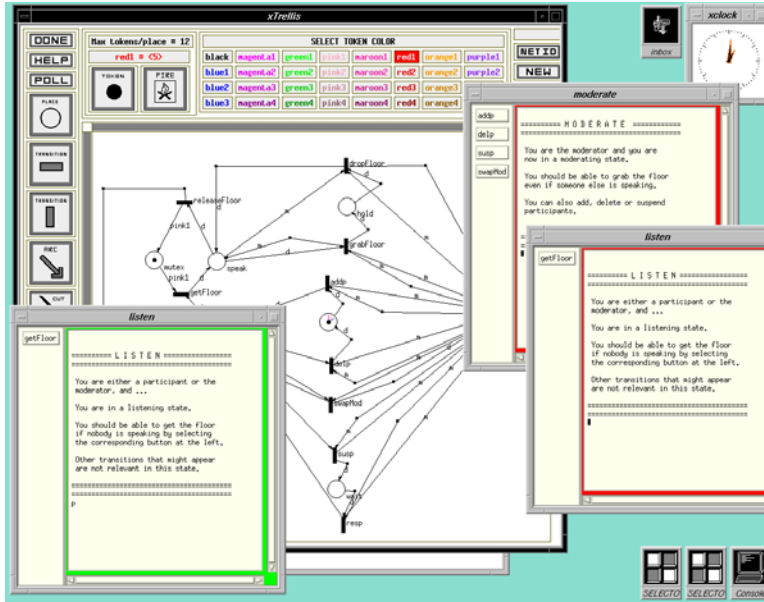


Figure 4: xTed net editor client for Trellis.

Figure 3 shows an example of a collaboration protocol expressed as a Trellis colored Petri net. It encodes the interaction rules for a simple moderated meeting. The moderator can add and remove people from the meeting, and one person at a time may get the floor to speak. The moderator may grab the floor temporarily from a speaker, and a participant may exchange roles and become the moderator. Figure 4 shows our current prototype displaying this hyperdocument as a simulation of a groupware system controlled by the meeting protocol.

We have built a group editor called Sui'tre to further see how a collaboration protocol could be externally specified yet still drive a collaborative software system [17]. In Trellis each protocol is a hyperdocument and is implemented as a server in a client-server system. Hence, it works as a generic protocol/collaboration server as well. In Sui'tre, low level coupling issues [6, 11] were driven from the internal code whereas high-level decisions – such as how many iterations to allow on a document – were derived from the Trellis protocol, which could be dynamically altered while a collaboration was in effect.

Verification via model checking We have developed a novel and effective way to analyze a simple form of Trellis document, one that uses normal Petri nets for structure [18, ?]. We employ model checking algorithms originally developed for verification of liveness and safety properties of concurrent computations [4]. The nets are converted into Kripke structures and are annotated with atomic factual information about the protocol. Then model checking is used to determine the truth of temporal logic formulae that express various properties of the protocol. For example, we can determine for a Trellis hypermedia document is there is any way a browser can get into a state where it is impossible to browse back to the table of contents; or we can ask if a document is structured such that all readers see a "warnings" before accessing a "copyrighted" page the first time, but not thereafter.

We propose to extend this analysis approach to the colored nets used to express collaboration protocols. We will be using newer model checking methods based on *binary decision diagrams* [1, 13] rather than Clarke's original methods (which were employed in Trellis). For the moderated meeting protocol show

previously, we might want to ask if it is possible for a speaker to get stuck on hold after having the floor grabbed by the moderator; or if more than one person can be speaking at a time; or if the moderator spot can ever be empty; or if a person desiring to speak will eventually be able to do so. The analysis methods we develop will allow these kinds of queries to be resolved by the designer of a collaborative DHIVE before traditional software testing begins.

Object Semantics Our early experiments in DHIVE have centered on syntactic issues mainly. We have established Polyolith methods for exchanging information among VE process that have different executable code, different data formats, and execute on different hardware/operating system platforms. However, the tougher problem remains to be solved: how does the dynamic behavior an object has in one VE get retained and faithfully exhibited in another VE? If I pass you a sphere from my VE into yours, what will happen when you drop it? Will it bounce, like a basketball? Or will it "thud" to the floor like a bowling ball? Perhaps it should not fall at all, but rather rise to the ceiling.

We think Java methods are a logical place to begin experimenting with interoperable object semantics. The interpreted nature of Java means that executing a single object behavior on one platform should give the same results as seen on others. The technical issues to examine are how to store and transmit Java methods for objects, and how to represent the collection of behaviors for an entire DHIVE (so it may be nested in another). Also, varying environmental aspects of VE's will have to be represented and have an impact on object semantics. For example, if the orange sphere has the behavior of "fall and bounce" in its home world, it will fall more slowly and bounce differently if taken to a VE that has a different gravitational constant... or no gravity at all. The value of gravity is a factor outside the actual object behavioral semantics, yet one that clearly will alter the perceived behavior of the object.

We will develop a generic VE object software architecture that will contain various attributes such as appearance, geometry, etc. and also various action attributes defining its behavioral semantics: sensors to gather required input about the environment (gravity levels, light levels, etc.) and reactors to define the actions an object will take on the environment.

We will use the Polyolith toolbus to exchange Java methods as needed when objects are transferred from one VE in a DHIVE to another.

A Modified MOO We intend to investigate the use of Xerox's LambdaMOO (or an appropriate Java equivalent, if one can be found) as a delivery vehicle for both interoperating object semantics and collaboration protocols in the extended DHIVE. The MOO will be modified in several ways:

- allow storage of Java methods along with the intrinsic MOO code
- operate on the Polyolith toolbus as one of many distributed processes in a DHIVE
- operate using group interaction rules obtained dynamically from a Trellis-like collaboration protocol rather than from (or perhaps in addition to) intrinsic MOO code

The LambdaMOO (and dozens of others MUD/MOOS) are in constant heavy use. Thousands of users daily log onto these collaborative simulations and spend hours each interacting. This is voluntary behavior, which we think speaks well of the usability and accessibility of the basic technology in the MOO. LambdaMOO is designed as a collaboration control structure, one that is extensible with new objects. We propose to leverage this successful technology by including it in a DHIVE as both an OODB and as a collaboration support structure. By modifying the MOO to use externally defined collaboration protocols

we will gain the ability to analyze the collaboration rules for correctness and also gain the ability to pass collaboration rules from one DHIVE to another.

The MOO (or perhaps several) in a DHIVE may function as a permanent repository for the behavior of objects, or it may only be needed as a temporary storage location for these methods during times of transition. We may use it to store other attributes of objects since the OODB in the MOO will be available over the toolbus to all VE processed in the DHIVE. These are issues to be resolved with experimentation.

SUMMARY OF RESEARCH GOALS AND METHODS

Goals We will extend our early work in DHIVE by more fully developing a formal methodology for constructing *Distributed Heterogeneous Interoperating Virtual Environments*. Specifically, we will be adding formal collaboration semantics and object behavioral semantics. The questions we propose to research in this project are:

- What abstract software architecture is best for construction on VEs so that they may be interconnected and interoperate?
- How can performance and appearance aspects of VEs be specified formally to make interconnection of VEs via Polyolith more automated?
- How can the semantics of objects in a DHIVE be preserved as objects are transferred from one VE into another?
- How can the interactions of multiple users be managed flexibly in a DHIVE? Can the interaction rules be expressed independently of the code of the individual interoperating VEs? Can the rules be modified dynamically and made to adapt to changing circumstances in a DHIVE?
- Can the group interaction rules be analyzed for correctness?
- Can adequate performance be maintained across a network to allow the DHIVE to implement realistic and useful collaborative systems?
- What parameters/aspects of individual VEs can be adjusted dynamically to maintain overall acceptable DHIVE performance?

Methods We propose to expand the basic DHIVE technology to enable provably correct, flexible multi-user collaborations, and to allow collaborative DHIVES to interoperate:

- We propose to develop notations for expressing group interactions in multi-user VEs; we call such expressions of group interactions *collaboration protocols*.
- We propose to develop analysis methods for determine the correctness of collaboration protocols. Like concurrent algorithms in general, they can, if written incorrectly, contain problems like deadlock. They can also fail to meet requirements related specifically to collaboration.
- We propose to further develop the Polyolith toolbus to support interconnection and interoperation of *collaborative* DHIVE. Specifically, we will alter the toolbus to use collaboration protocols as the "rules" governing the interactions of multiple users in a VE.

- We will alter a collaboration support environment (Xerox PARC's Lambda MOO) to function as a process on the Polyolith toolbus. The altered MOO will be structured to enforce our collaboration protocols rather than the "burned-in" interaction rules.
- We will develop and experiment with multi-user VEs built on this infrastructure. We expect to find our approach more flexible than current technology, furthering us towards the goal of interconnectable and interoperable collaborative VEs.

Evaluation To validate the techniques we develop we will perform experiments with virtual environments developed by persons in the UNC graphics lab outside the DHIVE research group. One Experiments will specifically be the scenario posed earlier:

Consider an architectural VE built for visualizing new ship designs; it allows an individual to construct a 3-D model of a new ship and to "walk through" the ship, experiencing the design from a ship-dweller's perspective. Consider also a second VE built to simulate a group conference; it allows a collection of individuals physically remote from one another to experience a face-to-face meeting in a virtual 3-D environment. Could we not combine these two systems (without the expense and time required to develop a separate third system) so they interoperate, allowing a group of physically separated people to virtually walk through a ship "together," communicating as a group?

We have two such VE applications at the UNC graphics lab, each written by independent research teams. We will test the utility of the collaborative DHIVE by constructing an interoperating system from these two separate VEs. We expect there will be others as well in the lab to combine, as we found in our early experiments.

We will also seek to interact with teams at other graphics labs. UNC participates in a 5-site graphics technology center with 4 other universities. We will solicit candidate VE systems from those labs and experiment with collaborative DHIVE for interconnecting them.

References

- [1] J. R. Bursh, E. M. Clark, and K. L. McMillan. Symbolic model checking: 10 to the 20 states and beyond. Carnegie Mellon and Stanford Universities, 1989.
- [2] M. Capps, D. Stotts, J. Duff, and J. Purtilo. Distributed interoperable virtual environments. In *Proc. of the Int'l Conf. on Configurable Distributed Systems*, pages 202–209, May 1996.
- [3] C. Carlsson and O. Hagsand. Dive – a platform for multi-user virtual environments. *Computers and Graphics*, pages 663–669, 1993.
- [4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
- [5] P. Curtis. Lambdamoo programmer's manual. Technical report, Xerox PARC, August 1993.

- [6] P. Dewan and R. Choudhary. Coupling the user interfaces of a multiuser program. *ACM Transactions on Computer Human Interaction*, 2(1):1–39, March 1995.
- [7] J. Duff, J. Purtilo, M. Capps, and D. Stotts. Software engineering of distributed simulation environments. In *Proc. of WETICE '96*, pages 262–267, June 1996.
- [8] T. A. Funkhouser. Ring: A client-server system for multi-user virtual environments. In *Computer Graphics (1995 SIGGRAPH Symposium on Interactive 3D Graphics)*, pages 85–92, April 1995.
- [9] T. A. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proc. of IEEE VRAIS '96*. IEEE, April 1996.
- [10] R. Furuta and P. D. Stotts. Dynamic hyperdocuments: Replacing the programming metaphor. *Communications of the ACM, special issue on Hypermedia Design*, pages 111–112, August 1995.
- [11] R. Hill, T. Brinck, S. Rohall, J. Patterson, and W. Wilner. The rendezvous architecture and language for constructing multiuser applications. *ACM Transactions on Computer Human Interaction*, 1(2), June 1994.
- [12] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. In *Proc. of VRAIS '95*, pages 2–10, 1995.
- [13] K. L. McMillan and J. C. Schwalbe. Formal verification of the gigamax cache consistency protocol. Carnegie Mellon and Encore Computer, 1992.
- [14] J. Navon, D. Stotts, and R. Furuta. Subdocument invocation modes in collaborative hyperdocuments. *Computers in Industry*, 29:91–104, 1996.
- [15] J. Purtilo. The polyolith software bus. *ACM Trans. of Programming Languages and Systems*, pages 151–174, January 1994.
- [16] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng. Bricknet: Sharing object behaviors on the net. In *Proc. of VRAIS '95*, pages 19–25, 1995.
- [17] D. Stotts, P. Dewan, J. Navon, and J. Munson. A three-level binding for collaborative editing semantics. In Roy Rada, editor, *Groupware and Authoring*, pages 297–324. Kluwer, 1996.
- [18] P. D. Stotts, R. Furuta, and J. C. Ruiz. Hyperdocuments as automata: Trace-based browsing property verification. In *Proceedings of the 1992 European Conference on Hypertext (ECHT92: November 30–December 4, Milan, Italy)*, pages 272–281. ACM Press, New York, 1992.
- [19] Q. Wang, M. Green, and C. Shaw. EM—an environment manager for building networked virtual environments. In *Proc. of VRAIS '95*, pages 11–18, 1995.