

BAMBOO - SUPPORTING DYNAMIC PROTOCOLS FOR VIRTUAL ENVIRONMENTS

Kent Watsen & Mike Zyda
NPSNET Research Group
Naval Postgraduate School
Monterey, CA.

Abstract

Distributed virtual environments enable interaction between participants using networking protocols. Historically, for lack of better methods, a single, all encompassing, highly enumerated protocol would be used. However, all entity expressions would be limited to just those having enumerations. Some more recent protocols enable dynamically enumerated objects using remote method invocation. Unfortunately, remote method invocation increases network traffic while delaying latency-critical interactions. This paper introduces another approach whereby the system dynamically downloads and installs client-specific networking protocols at runtime. This approach incurs longer object initialization while providing optimal runtime performance as the protocol can be specialized for the specific needs of that object. The reason that this approach has not yet been realized, until now, is due to the lack of a VE toolkit, like Bamboo, capable of supporting dynamic extensibility. This paper details how Bamboo enables per-object network protocols for distributed virtual environments.

Introduction

The development of distributed systems has been of active academic and commercial interest since the advent of networking technology. These systems traditionally implemented cost-effective solutions that maximized the utilization of valuable resources. More recently, these systems have been used to distribute computationally intensive calculations across machines hosted on a network. Another class of distributed systems enable the interaction between geographically remote participants within a shared, three-dimensional space. This last class implements what is commonly referred to as a distributed virtual environment (VE).

The development of distributed VE applications has been a primary focus of many research institutions. Recognizing this trend, some groups have developed toolkits that help implement specific applications by providing features common to those types of

applications. However, these toolkits tend to be monolithic architectures, limited in capability, difficult to extend, and/or available on only a few platforms, if not just one. The most significant of such toolkits include Alice (Deline, 1993), AVIARY (Snowdon, 1994), BrickNet (Singh, 1994), DIVE (Carlsson, 1993), dVS (Division, 1998), EasyScene (Coryphaeus, 1998), MASSIVE (Greenhalgh, 1995), MR Toolkit (Shaw, 1993), NPSNET (Macedonia, 1994), Vega (Paradigm, 1998), VEOS (Bricken, 1994), and World Toolkit (Sense8, 1998).

Another general failing of distributed VEs is that the networking solutions tend to lack flexibility. Historically, for lack of better methods, a single, all encompassing, highly enumerated, networking protocol, such as DIS (IEEE, 1993), would be utilized. However, the protocol's capabilities would be quickly exceeded as the individual participants desire to more fully express themselves (Singhal). Naturally, the missing features would be appended to the next version of the protocol, yet there would inevitably be the need for additional features. This test-and-patch approach is visible by the sequence of DIS protocols released over its six years of development. More recently has there been the development of protocols, such as CORBA (Ben-Natan, 1995), that enable dynamically enumerated objects using remote method invocation. Unfortunately, this approach tends to increase the amount of network traffic while delaying latency-critical interactions.

Another approach is for the system to dynamically download and install client-specific networking protocols at runtime. Although this approach is mentioned in the VRTP white paper (Brutzman, 1997), it has not yet been previously implemented, at least not in the virtual environment community. One of the more interesting aspects of this approach is the ability to have more than one protocol active at a time. In particular, rather than have one protocol for N entities, there could be N protocols, one for each entity. This suggests a fundamentally different solution for data distribution as it is no longer the case that a single, monolithic protocol must handle all the networking. Not only does this afford greater flexibility for the programmer, but it also implements optimal delivery mechanisms for specific data sets.

Bamboo (Watsen, 1998) is the result of years of trying to develop an adequate toolkit for the research and development of distributed VEs. It achieves this goal by understanding key issues and providing direct support for them, applying lessons learned from previous efforts towards an efficient implementation. These solutions are provided in the form of practical mechanisms implemented using object oriented and generic programming techniques. In particular, Bamboo is a virtual environment toolkit focused on the ability for the system to dynamically configure itself without explicit user interaction, enabling the system to take on new functionality after execution.

Enabling Dynamic Extensibility

All programming efforts, including distributed VEs, benefit from good software engineering design and development practices. These techniques attempt to facilitate low coupling, high cohesion, and code reuse. The ability to delay programming decisions until late in the engineering design cycle also provides greater flexibility. This benefit is especially noticeable when such decisions can be made after execution. Systems that do enable such decisions to be made after execution are said to be dynamically configurable. These systems facilitate orthogonal decomposition, save memory, reduce swapping, save disk space, while simplify modifications (Ho, 1991).

As successful as dynamically configurability is, it has not previously been applied to distributed VEs. A likely explanation for this is that its most effective utilization requires consideration early in the design cycle, thus requiring a greater amount of foresight. However, dynamic extensibility has been Bamboo's single most influential design decision. So much so that all of Bamboo itself is comprised of many modules, to the extent that the original executable, the core (see Figure 1), has only just enough logic to page modules and provide the initial framework for the plug-ins to hook into. In this way, no assumptions are made regarding what capabilities are needed by the kernel, but are determined at runtime by the application being loaded. For instance, if the particular application does not need a device-specific driver, the system would not load that module and thus save the memory and processing time that would ordinarily be consumed by such a mechanism.

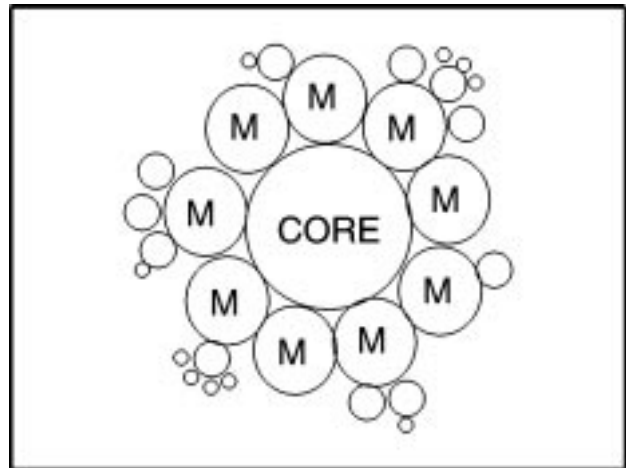


Fig. 1 Abstract plug-in view.

However, having each application specify every module it depends on could be a complex and error-prone process. Fortunately, each module, when being loaded, only needs to verify that its immediate dependencies are already in memory, loading them if not. For example, using Figure 2 as a reference, assume that module M4 is to be loaded into the system. The system must first verify that M3 has already been loaded into memory. Assuming that M3 has not yet been loaded, the system must then verify that both M1 and M2 have been loaded into memory. Assuming that neither M1 nor M2 have yet been loaded into memory, the system may go ahead and do so as they do not have any dependencies. Once M1 and M2 are loaded, M3 can load, and finally can M4. This sequence illustrates how a module (M4) need only specify its immediate dependants (M3). Now assume that M5 is to be loaded. Since M3 is now already in memory, the system may load M5 immediately. This example shows how shared dependencies do not require multiple loads. Naturally, the unloading of modules proceeds in the reverse order while reference counts identify shared dependencies.

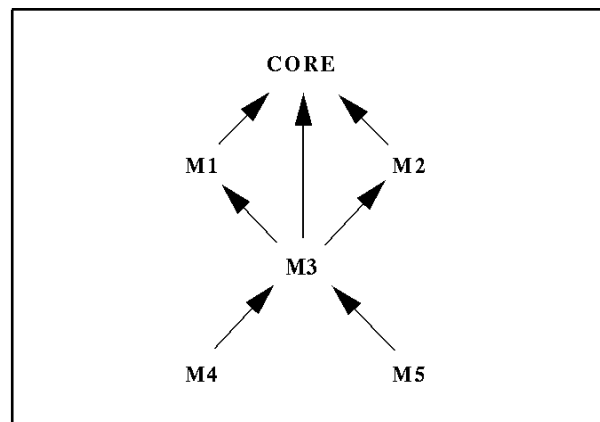


Fig. 2 Module dependency view.

Because it is desirable to have a module loaded off the network, if not found locally, its integrity may be suspect. This concern will be mitigated by Bamboo's insistence that a trusted partner sign all modules being loaded off the network. If the module does not have a trusted signature, the system prompts to have the signature added, to just load the module, or to ignore the module altogether. Modules may be multiply signed, thus enabling a hierarchy of trusted partners. For instance, the author may sign the module and the author's company may sign the module. Few may trust the author directly, but many may trust the author's company. At the top of this hierarchy is Bamboo itself; a module signed by Bamboo is implicitly trusted by all. It is recognized that this scheme can not be trusted to secure global-wide simulations, but does provide reasonable trust for academic and commercial research institutions.

Enabling dynamic extensibility requires more than simply bringing new code into the same address space as the current executable. In order for the module to perform work, as most modules do, it must either attach itself to an existing thread or create its own. Therefore, Bamboo provides a framework that the new code can explicitly attach itself to. Although this framework supports the management of lightweight threads, it is still possible for the newly loaded module to create its own. Therefore, this framework also has built-in synchronization primitives for thread safe access of global methods and/or variables. Following is a brief introduction to the framework itself.

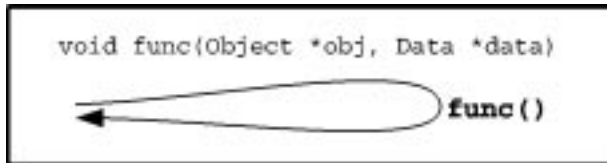


Fig. 3 A simple callback.

The callback class implements one of the most fundamental mechanisms in all of Bamboo. In its simplest form, depicted in Figure 3, a callback abstracts the execution of a single function having a specific declaration. In particular, the callback abstraction passes one reference to the invoking object and another to some user-specified callback data. As trivial as it may seem, the callback will be seen to be the backbone of Bamboo's architecture.

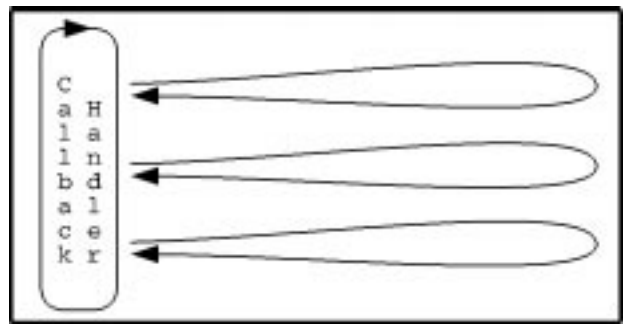


Fig. 4 The callback handler.

The callback handler class, depicted in Figure 4, collaborates with the callback class. It provides an interface that callbacks can add or remove themselves from. Callback handlers have the property of sequentially executing each of its callbacks in order when it itself is executed. The callback handler may be executed explicitly every time its subroutine is executed, or in response to an event, such as a timer interrupt.

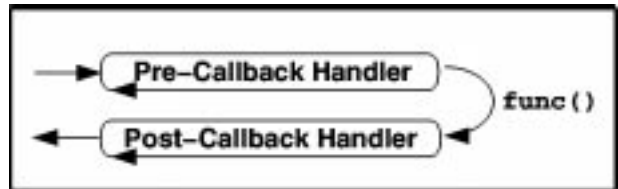


Fig. 5 All callbacks are recursive.

Of course, the callback handler itself would be a limiting solution if left in this form. Ensuring that the system can support robust behavior, each callback is actually recursive in that it embeds two callback handlers (see Figure 5), one just before and one just after the callback function is executed. This approach facilitates the grouping of like functionality. In this way, the executable can be thought of as a tree of callbacks (see Figure 6). Any sub-tree of this execution tree may be selectively pruned or simply paused, automatically doing the same to its children.

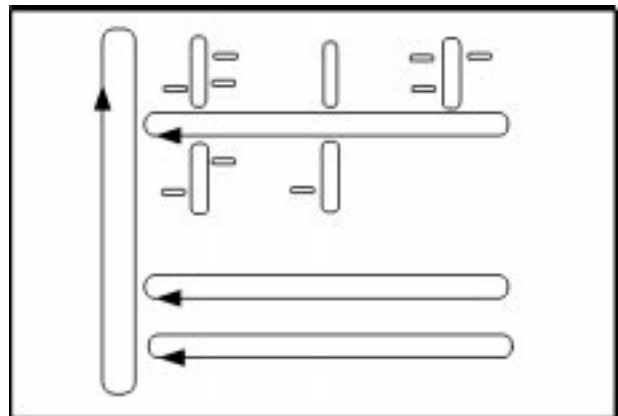


Fig 6 Single threaded execution tree view.

Enabling Dynamic Protocols

Now that a method for dynamically loading new code into a networked application has been established, it is interesting to consider how these modules might be used in a distributed VE. The initial use for modules, as suggested by the original Bamboo paper (Watsen, 1998), is to enable various groups to share their results within each other's environments. These results typically represent the geometry, texture, sound, and/or behavior for some object in the VE. However, if the module could actually define a communication protocol, a fundamental shift to how networked VEs are defined might ensue. Naturally, all users could agree to use a common protocol, like DIS, but that would not add much to existing scenarios. Instead, moving towards the other extreme, imagine every user defining their own protocol. That is, the only way to represent a user in your environment is if your system first downloads that user's client module, which may not only include the above characteristics but also initiates and decodes network traffic with that user. This approach would not only enable new (never before seen) users to enter already existing simulations, but would also enable the packets sent from the host to its clients to be optimized for its particular needs.

Given this goal, the outstanding question remaining is how or when the module is supposed to get itself installed. That is, what event will initiate a system to download another user's client module. There are two basic solutions to this problem: proactive and reactive. The proactive approach is to have the system somehow actively identify that it is missing the module. The reactive approach is to have the module inform the system that it is missing. Both of these approaches are considered next.

Proactive Approach to Module Installation

A practical scenario illustrating the proactive approach is discovering the client module's URL in a VRML (Carey, 1997) file loaded off the network (see Figure 7). This scenario depicts a server-supported client module. That is, the dynamically loaded module must listen for packets being sent to it in order to maintain its state. It is interesting to note that some modules may not require server support, as they are able to maintain their own state internally (e.g. an autonomous agent), while other modules that do require server support might represent non-interactive objects not requiring knowledge of your existence in the VE (e.g. an environment server). But assuming that the client module actually represents another interactive user in the VE, it is not clear how the other user's system discovers your presence. It might be possible for the other system to infer your existence based on your downloading of its module or perhaps

the module itself sends a message back to the other system while being initialized.

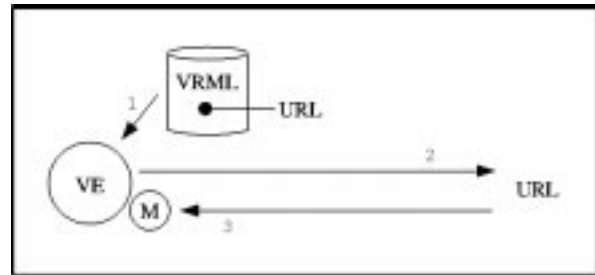


Fig. 7 Proactive module installation.

While it is possible for two interactive users to discover each other's existence using the proactive approach, the above scenario is best suited for non-interactive virtual objects limited to the space defined by the VRML file. However, participants in a shared space are likely to require greater mobility. It is therefore necessary to establish a more robust convention by which users can introduce and remove themselves from each other's systems.

Reactive Approach to Module Installation

The reactive approach requires users to actively inform each other of their existence. Since the users do not know whom the others are until after runtime, each system must actively open and listen to a port on which it expects to receive client requests. A simple solution, implemented by DIS, is to have every object periodically broadcast a heartbeat message identifying its existence to the rest of the world. Not receiving a heartbeat from a user within a predefined timeout threshold indicates that the user is no longer being represented and can be safely removed from the system. However, network packet analysis of a DIS exercise identified that nearly 96% of all network traffic was due to these entity state packets (Pullen, 1995).

An alternative approach is to use reliable packet transmission to guarantee that individual "creation" and "destruction" notifications will be received without having to periodically send out heartbeat packets. Officially, reliable packet transmission is currently only available for unicast addressed packets. Unfortunately, addressing a unicast packet requires knowing the recipient's address beforehand, which is exactly the issue being solved. ISTP (Waters, 1997) avoids this issue by using a central server from which new users can receive information about all other users in the VE, and visa versa. This solution appears promising as the protocol's designers have been careful not to limit the system's scalability, as servers are often the bottleneck in distributed systems.

Another reliable packet transmission approach can be achieved by using one of the experimental reliable multicast protocols, including RMP (GlobalCast Communications, 1997), SRTP (Pullen, 1996), RMTP (Paul, 1997), and RAMP (Koifman, 1996). These protocols achieve an optimal solution as they provide reliable peer-to-peer communications to an unknown number of subscribers. Both the PARADISE Project (Holbrook, 1995), and the TASC (Smith, 1996) system implement distributed VEs using reliable multicast. As depicted in Figure 8, new users send creation notifications out on a predetermined configuration multicast channel (8.1). As some participants will introduce themselves after others, all participants agree to send a reply back to the new user's system when first discovered (8.2). Both message contain the URL from which the user's client module can be retrieved (8.3 & 8.4). No special precautions need to be taken into account for deletion. That is, the notification, a reliable multicast message, does not require a response.

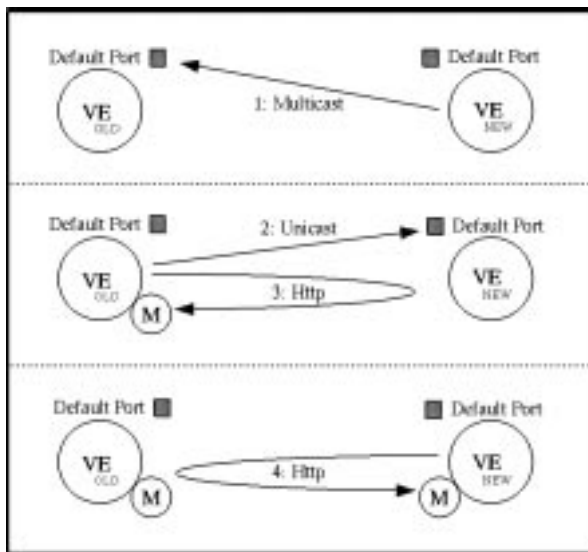


Fig. 8 Reactive module installation.

Implementing the Reactive Approach

The above scenario illustrates the basic approach but glosses over many important details. In particular, it has not been determined how the new user's system knows what multicast channel to send out its introduction on. Stated another way, imagine surfing the net and finding a description of an environment that you decide to check out. The way of the web is to follow links to their destinations, so will it be for joining the VE. Without going into all the details, the end result of this link should be your likeness (e.g. an avatar) immersed in the specified environment and interacting with the other users there.

The initial approach will be have Bamboo define an unique MIME type ".bar" (Bamboo Archive). Thus, the web browser will know to launch Bamboo and pass the file as a command line parameter. Bamboo, in turn, will test for a valid signature, unarchive it, and load it into memory. This module may have any number of dependencies, but it itself must define the application specifics of the environment that the user is joining. Rather than have this module define the whole environment, there are advantages to delaying the decision until later. In this way, the module is like a boot loader (e.g. LoadLin, Lilo, etc.) that helps load the rest of an operating system. The advantages gained by this delay pertain to the environment's relation to an AOIM (area of interest manager). An AOIM is used to reduce the complexity of a simulation to the subset that is only of interest to the client. Immediately loading the whole environment defeats the purpose of the AOIM, if one is being used it all.

The exact nature of the system's next step is completely determined by the loaded module. Many viable sequences exist, three of which we have tested in our lab. These sequences roughly correspond to DIS, HLA, and CN (completely new) and are described below but are first introduced by some theory that explains how they are all related.

Three Tiers to Network Management

Experience with implementing these dynamic protocols has led to the observation that there may exist (at least) three tiers to network management. These tiers, Global, Per-Environment, and Per-Object, describe the abstract network layer that traffic exists on to perform certain operations. For now, assume that each is implemented by a unique multicast address.

Global. This layer is shared world-wide and consists of very low level traffic used to synchronize environments. In particular, the sole purpose of this layer is to enable the discovery of environments.

Per-Environment. This layer is shared per-environment and consists of low level traffic used to synchronize objects. The primary purpose of this layer is to enable the discovery of objects. A secondary use may be the management of the environment itself.

Per-Object. This layer is used on a per-object basis and consists of traffic used to synchronize that object. The purpose of this layer is to enable the object's state to be transmitted to remote clients. There may exist multiple sub-layers, each corresponding to increasing or different fidelities.

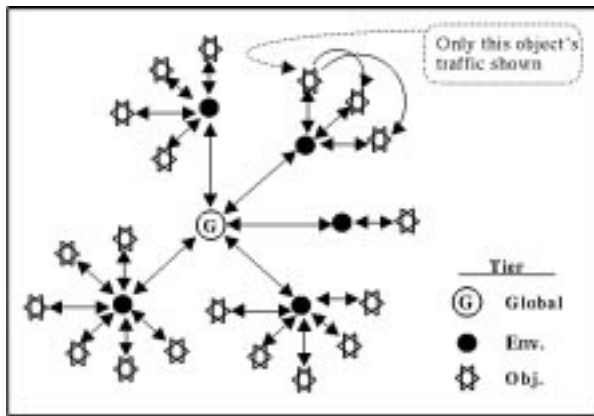


Fig. 9 Three tier network view.

Figure 9 illustrates the three tiers to network management. Note that there exists a single Global object that each Environment communicates with. Each environment also communicates with the objects using it. Finally, each object communicates with its clients located on the other systems sharing the same environment. Both the Global-Env. and Env.-Obj. traffic flows are transmitted using reliable multicast, while each object's peer-to-peer traffic is, most likely, unreliable.

Three Dynamic Protocols with Bamboo

As mentioned above, the next step to loading the environment depends on the module being loaded. The module must connect to the environment's network traffic. This process must consider how the networking is implemented. Following are the three attempts implemented in our lab.

DIS-Based Approach

The DIS-based test inserts a new and previously unenumerated entity into an already running DIS exercise. In terms of the three tiers to network management, DIS does not implement a concept similar to the Global layer. DIS does, however, implement the Per-Environment layer in terms of a hard-coded "exercise number" that is sent in the header of every PDU broadcasted to hosts on the local subnet. Furthermore, DIS also implements the Per-Object layer using an Entity ID that is also sent in the header of every PDU broadcasted. Because DIS uses broadcast, as opposed to multicast, the traffic from each of the three layers exists on the same virtual channel and thus becomes the host's responsibility to filter based on the information in the PDUs themselves.

All of the simulators participating in this exercise were running Bamboo with a DIS-lite module. This module only implemented a single Entity State PDU, having just position and rotation, and a special Dynamic Protocol PDU. Each simulator represented one entity in the VE via the sending out of these PDUs every frame (i.e. no dead-reckoning). Furthermore, each simulator maintained a view into the world such that the other entities could be seen from its first-person perspective. All but one of the simulators, each already having the logic to decode and visualize the default entity, would be initialized at the same time and allowed to reach a steady state (i.e. all the simulators knew of the others). At this point the remaining simulator would attempt to join the exercise by inserting a new entity type into the simulation via the Dynamic Protocol PDU. This protocol is similar to the standard Entity State PDU, in that it broadcasts position and rotation information every frame, except that its header includes a special field containing the URL of its client module. The first time the previously running simulators encountered this PDU, they would download and install the entity's client module, which would include the new entity type's geometry and runtime behavior. At this point the original simulators were able to visualize the new entity. If the new entity ceased to send its packets, it would timeout and be removed from the other simulators.

HLA-Based Approach

The HLA test's goal was similar to the DIS-based test – to dynamically incorporate a new object into an already running HLA simulation. In terms of the three tiers to network management, HLA also does not implement a concept similar to the Global layer. HLA's Per-Environment layer is defined by the "federation object model" (FOM) and implemented by the "runtime infrastructure" (RTI). Furthermore, its Per-Object abstraction is referred to as a "federate." HLA uses a multicast channel with a unique port per federation.

All simulators participating in the federation were running Bamboo with an HLA administration module. This module implemented a FOM that defined a single object that, like the DIS-based approach, contained only position and rotation values that were transmitted every frame. Each federate object was implemented by yet another module, that subclassed the known HLA Admin module. As the federates discovered each other, the name of the module to be dynamically loaded was determined by the "user supplied tag." When this module loaded, it would insert itself into the HLA administration structure and thus become part of the federation. Similarly, when the object ceased to exist, it would be removed from the federation.

CN-Based Approach

The CN-based approach tests the ability to insert a completely new object into an already running CN virtual environment. In terms of the three tiers to network management, the CN-based approach utilizes unique multicast addresses for the Global, Per-Environment, and Per-Object layers. As unique addresses are an important design specification, the MBone session management tool, SDP (Handley, 1998) is used to reserve the addresses in a network friendly way. Participating systems that are "CN-compliant" must implement the protocol defined for each of these layers.

In order to guide the development of the next section; please recall the scenario from the "implementing the reactive approach" above. In this example, assume that the downloaded module defines its environment by name. This name will become the key that opens the door through which the rest of the environment is loaded.

The Global layer is supposed to enable the discovery of individual environments. For this example, it simply maps the environment's name to a multicast address. If this multicast address is to be dynamically allocated, then a server listening to that address must allocate it the first time the environment's address is requested and released as soon as the environment is no longer in use. Fortunately, it is not necessary to implement a special server as this is exactly what SDP does. That is, the CN's Global layer can be completely implemented using existing multicast tools on the network.

Continuing with the example and using Figure 8 for reference, now that a unique multicast address is known, the system announces its existence to the environment via reliable multicast. This announcement not only identifies the URL where its client module may be downloaded, but also the configuration data that is passed into the module itself while its being initialized. Each participating system responds to the announcement with a reliable unicast message that also identifies its URL and configuration data. The configuration data specifies yet another unique multicast address on which the Per-Object traffic (unreliable multicast) will exist. The reason this data is not coded into the module itself is because it is possible that more than one system will choose to represent itself using the same server/client module combination.

At this point, all systems in the environment are synchronized. When an object leaves the environment, its host system transmits notification via reliable multicast on its Per-Object channel. Each of its client modules then removes itself from its host system. Each system also unsubscribes from the Per-Environment and Per-Object addresses so that SDP can reclaim those resources.

Availability

Bamboo is scheduled for a mid-1998 release, although beta versions are currently being made available. The standard distribution is a collection of header files, dynamically linkable libraries, Java class files, and an extensible runtime environment. A developer's distribution providing the source code is also freely available. There will be no licensing fee or shareware charge. Bamboo has been designed to be portable to many platforms by only using standard APIs (C++, Java, STL, JGL, OpenGL, etc.) and other multi-platform toolkits Fahrenheit (Silicon Graphics, 1997) and ACE (Schmidt, 1993). Although portability is not necessarily secured by this approach, the system's concurrent development on several platforms has not been hindered thus far. Furthermore, a mailing list has been established so that interested developers can freely exchange comments. Plans are being made to provide ongoing support and maintenance; developments will be announced on the mailing list as they become known. Additional papers and information may be found at <http://watsen.net/Bamboo>.

Conclusions

Bamboo overcomes many common VE system architecture pitfalls by enabling dynamic extensibility. Not only has this approach been shown to facilitate modular decomposition of functionality, but it also provides the ability to dynamically install networking protocols at runtime. This capability alters the fundamental approach to implementing large-scale virtual environments. It is hoped that this research will be applied towards the development of a shared, global, persistent VE, which would require dynamic extensibility given that it itself never goes down.

Although this paper has emphasized the applicability of this approach to virtual environments, there is actually nothing in Bamboo's kernel that is VE-specific. In particular, it is the modules that plug into the system that give an application its functionality. Therefore, this system might provide an appropriate infrastructure for routers, switches, and/or servers.

Acknowledgement

Bamboo has evolved over time as the result of the efforts of the main author and colleagues Joel Brand and Andrzej Kapolka. Research for this paper was enhanced by technical discussions with colleagues Howard Abrams, Don McGregor, and Don Brutzman. Special recognition is given to Stewart Liles for the HLA integration. Furthermore, the patience of Dr. Mike Zyda and the NPSNET Research Group has been appreciated. Finally, this effort could not have been without the generous support of our sponsors: DARPA, ONR, DMSO, and ANS.

References

- Ben-Natan, R. (1995). CORBA : A Guide to the Common Object Request Broker Architecture, McGraw Hill Text.
- Bricken, W. and G. Coco (1994). "The VEOS Project." Presence 3(2): 111-129.
- Brutzman, D., M. Zyda, et al. (1997). virtual reality transfer protocol (vrtp) Design Rationale. Workshops on Enabling Technology: Infrastructure for Collaborative Enterprises (WET ICE), MIT, Cambridge Massachusetts, http://www.stl.nps.navy.mil/~brutzman/vrtp_design.ps.
- Carey, R. and G. Bell (1997). The Annotated Vrm1 2.0 Reference Manual, Addison-Wesley.
- Carlsson, C. and O. Hagsand (1993). "DIVE - A Platform For Multi-User Virtual Environments." Computer and Graphics 17(6): 663-669.
- Coryphaeus (1997). EasyScene, http://www.coryphaeus.com/products_dir/es.html.
- Deline, R. (1993). Alice: A rapid prototyping system for three-dimensional interactive graphical environments. Computer Science Department. Charlottesville, University of Virginia.
- Division (1997). dVS, http://www.division.com/5.tec/a_papers/uvp.htm.
- GlobalCast Communications (1997). The Reliable Multicast Protocols, <http://www.gcast.com/reliablemulticast.html>.
- Greenhalgh, C. and S. Benford (1995). MASSIVE: a Distributed Virtual Reality System Incorporating Spatial Trading. Distributed Computing Systems (DCS'95), Vancouver, Canada, IEEE Computer Society.
- Handley, M. and V. Jacobson (1998). SDP: Session Description Protocol. RFC 2327, <ftp://ftp.isi.edu/in-notes/rfc2327.txt>: 42.
- Ho, W. W. and R. Olsson (1991). An Approach to Genuine Dynamic Linking.
- Holbrook, H., S. Singhal, et al. (1995). Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation. ACM SIGCOMM, <ftp://ftp.dsg.stanford.edu/pub/papers/lbrm.ps.gz>.
- IEEE (1993). Standard for Information Technology, Protocols for Distributed Interactive Simulation (DIS ANSI/IEEE standard 1278-1993), American National Standards Institute, 1993.
- Koifman, A. and S. Zabele (1996). RAMP: A Reliable Adaptive Multicast Protocol. Fifteenth Annual Joint Conference of the IEEE Computer and Communication Societies, San Francisco, CA., <http://www.tasc.com/arpa/tbone/ramp.html>, <http://www.tasc.com/simweb/papers/RAMP/ramp.htm>.
- Macedonia, M. R., M. J. Zyda, et al. (1994). "NPSNET: A Network Software Architecture for Large Scale Virtual Environments." Presence 3(4): 265-287.
- Paradigm (1997). Vega, <http://www.paradigmsim.com/vega.html>.
- Paul, S., K. K. Sabnani, et al. (1997). "Reliable Multicast Transport Protocol (RMTP)," IEEE Journal on Selected Areas in Communications.
- Pullen, M. and V. Laviano (1996). Selectively Reliable Transmission Protocol (SRTP) , <http://www.nac.gmu.edu/~vlaviano/>.
- Pullen, M. J. and D. C. Wood (1995). Networking Technology and DIS. IEEE.
- Schmidt, D. (1993). The ADAPTIVE Communication Environment: Object-Oriented Network Programming Components for Developing Client/Server Applications. 11th and 12th Sun Users Group, <http://www.cs.wustl.edu/~schmidt/SUG-94.ps.gz>.
- Sense8 (1997). WorldToolkit, <http://www.sense8.com/products/worldtoolkit.html>.
- Shaw, C. and M. Green (1993). The MR Toolkit Peers Package and Experiment, IEEE.
- Silicon Graphics (1997). Fahrenheit, <http://www.sgi.com/cosmo/cosmo3d>.

Singh, G., L. Serra, et al. (1994). "BrickNet: A Software Toolkit for Network-Based Virtual Worlds." Presence 3(1): 19-34.

Singhal, S. and M. Zyda (in preparation). Networked Virtual Environments, ACM Press.

Smith, W. G. and A. Koifman (1996). A Distributed Interactive Simulation Intranet Using RAMP, a Reliable Adaptive Multicast Protocol. Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations, Orlando, FL., <http://www.tasc.com/simweb/papers/disramp/index.html>.

Snowdon, D. N. (1994). "AVIARY: Design Issues for Future Large-Scale Virtual Environments." PRESENCE 3(4): 288-308.

Waters, R. C., D. B. Anderson, et al. (1997). The Interactive Sharing Transfer Protocol (ISTP) Version 1.0, MERL.

Watsen, K. and M. Zyda (1998). Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments. 1998 IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, Georgia.

Author's Biographies

Kent Watsen is pursuing a Ph.D. under Professor Mike Zyda while acting as project manager of the NPSNET Research Group in the Computer Science department at the Naval Postgraduate School in Monterey. He is the lead architect and developer of Bamboo, a virtual environment toolkit supporting, among other things, the next generation of NPSNET. His relevant experience includes the design and development of the character animation and 3D ocean modules for EasyScene, another virtual environment toolkit that he co-developed while working with Coryphaeus Software. He also developed Visual World, the rendering engine for a DIS simulator, while with DCS Corporation. Finally, He is responsible for a

raytracing-for-animation package developed as his undergraduate thesis. He holds Computer Science and Applied Mathematics engineering degrees from the University of Virginia. He is currently a co-chair of the VRML symposium and is actively publishing and presenting papers at both IEEE and ACM sponsored conferences. He can be emailed at kent@watsen.net.

Michael Zyda is a Professor in Department of Computer at the Naval Postgraduate School, Monterey, California. Professor Zyda is also the Academic Associate and Chair of the NPS Modeling, Virtual Environments and Simulation curriculum. He has been at NPS since February of 1984. Professor Zyda's main focus in research is in the area of computer graphics, specifically the development of large-scale, networked 3D virtual environments. Professor Zyda was a member of the National Research Council's Committee on Virtual Reality Research and Development. Professor Zyda was the chair of the National Research Council's Computer Science and Telecommunications Board Committee on Modeling and Simulation: Linking Entertainment & Defense. Professor Zyda is also the Senior Editor for Virtual Environments for the MIT Press quarterly PRESENCE, the journal of teleoperation and virtual environments. He is a member of the Editorial Advisory Board of the journal Computers & Graphics. Professor Zyda is also a member of the Technical Advisory Board of the Fraunhofer Center for Research in Computer Graphics, Providence, Rhode Island. Professor Zyda has been active with the Symposium on Interactive 3D Graphics and was the chair of the 1990 conference, held at Snowbird, Utah and the chair of the 1995 Symposium, held in Monterey, California. Professor Zyda began his career in Computer Graphics in 1973 as part of an undergraduate research group, the Senses Bureau, at the University of California, San Diego. Professor Zyda received a BA in Bioengineering from the University of California, San Diego in La Jolla in 1976, an MS in Computer Science/Neurocybernetics from the University of Massachusetts, Amherst in 1978 and a DSc in Computer Science from Washington University, St. Louis, Missouri in 1984. He can be emailed at zyda@siggraph.org.