Software Engineering of Virtual Environments: Integration and Interconnection

Donald J. Welch Jr.
`dwelch@cs.umd.edu`
Department of Computer Science
University of Maryland *

James M. Purtilo †
`purtilo@cs.umd.edu`
Institute for Advanced Computer Studies and
Department of Computer Science
University of Maryland

July 12, 1996

ABSTRACT

Virtual Environments(VEs) are proving to be valuable resources in many fields, and they are even more useful when they involve multiple users in distributed environments. Many useful VEs were designed to be stand-alone applications, without consideration for integrating them into a distributed VE. Our approach to connecting VEs is to define an abstract model for the interconnection, use integration tools to do as much of the work automatically as possible, and use a run-time environment to support the interconnection. With our experiences to date, we are learning that certain classes of techniques are common to all solutions using this approach. We have summarized these in a set of requirements and are building a system that features these techniques as first class objects. In the future you will be able to solve these interconnection problems cheaply, plus engineers of future VEs will have some guidance on how they should organize their implementations so that interconnection with other VEs will be easier. In this paper we coin the phrase *software engineering of virtual environments* (SEVE) to describe the above activities.

INTRODUCTION

Virtual Environments (which include virtual reality, simulation, and telepresence) are being applied successfully to many different applications. As we found with other computer applications, VEs are even more powerful when they cooperate over networks.

Currently distributed multi-user VEs, can be built from scratch or from existing VEs by manually integrating them into a distributed VE. A system developed from the start as a distributed VE will have many advantages, and there are a number of software engineering tools to make this task easier. However, the cost of developing new code is high even with software engineering support and may not always provide low-enough cost products. There are real advantages to re-using existing code. However, due to the complexity of VEs the process of manually integrating legacy systems is laborious and ripe with opportunities for error. The resulting system is still not guaranteed to be any easier to integrate the next time.

Many of the VEs currently in use were built without any consideration to integrating them with other systems into a distributed VE. Design decisions for stand-alone VEs were made to optimize that one system; as a result VEs can have very different architectures.

If distributed VEs are going to become widespread, then SEVE must make it easier to build them and reuse existing code. Universal standards and interpreted languages may solve this problem once and for all in the future, but there are too many existing systems to write them all off just yet.

Most of the research on building distributed multi-user VEs concentrates on designing the dis-

*Lieutenant Colonel Welch is in the U.S. Army and is studying at UMCP through the Army's advanced civil schooling program.

†This research was funded by Office of Naval Research contract number N000149410320

tributed VEs from scratch. The EM toolkit and DIVE environment are two of the best known systems. They provide an environment for building the entire VE to include interconnection.[WGS95] [Gre94] Virtual Design is another complete VE software engineering environment that also focuses on using a wide array of input data and hardware.[AFM93] DoD's Distributed Interactive Simulation (DIS) defines a standard for how VEs interact, but not how to meet that standard. There are tools developed to help developers meet that standard, but not to retro-fit existing VEs.[Com94] BrickNet uses an interpreted language to facilitate sharing object behaviors between VEs.[SSP+95] VRML is also an interpreted language that as of version 1.0 does not support connected VEs. There has been little research on how to retrofit legacy systems into distributed VEs.[DPCS96]

REQUIREMENTS FOR VE INTEGRATION

In this section we lay out the functional requirements for SEVE. The non-functional requirements have been discussed elsewhere [DPCS96] and are beyond the scope of this paper.

We use a running example to illustrate the requirements for a SEVE environment. Our example takes two existing simulators and integrates them into a distributed simulation to give the soldiers being trained the challenge of fighting a human opponent, vice a simulated opponent. The first is a flight simulator for an Apache attack helicopter. The simulator has two users, a pilot and gunner and is a stand-alone system. Most of the C code is devoted to simulating the flight dynamics of the Apache and controlling the displays in the simulator. The second simulator is a Bradley Infantry Fighting Vehicle simulator (BIFV). It is designed to work with other BIFV simulators so that crews can train in the simulators as a group. The ground soldiers that ride in the back have head mounted displays and can move outside the BIFV as they can in real life. The Ada code controls the shared virtual environment of the BIFVs as well as the virtual reality in the simulation. They each have their own events, data model, etc. The Apache has behaviors of interest to the BIFVs scattered throughout its code. This integration is tedious and fraught with risk.

Maintain the Object Hierarchy/Groupings When information is passed between VEs, the run-time environment must keep track of the objects that the information pertains to. In our example, such an association might be between soldiers and vehicles: when the squad is riding in a BIFV the members move as the BIFV moves, and when they dismount their locations must be tracked separately from the BIFV.

Translate the Reference Frames In a VE there may be multiple reference frames, e.g. object, observer, or world. Applications may use entirely different coordinate systems. There are numerous world coordinate systems to choose from when using geographic coordinates. In addition, coordinates are commonly specified in meters or some other physical measure. The translation must be invisible to the individual VEs.

When the Apache launches a Hellfire missile, the Apache tracks the missile's flight using its own reference frame and coordinate system. To render the missile the BIFV requires information in its world reference frame. The BIFV simulator must not have to know about the Apache's reference frame and coordinate system to render the missile.

Abstract the Communication VEs can have varied communication requirements and the engineer must be shielded from the implementation details. In most cases fast transmission time is key, but sometimes reliable communication is required. As an example, when a BIFV rotates its turret, that change can be sent by a fast-unreliable message. If the message is lost, then the next update will bring the system to the correct state. However, when the Apache joins the simulation, the passing of initial states must be done reliably, but speed is not as important. The type of message (light-weight object, network pointer, heavy-weight or real-time stream) as well as the recipients of the message (multi-casting) must be accessible to the programmer as a simple abstraction. [BMZ95] [MZP+95]

If VEs are only to receive pertinent messages, the run-time environment must have the logic to decide which entities get which messages. In a simulation such as this example, multi-casting is not critical, but if there were a brigade's worth of BIFVs instead of a platoon the Apache would not have to know about the turret direction of every BIFV, only those within 10 kilometers of it. The multi-cast groups the Apache must monitor change as the Apache moves about the simulated battlefield. The logic to handle these changes is best handled in the run-time environment and not the existing code.

Provide Concurrency Control Objects that are shared by multiple VEs must be protected to insure consistency. When a Hellfire missile is approaching a BIFV that is taking evasive action, without concurrency control the last action (maneuver or hit) would be the resulting state in the system. This is the opposite behavior from what we would like.

The concurrency control must have fine enough granularity to protect different aspects of an object such as degrees of freedom. This allows cooperative behavior to exist in the distributed system. When the soldiers dismount in our scenario, two may push on an obstacle at the same time. Concurrency control must insure that the obstacle reacts realistically.

Collision detection between objects from different VEs must also be handled in the run-time environment. Not only should the run-time be able to stop two objects from occupying the same virtual space, but it should also stop objects from passing through each other due to unequal or inadequate update rates. Rounds fired from the BIFV's 20mm cannon will cover large distances between updates. The system cannot let the Apache avoid hits because at t_0 the round was 15 meters short of the Apache and at t_1 it was 15 meters beyond it.

Translate Behaviors The behaviors that objects exhibit and constraints imposed on those objects can be very complex. The behaviors and constraints that are needed in a connected VE should not have to be completely rewritten in that VE. The code to simulate the behavior of the BIFVs smoke grenades is very complex and would not previously exist in the Apache simulator. Such characteristics as the location, size and density are too complex to be passed as a message, so the behavior will have to be coded into the Apache simulator and just the parameters passed during the simulation. Since the two simulators use different programming languages and graphics packages, translating the behavior is not just cut and paste. The integration environment must give the software engineer as much help as is possible with this task.

Direct translation of object methods such as CORBA architectures provide is not always appropriate. In our smoke grenade example, the BIFVs would need accurate (and computationally intensive) simulation of the smoke screen for a good training environment. In the Apache simulation a much rougher approximation would be appropriate, so as to not take the cpu cycles needed for the flight dynamics.

Translate Data Distribution Models There are multiple data distribution models that can be used in distributed VEs. (replicated, centralized, and distributed) The run-time environment must make it possible for a VE to interact with other VEs using the model with which it was designed. Modifying the VE to accept a mandated data distribution model requires too much programming, spread throughout the application. In this example the BIFV used a centralized database (because it was designed to have multiple copies of BIFV simulators) and the Apache had its own database. The run-time environment provides interfaces that allow the original VEs to interact with data as they were originally designed while insuring that the databases stay consistent.

Provide Time Mapping If VEs are to cooperate they must be on a similar time scale, however some VEs use *flying*, or *teleporting* for long movements. If these techniques are not appropriate for all the VEs in the system, then the run-time environment will have to handle the reconciliation. When ground soldiers *fly* in their VE, that would not look realistic to the Apache crew. The run-time environment must present a realistic portrayal of the soldier's movement and keep the soldiers from getting ahead of the rest of the simulation. The run-time environment will also have to handle time-stamps, to support predictive behavior and concurrency control.

Provide Event Mapping Each VE in the distributed VE has its own event set. Some of those events will correlate with events in other entities, and must be shared throughout the system. The run-time environment must provide a translation

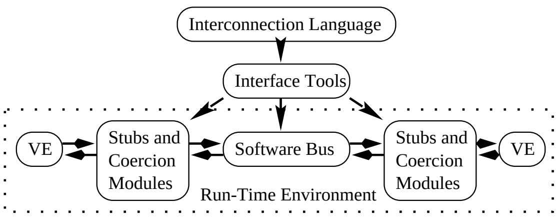

Figure 1: SEVE Architecture

and broadcast service for those events.

As VEs are integrated, some events from a standalone VE may no longer exist in the distributed VE. The run-time environment must automatically generate these events for the system. If the Apache simulation had a simulated Scout helicopter that helped guide it to the targets, the Scout would no longer provide accurate information unless it was reprogrammed to actually know where the BIFVs were. The messages that the Apache is expecting from the Scout must be taken care of by the run-time environment.

Finally the run-time environment must ensure the correct ordering of events when that is critical. The above example using round impact events and maneuver events is one example.

TOOL SUPPORT FOR SEVE

We are currently working to provide tool support for SEVE including a methodology, interconnection language, integration toolkit and run-time environment to support developing distributed VEs from existing VEs. Our solution is based on the Polylith software bus interconnection abstraction, a number of packagers and an interconnection language that allows the software engineer to describe VE interaction in abstract terms.

Interconnection Language (IL) This is where the interaction of the VEs is defined in a high-level manner. Using the interconnection language we express the geometry of the distributed simulation. This includes the communication paths and types of messages that the BIFVs and Apache will exchange. In addition we define the rules for translating between reference frames and coordinate systems. We use this language to define the objects that will be shared by the VEs. This includes rules on modifying the objects, translating their behaviors and how they are stored. To illustrate, the concept that the turret is part of the object hierarchy of a BIFV, while the infantry are grouped with the BIFV only while riding in the back is expressed in the IL. Shared events are also delineated here along with time mapping and any ordering constraints. By expressing how the VEs interact in an abstract form and by applying automatic techniques we allow the software engineer to focus more on making the distributed VE realistic and effective; less on just getting it to work.

Software Bus The software bus provides an interconnection abstract run-time environment that shields the VE from the communication and data coercion details. Network communication and language coercion code is left out of the BIFV and Apache simulations, it all exists in the bus. The VEs are modified to interface with the bus through stubs and not each other. The bus allows multiple types of communication, multi-casting and dynamic reconfiguration. The bus contains the logic to control multi-cast group membership and it also handles message and synchronization. This abstraction that has proved so effective for general program interconnection also makes VE interconnection much easier.[Pur94]

Object Base The object base is part of the software bus and therefore the run-time environment. It is used to keep track of the object hierarchies and groupings of objects so that each entity in the system can deal with its own groupings. It also handles concurrency control for the objects down to the attributes level. It presents each VE in the system with the data distribution model that it was originally designed to use.

The Apache in our example expects all information to be available in its database. When a BIFV moves it sends that information to its centralized database only, the bus takes that information and updates the Apache database. Each VE still interacts with the data the way it was originally designed. The performance advantages of the various data models might be lost with this arrangement, but the goal is to allow the VEs to cooperate by re-using as much code as possible.

Packagers The packagers are used to help set up the interface environment for the programmer. The packager uses the interconnection language to generate source code stubs that make it easier for the software engineer to connect to the bus. A packager also creates data coercion modules to translate data between reference frames, and data formats so that they can be used directly by each entity in the VE. In our running example once we define the translations between the reference frames the packager sets up the coercion module for the run-time environment. No translation code is added to the original VEs. The behavior packager sets up the framework for translating the smoke grenade behavior to the Apache VE.

Event Mapping Tool This is a software tool that automatically generates the software modules that implement the interaction defined in the IL between existing VEs. The event interaction between the modules is described in the IL. The software engineer then uses the event mapping tool to create coercion modules that combined with the software bus and VE code make up the execution environment. [CP95] For example, the *change location* event in the Apache simulation will map to the BIFV *update display* event.

Methodology The methodology includes the way to approach the analysis of legacy VEs prior to writing the IL modules, as well as the decisions of what to incorporate in the IL modules. We do not expect the methodology to be finalized until the rest of the system is complete and we have more experience integrating VEs.

CONCLUSIONS

We have performed some preliminary experiments connecting 3D and 2D virtual reality walkthroughs over the Internet and have been successful. To do this we have used a packager in conjunction with the Polylith software bus to connect the VEs. Our experience shows that modern software engineering techniques are a great help in integrating VEs. We were able to isolate all the interconnection to a single module that was easy to modify as we prototyped designs and then again as we added more users. The programming language used in the VEs was transparent, as the bus took care of any necessary data representation translations. Finally, changing the location of the executable code was trivial requiring no changes to the source code as the project evolved from multiple workstations in one department to VEs at two campuses communicating across the Internet.

If distributed VEs are going to be available for widespread use, then the cost of developing them will have to come down. Not all distributed multiuser VEs can be built from scratch, some will have to be built by reusing software. Re-using VE software can benefit from software engineering environments developed specifically for this purpose. A software bus abstraction combined with packaging technology and an abstract model can make the integration of existing VEs easier and therefore less costly.

References

[AFM93] Peter Astheimer, Wolfgang Felger, and Stefan Müller. Virtual Design: A generic VR System for Industrial Applications. *Computers and Graphics*, 17(6):671–677, November/December 1993.

[BMZ95] Donald P. Brutzman, Micheal Macedonia, and Micheal Zyda. Internetwork Infrastructure Requirements for Virtual Environments. In *Proceedings, VRML Symposium*, San Diego, CA, December 13-15 1995.

[Bro95] Wolfgang Broll. Interacting in Distributed Collaborative Virtual Environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 148–155, Research Triangle Park, NC, March 11-15 1995.

[Com94] DIS Steering Committee. The dis vision, a map to the future of distributed simulation, May 1994. Version 1.

[CP95] Chen Chen and James Purtilo. Event Adaptation for Integrating Distributed Applications. In *Proceedings of the Conference on Software Engineering and Knowledge Engineering*, June 1995.

[DPCS96] James W. Duff, James Purtilo, Michael Capps, and David Stotts. Software engineering of distributed simulation environments. In *Proceedings of the Conference on Configurable Distributed Systems*, pages 202–209, Annapolis, Maryland, May 6-8 1996. IEEE Computer Society Technical Committee on Distributed Processing.

[Gre94] Mark Green. Environment Manager. Technical report, Department of Computing Science, University of Alberta, February 1994.

[Gre96] Mark Green. Shared Virtual Environments: The Implications for Tool Builders. *Computers and Graphics*, 20(2):185–189, March/April 1996.

[MZP+95] Michael R. Macedonia, Michael J. Zyda, David R. Pratt, Donald P. Brutzman, and Paul T. Barham. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 2–10, Research Triangle Park, NC, March 11-15 1995.

[Pur94] James M. Purtilo. The POLYLITH Software Bus. *ACM Transactions on Programming Languages*, 16:151–174, January 1994.

[SSP+95] Gurminder Singh, Luis Serra, Willie Png, Audrey Wong, and Hern Ng. BrickNet: Sharing Object Behavoirs on the Net. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 19–25, Research Triangle Park, NC, March 11-15 1995.

[Vin95] John Vince. *Virtual Reality Systems*. Addison-Wesley, Wokingham, England, 1995.

[WGS95] Qunjie Wang, Mark Green, and Chris Shaw. EM - An Environment Manager For Building Networked Virtual Environments. In *Proceedings of the IEEE Virtual Reality Annual International Symposium*, pages 11–18, Research Triangle Park, NC, March 11-15 1995.