NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments

MICHAEL CAPPS, DON MCGREGOR, DON BRUTZMAN, & MICHAEL ZYDA
capps@computer.org, mcgredo@cs.nps.navy.mil, brutzman@nps.navy.mil, zyda@acm.org
Department of Computer Science
Naval Postgraduate School

Recently in this column[1], we presented a vision of the future of computer graphics technology. The rapid improvement of rendering capacity and fidelity led us to search for the next bottleneck in the development of compelling virtual worlds. The content development process does not scale, and we predicted that intelligent, reactive content will be required to engage and entertain participants in a virtual environment. This intelligence can stem from computer-generated autonomy, or from other human participants.

In the months since the previous column, these trends have become even more pronounced. Microsoft demonstrated prototypes of their entry into the console gaming market, the XBox (xbox.com), which claims polygon throughput of 300 million per second. Sony's Playstation 2 gaming console (www.playstation.com) has been classified a munition by the Japanese government, and they plan a workstation market based on the console's graphics processor. Both consoles promise built-in broadband Internet connectivity in their U.S. versions, in an attempt to capitalize on the success on multiplayer gaming in the PC market. Likely the most telling event, though, was the speech at the X-Box unveiling—where Bill Gates said, "We think there's going to be incredible, persistent, online worlds that are created because of what you can do with [the network-enabled game console]."

We at the Naval Postgraduate School, and in the collaborative virtual environment community, have

[1] See the Projects in VR article in the November/December 1999 issue, "Cyberspace and Mock Apple Pie: A Vision of the Future of Computer Graphics"

been extolling the virtues of persistent online virtual worlds for years. It's both comforting (and sincerely frightening) to hear that same rhetoric from the world's largest corporation.

The bulk of the last column addressed the infrastructure requirements of a large-scale, persistent online virtual-world:

- run-time **extensibility** of both content and applications,
- **scalability** in world complexity and number of participants, and
- **composability** of heterogeneous content and applications.

Today, we follow those arguments with the first public introduction to NPSNET-V.

NPSNET Historical Perspective

The latest incarnation of the NPSNET system has been designed to meet the goals above, and thereby encourage the development of the next generation of Cyberspace applications. In order to appreciate the rationale behind those design decisions, it is first necessary to understand the previous steps in the NPSNET series. Space limitations in this column do not allow for a discussion of related work over the general field of networked virtual environments; for a complete literature review, we recommend the Singhal and Zyda textbook [3].

The NPSNET Research Group is the longest continuing academic research effort in networked virtual environments. The focus of the group is on the complete breadth of human-computer interaction and software technology for implementing large-scale virtual environments. There have been several generations of software formally named NPSNET. NPSNET-1 was demonstrated live at the SIGGRAPH 91 conference as part of the Tomorrow's Realities Gallery. NPSNET-2 and 3 were designed to explore better, faster ways to do graphics, and to extend the size of the terrain databases possible. NPS-Stealth spawned from NPSNET-1 in 1993, with the goal of developing a system capable of reading SIMNET terrain databases and SIMNET networking protocols.

NPSNET-IV [2] was built in 1993 to take advantage of sgi's new Performer library. The system was DIS-compliant, included dead-reckoning algorithms to reduce network traffic, and included spatial sound. NPSNET-IV has interoperated with almost every DIS-compliant virtual environment ever constructed, and was a significant advancement in the state of the art of scalable network architecture for virtual environments. It was undoubtedly the most popular NPSNET system, and was deployed in over a hundred military and civilian laboratories. In line with the Naval Postgraduate School's primary mission of education, NPSNET-IV also served as the base for a number of graduate theses. Accordingly, the code continued to grow over an additional three years as students added new functionality to the system. In less glamorous terms, this means that newly-trained programmers spent approximately six months modifying some part of a massive code-base consisting of 150,000 lines of C++ code. The engineering problems of such an approach are obvious: the single NPSNET-IV executable was extremely versatile, but invariably included modules or libraries that were not needed. And in order to make any improvements to the system, programmers would first spend months familiarizing themselves with the monolithic structure.

Virtual Environment Components

These lessons led us to realize the need for a component-based solution, a method rarely used in virtual environment systems. This fit with the findings of the first Shared Virtual Reality Workshop [1], which proposed that improvements to the state of the art would always be incremental without the ability to combine results from independently-designed systems. This of course begs the notion of a shared base-level component architecture.

Our initial approach was to design our own component architecture, as no system sufficed for all of our requirements: code modules needed to operate in a cross-platform and cross-language manner, and the system had to allow for dynamic discovery, replacement, and composition of those modules. NPS' Bamboo project, spearheaded by Kent Watsen, embodies all of those goals (http://npsnet.org/Bamboo).

Bamboo is currently in beta release after three years of open-source development; it runs on over 20 platforms and currently incorporates seven programming languages.

While Bamboo is an excellent technical achievement, such flexibility unsurprisingly comes at a cost of significant programming complexity. Additionally, our teaching situation is unchanged; students generally take months to grasp the subtleties of Bamboo, and therefore their opportunity to author virtual environment modules is severely limited. Also, development of a Bamboo-based virtual environment has been somewhat slowed by the lack of a cross-platform high-level graphics API. As discussed in the prior column, we believe that public acceptance of virtual environments will occur only when authoring is as simple as writing a web page. So, while we are continuing our first phase still very much look forward to the development of the first Bamboo-based virtual environment (currently code-named actiVE), we have had to turn away from Bamboo for NPSNET-V.

The Java programming language offers many of the benefits of Bamboo (cross-platform, networked module discovery, dynamic module composition), and with greater simplicity. Java additionally includes a cross-platform graphics API, Java3D. Choosing Java does lose the flexibility associated with multiple combined programming languages, and its use involves a signifcant loss of computation speed as compared to natively-compiled languages. Performance has been perfectly adequate to date—and as previously stated, expected improvements in the underlying hardware are already dismissing any concerns regarding the performance difference.

NPSNET-V Design

Our primary intent in designing NPSNET-V was to construct a platform for research on infrastructure technology for networked virtual environments. In the previous column, we explained the various benefits of dynamic extensibility and composition for virtual environments; NPSNET-V was designed to offer a tangible demonstration of those benefits. NPS students are already using those capabilities for

classroom and thesis work. We expect that the ability to substitute and modify base-level components will engender significant experimentation outside our institution as well.

In NPSNET-V, an Entity represents some object in the virtual world. Examples might be a tank, plane, tree, or person. An entity has state information—attributes like location, velocity, color, or orientation. An entity also maintains a list of the protocols it understands. These protocols represent a mapping between network messages and a set of actions or behaviors by the entity. A protocol receives some network message, determines its content, and possibly changes the state of the entity in response. Protocols drive entities—a protocol is an instance of behavior of which an entity is capable. To add new behaviors to an entity, one must add or modify a protocol.

Extensive experience with code design and reuse shows that it is desirable to separate the model of an entity from the view of an entity. In NPSNET-V this is done with an EntityView. The EntityView is the graphical representation of the object. (Existing EntityView types include Java3D immediate-mode, Java3D retained-mode, OpenGL, and plain-text.) The Entity maintains the ”model” of the object, which consists of programmer-defined state information, a list of protocols the entity understands, and a reference to the EntityView object.

In a distributed system, some entities will have authoritative state information, while other entities will only reflect the state of an object on another machine. A jet aircraft simulator will have complete information about the jet it is simulating—its speed, orientation, location, and graphical appearance, for example. That jet’s representation on other systems will need to have much of the same information, but it need not be completely authoritative or current. The speed-throughput tradeoff described by [3] suggests that for best results the remote representation will often be inconsistent with the current state information on the simulator. In NPSNET-V, the object with current, authoritative information is called the EntityMaster. Objects that represent the same entity, but that do not have authoritative data are called EntityGhosts. These two classes are actually quite similar at a high level. EntityMasters have more

behavior either hardcoded into them or driven by user input devices. EntityGhosts primarily shadow the state information in an EntityMaster. Typically there will be an EntityGhost on every machine interested in the EntityMaster object.

In a given NPSNET-V application, information needs to find its way from the network wire to entities. In NPSNET-V the primary object responsible for coordinating this information flow is the EntityDispatcher. The EntityDispatcher is responsible for managing the collection of sockets or other information sources and forwarding messages to the appropriate Entities.

When transmitting data, the EntityDispatcher prepends information about the entity the message is being sent to–an integer ID that represents the entity, and an integer protocol ID that identifies the intended interpreting protocol. On the receiving machine, another EntityDispatcher reads the information via its managed sockets and strips off the information the sending EntityDispatcher had prepended. This information is used to dispatch the message to the correct Entity and Protocol. Once at the protocol, the message can be interpreted; the protocol can, in turn, modify the state of the entity.

In this system there is minimal examination of message contents until the message arrives at the protocol. This is necessary because it is the protocol that actually interprets the message; the EntityDispatcher has no and needs no *a priori* knowledge of the semantics of messages. This allows us to both dispatch messages to entities, and at the same time dynamically load new protocols.

Extensibility

In the previous column, we argued that a persistent virtual environment must be extensible to be successful. Most virtual environments are statically extensible; that is, the environment implementation can be modified during a halt in execution, and clients must then be accordingly modified before rejoining the environment. An environment cannot be considered truly persistent unless it both (1) maintains state regardless of client participation, and (2) has no planned interruptions in execution.

NPSNET-V allows the addition of application components during run-time, thereby lending new functionality to a virtual environment. These components, which consist of Java classes, can be loaded from disk or over the network using the standard Java class loading mechanisms. New types of entities can be discovered during run-time in the same manner; the first time a client encounters an entity type, it has only to download its description (the entity's Ghost class) and create an instance. In this way, simple client applications can be dynamically extended to understand new environments and entities as they become available. This process can make use of Java's intrinsic security features, to help prevent exchange of malicious code, though this has not yet been important to incorporate.

This capability to create and share new entities recalls to mind the benefits of text-based MOOs, in which users could generate their own objects and behaviors. Such freedom to create is usually associated with virtual environments in only the most limited of terms—designing the graphical depiction of the avatar, assembling a residence from a template, and so forth. Concerns range from complex security issues to simple "look and feel" (multiplayer game designers have found that users rarely exhibit any artistic talents). It is hoped that the open NPSNET-V platform will provide a successful testbed for the possibilities in this realm, thereby encouraging community authoring in other virtual environments.

Dynamic Behavior Protocols

Dynamic discovery and modification of entities offers a number of significant benefits. Of primary interest to the NPSNET Research Group is the capability to modify an entity's network protocols during execution. This might be a switch of low-level protocols (such as between unicast TCP and multicast) or a modification to the application-level protocol (sending different data). Dynamic behavior protocols offer limitless possibilities for run-time network traffic configuration based on network load, environment changes, and the like. More importantly, dynamic protocols are the expression of behavior in virtual environments—they map network messages to events in the world. This means that loading new protocols at runtime can add new behaviors to the world, which is an important factor in the design of

scalable environments. This is discussed in a later section.

NPSNET-V allows multiple protocols per entity; all these protocols can be in operation at once. If a previously unrecognized protocol is requested, the class description is loaded into the Java Virtual Machine.

Dynamism in protocols can be divided into two distinct concepts: parsing messages to retrieve data from new packet formats, and deciding what to do with the data. The first addresses dynamic *syntax* recognition; the second, dynamic *semantics* recognition. Dynamic semantics recognition will always require dynamically loaded code be added at runtime. Dynamic syntax recognition can be automated and loaded from a simple description file with a suitable pre-existing infrastructure of interpreters. Implementing an interpreter for protocols is a NPSNET Research Group project led by Don Brutzman and Don McGregor called Dial-a-Behavior Protocol (DABP). DABP uses XML to describe the syntax of network messages. At runtime an engine reads the XML file and uses this information to extract data from binary format network messages.

NPSNET-V implements the dynamic addition of new behaviors via the loading of new protocol semantic interpretation code. This code must be able to extract information from network messages, which can be done in one of two ways: either hand-coded interpretation of packets, or via dynamic syntax recognition with DABP. DABP will often be faster to implement and require less tedious coding. Hand-coding the syntax portion of protocols may offer faster performance in some instances, or greater flexibility.

Composability

NPSNET-V offers infrastructure support for run-time composition of application components. By adhering to a basic interface, entities and their behaviors can be written independently and combined in a single environment. Because Java compiled byte-code is platform-independent, components for

heterogeneous platforms can be combined into a single operational application.

During the development of the Bamboo framework, we discovered the difficulty of providing a standard interface for communication between components. Though the interface complexity is reduced for NPSNET-V with the selection of a single programming language, no single technique was found to be appropriate in all cases. Here, we benefit from the dynamic nature of the system; even the module-interface libraries can be changed during run-time.

Though we do not enforce a standardized interface for communication between components, we hope that NPSNET-V can be used as a research platform for exploring inter-module and inter-entity protocols. This research field has exhibited slow progress because testing each method required constructing a new virtual environment system. We hope that dynamic interface loading will help to accelerate that research cycle.

The above exemplifies the philosophy of using NPSNET-V as a framework for experimentation. Performance is adequate for moderate environments, but the main intent is to provide a substrate for rapid prototyping of virtual environment concepts. For instance, we are currently exploring the use of XML, a structured-data description language which seems perfect for component specification and semantic description. XML figures prominently in our non-code-sharing implementation of dynamic behavior protocols. For the application builder who wants the shortest path to a fully-operational system, we include a recommended practice document and example base classes.

Networking and Scalability

A dynamically loadable component in a distributed component-based environment gives no benefit if it is limited to a single hard disk—the component must be available to machines on the network at large. A distributed, dynamic environment requires an integrated way to locate and download components. In NPSNET-V this is done via the Lightweight Directory Access Protocol (LDAP).

LDAP is a directory service—a distributed, cross-platform, standards-based database. Directories can have two attributes that are important to scalability: hierarchy and replication. Hierarchy allows scalability via partitioned responsibility. Parts of the directory namespace can be divided up and assigned to different authorities, much as the Domain Name System (DNS) is divided up into different domains and managed by different authorities. This delegation of authority makes DNS workable by assigning responsibility for a reasonably-sized domain to those most interested in maintaining correct information. A similar delegation of responsibility in virtual worlds would likewise make more workable the management of loadable components.

Replication allows the information contained in a directory on one machine to be automatically updated to other directory server machines. Classically server-based systems have had a single bottleneck and a single point of failure. Automatically replicated data would reduce exposure to this problem. By spreading the information out across several servers the load on any one server would be reduced, and the overall system would become less vulnerable to a single failure. Our initial implementation uses a single LDAP server, without replication or hierarchy, but we plan to expand the capabilities subsequent to further testing.

Inter-entity communications will generally not use this LDAP hierarchy, and therefore will not necessarily be limited to client-server traffic patterns. Multicast-based network architectures are generally acknowledged to be preferable, in the case where a large number of multicast groups are available. Unfortunately, since its inception, multicast has seen disappointingly limited WAN penetration. Many residential ISPs do not even support multicast backbone traffic, and often network interface cards do not support the multicast features needed for use in large-scale networked environments.

In response to this, NPSNET-V's dynamic protocol mechanism allows run-time selection of low-level Internet protocols, thereby enabling "best effort"-style network service. In the case that multicast support is not available, other low-level protocols are used. The EntityDispatcher supports automatic

switching between network-broadcast (with multicast) or manual-broadcast (with unicast) as dictated by this protocol choice.

Previous work has shown protocol-based filtering to be an effective mechanism for reduction of overall network traffic. Such extrinsic filters do not require parsing or semantic examination of data, and therefore better lend themselves to large-scale efforts. Normally, a participant in a virtual environment might have the choice to ignore certain types of entities. Protocol filtering in NPSNET-V gives the option to ignore classes of updates across all entities; for example, a radar-dish participant might well choose to ignore audio-traffic protocols of all kinds.

Decoupling protocol implementation from entity design, and allowing dynamic recombination of the two, gives incredible flexibility for scalable network design. Network architecture can be altered on the fly, by changing low-level network protocols; protocol-based filtering can be used to implement interest management; and protocol selection can even be made based upon load conditions, so that trade-offs between data accuracy and bandwidth consumption can be made at run-time. We look forward to NPSNET-V as an experimentation platform in all of these areas.

Building an NPSNET-V application

The source and binaries for NPSNET-V, like the previous NPSNET projects, are freely distributed. The Java classes, source files, and documentation are available at http://npsnet.org/NPSNET-V. While the project is not truly open source, in that the code is managed by the NPSNET Research Group, modifications and extensions by third party developers are welcome. There will also be a design review and tutorial at SIGGRAPH 2000 in New Orleans, at the “Developing Shared Virtual Environments” full-day course.

Acknowledgements

NPS is a partner in the Internet2 National Tele-Immersion Initiative, supported by Advanced Net-

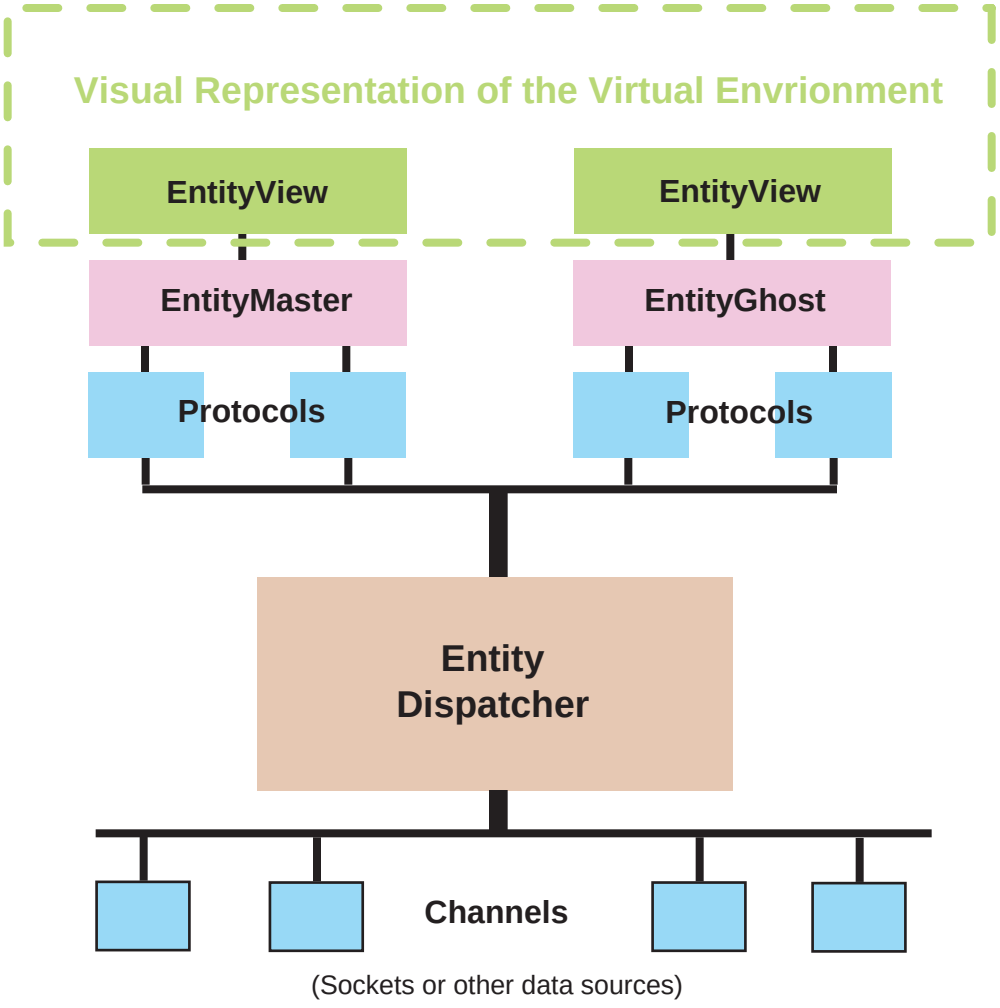
work and Services. Much of this research has been performed with an eye towards our expectations of next-generation Internet services for collaborative virtual environments. NPSNET-V and Bamboo have also been supported by the Navy N6M curriculum MOVES Research Center and the Center for Reconnaissance Research.

References

[1] Michael Capps and David Stotts. Research issues in developing networked virtual realities. In *Proceedings of the Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 205–211, Cambridge, MA, June 1997.

[2] Michael Macedonia, Michael Zyda, David Pratt, Paul Barham, and S Zeswitz. Npsnet: A network software architecture for large-scale virtual environments. *Presence*, 3(4):265–287, 1994.

[3] Sandeep Singhal and Michael Zyda. *Networked Virtual Environments*. SIGGRAPH Series. ACM Press Books, 1999. ISBN 0-201-3257-8, 352 pages.


Visual Representation of the Virtual Envrionment
EntityView
EntityView
EntityMaster
EntityGhost
Protocols
Protocols
Entity
Dispatcher
Channels
(Sockets or other data sources)