The Extensible Run-Time Infrastructure (XRTI): An Emerging Middleware Standard for Interoperable Networked Virtual Environments

Andrzej Kapolka
The MOVES Institute
Naval Postgraduate School
700 Dyer Road, M.E. 280
Monterey, CA 93943

Voice: 831-656-2265
Fax: 831-656-7590
akapolk@nps.navy.mil

I. Overview

The Extensible Run-Time Infrastructure, or XRTI, is an experimental implementation of proposed improvements to the High Level Architecture (HLA) [Kapolka 03]. The HLA, a middleware standard for sharing information between distributed simulation components, can act as the communications infrastructure for an enduring network of shared virtual worlds, but only if several of its shortcomings are resolved. First, in order to encourage widespread acceptance, it must be made easier to use. Second, to ensure that all applications based on the HLA can interoperate effortlessly, the HLA must specify a common message protocol for its middleware implementations (Run-Time Infrastructures, or RTIs) to use. Third, because supporting large-scale and long-running heterogeneous networked virtual environments requires the ability to introduce new classes of objects and interactions at run-time, the HLA must support the dynamic extension and composition of the Federation Object Models (FOMs) that act as its simulation schemas. Finally, to ensure that application developers always have access to a no-cost version of the HLA middleware, and to provide the RTI development community with a reference implementation and a test bed for future enhancements, there must exist a freely available, freely redistributable open-source RTI.

The XRTI addresses all of these issues, making it a candidate technology for connecting existing virtual environment applications in order to form a combined network of interoperable shared virtual worlds. To increase the HLA's ease-of-use, the XRTI includes a proxy compiler that converts the FOMs stored in arbitrary FOM Document Data (FDDs) into sets of fully documented Java™ proxy classes with consistent and intuitive type-safe interfaces. To ease the process of defining a common RTI message protocol, the XRTI employs a novel bootstrapping technique to define its messages in terms of HLA constructs and encodings. To allow applications to introduce new kinds of objects and interactions into active worlds, the XRTI uses its Reflection Object Model (ROM) to represent each world's FOM as a set of manipulable shared objects, and introduces the `mergeFDD` method as a means of dynamically extending the FOM. This extension capability is particularly important for applications such as those based on NPSNET-V, a component-based platform for dynamically extensible virtual environments. A set of XRTI controller modules allows NPSNET-V applications to exchange data using the XRTI. Combined with existing HLA controller modules and a simple test configuration, the XRTI modules also provide a context in which to compare the XRTI's performance to that of commercial RTIs. For a small-scale shared world, the XRTI offers a level of performance that matches or exceeds the levels offered by commercial RTIs. However, the current version of the XRTI is simply a research prototype, and as such it is missing many of the features required to turn it into a full-fledged RTI, let alone a complete solution to the problem of sharing data between networked virtual environment applications. It is the author's hope that by maintaining the XRTI as an open-source project on the World Wide Web, he can enlist the help of other developers both to complete the XRTI's development and to test the XRTI in real-world applications.

II. Background

The establishment of an Internet-scale, persistent network of highly interactive shared virtual worlds has long been the dream of military simulation developers, virtual environment researchers, video game players, and science fiction authors alike. Such a network would be the virtual reality equivalent of the World Wide Web: an environment where anyone with a browser application could join and participate

in any world without having to download and install additional software, and where worlds could be linked together so robustly that a user could drive a virtual vehicle through a portal from one world to another and expect the vehicle to appear and behave the same in the new world as it did in the old one. Although glimpses of such a network have appeared in the form of worlds based on the Virtual Reality Modeling Language (VRML), and in networked games and distributed simulations, there does not exist at present a mechanism by which one can connect these various applications in order to create shared worlds accessible to any kind of client. As with the World Wide Web, the growth of a network of shared worlds will rely on the establishment and maintenance of standards: both standards for data representation, like the Hyper-Text Markup Language (HTML), and standards for messaging between applications, like the Hyper-Text Transfer Protocol (HTTP). Such standards, if they are to accommodate any kind of environment, must be designed with extensibility in mind. In the example above, for instance, if the clients connected to the destination world are not familiar with the kind of vehicle introduced by the user, then they must dynamically extend themselves, their world description formats, and their messaging languages to adapt to its presence.

NPSNET-V, a component-based platform for shared virtual worlds, supports this kind of extensibility at the code level. Applications hosted within the NPSNET-V framework consist of hierarchies of dynamically loaded, loosely coupled modules rooted at an invariant microkernel [Kapolka 02]. Modules may be added to or removed from the framework at any time, and loaded modules may be hot-swapped—that is, seamlessly upgraded or otherwise replaced with new modules. Within the NPSNET-V platform, virtual worlds and entities are represented by sets of modules conforming to the Model-View-Controller design pattern, which requires that the abstract state, or model, of each entity be represented by a module that is separate from the view modules which present that state to the user, and from the controller modules which modify that state. NPSNET-V's controller modules include user interface controllers, physics controllers, and network controllers. The network controllers, which are responsible for transmitting or receiving the states of their models across application boundaries, have a unique problem. Because NPSNET-V can load new modules at any time, and because those new modules may have unforeseen communication requirements, whatever mechanism the controller modules use to exchange state data over the network must be amenable to dynamic extension; that is, the protocol or middleware interface used for networking must support the addition at run-time of new forms of data and new types of messages. Unfortunately, this kind of extensibility is not well supported by the mechanisms typically used for networked virtual environment applications—at least not by the mechanisms that provide the other qualities needed in a robust communications infrastructure for shared virtual worlds.

III. Problem Statement

At present, there exists no communications mechanism for networked virtual environments that is generalizable enough to support any kind of environment, usable enough to encourage widespread adoption, standardizable enough to allow universal interoperability, and adaptable enough to permit run-time extension of its ontology. These are the qualities necessary to support an Internet-scale network of diverse and compelling, persistent and robustly linked shared virtual worlds based on heterogeneous software elements. Without generalization, the mechanism would limit the type of worlds that one could create; without usability, application developers would avoid using the mechanism; without standardization, separate implementations could not interoperate; and without run-time extensibility, the mechanism would prevent worlds from evolving over time and adapting to new application requirements.

IV. Previous Work

Existing mechanisms fall into four different categories: heavyweight fixed protocols, composable micro-protocols, generic protocols, and middleware solutions. Heavyweight fixed protocols, such as the Distributed Interactive Simulation (DIS) protocol [IEEE 1278], attempt to satisfy the requirements of a particular application domain. The DIS standard, for instance, specifies a number of message types and other constructs specific to its target domain: platform-level military simulations. Although it supports a limited form of extensibility, DIS is ill-suited to supporting environments outside of that domain. The composable micro-protocol approach, as exemplified by the TreacleWell architecture [Oliveira 02], addresses the limitations of fixed protocols by allowing applications to use several protocols at once, with

each protocol devoted to carrying a specific kind of data. This approach, however, can be difficult to standardize and re-implement in contexts other than dynamically extensible component frameworks. Generic protocols, such as the Cross-Format Schema Protocol [Serin 03], allow application developers to redefine protocol syntax at run-time. Again, however, this approach is difficult to re-implement, as it requires the presence of an additional management layer to handle the distribution of new protocol schemas. Middleware solutions such as the Common Object Request Broker Architecture (CORBA) and the HLA solve the re-implementation problem by specifying an application programming interface (API) rather than a network protocol. Applications exchange data using the API methods of the middleware implementation, which assumes responsibility for networking with other middleware instances. Although developers have created networked virtual environments based on CORBA [Wilson 01], the HLA is particularly promising in that it includes features specifically intended for use in distributed simulation, such as region-based filtering.

However, the HLA has a number of limitations. Its API is awkward compared to that of CORBA, as it does not specify a means of creating new interfaces based on its object models, as CORBA does with its Interface Definition Language (IDL) compilers. The HLA also does not specify a common message protocol for its middleware implementations, RTIs, to use to communicate over the network. CORBA Object Request Brokers (ORBs), by contrast, can use the standardized Internet Inter-ORB Protocol (IIOP) to interoperate with one another. Nor does the HLA support the insertion at run-time of new object and interaction classes into the FOMs that act as templates for its simulations. Each simulation's FOM, which defines the types of objects that one can instantiate and the types of interactions one may send, along with the attributes of each object and the parameters of each interaction, becomes a fixed contract when that simulation begins. In order to extend the FOM, one must shut down the simulation and recreate it with the modified FOM. Furthermore, there are no open-source RTIs available. Simulation developers using the HLA must choose one of the proprietary RTIs, such as the DMSO RTI-NG [DMSO 03] or the MÄK RTI [MÄK 03]. Because the HLA does not specify a protocol for interoperability, these RTIs are not compatible with one another, and so all participants in a simulation must use the same RTI.

V. XRTI Design and Implementation

The XRTI resolves the HLA's shortcomings in order to establish itself as a candidate technology for supporting the network of shared virtual worlds described earlier. First, the XRTI takes advantage of the fact that the HLA standard [IEEE 1516] specifies exact encodings for FOM-described data types to define a message protocol in terms of FOM constructs. The XRTI's Bootstrap Object Model (BOM) acts as a protocol specification, making the process of establishing a common RTI message protocol similar to the act of distributing a FOM for a new type of simulation. To make the HLA easier to use for application developers, the XRTI includes a proxy compiler that converts the Extensible Markup Language (XML) FDDs that represent FOMs into sets of type-safe Java proxy classes with Javadoc comments containing the FOM's natural language semantics. The XRTI makes extensive use of these autogenerated proxies internally, using them, for instance, to send and receive the low-level interactions defined by the BOM. To allow applications to introduce new classes of objects and interactions into running simulations, the XRTI introduces a Reflection Object Model (ROM) with object classes whose instances represent FOM elements, much as the Java language represents loaded classes with instances of `java.lang.Class`. Each simulation's collection of reflection objects acts as a live, manipulable version of its FOM. At the API level, the XRTI provides the `mergeFDD` method to allow applications to extend this live FOM by merging it with the FOM contained in a new FDD.

The current version of the XRTI is a research prototype with limited functionality, designed to act as a proof-of-concept and a basis for further development. It does not support, for instance, advanced features such as the HLA's attribute ownership management, time management, and region-based filtering services. The current version requires the presence of a server application, the XRTI Executive, to handle all communication between simulation participants. Participants create instances of the XRTI Ambassador class, which in turn connect to the XRTI Executive. To create, join, leave, and destroy simulations, as well as to send and receive information, participants invoke the methods of their XRTI Ambassadors. When XRTI Ambassadors receive new information from the XRTI Executive, they notify their applications through the Federate Ambassador callback interface. Proxy objects reside between the application and the

XRTI Ambassador, turning the Java method calls of the application into HLA interactions and attribute value updates, and capturing the XRTI Ambassador's callbacks in order to notify the application through specialized, autogenerated interfaces.

VI. Integration into NPSNET-V

NPSNET-V's XRTI controller modules use a set of proxies generated from the Real-time Platform Reference FOM, or RPR-FOM: a standard FOM based on the DIS protocol. The XRTI's proxy compiler turns the 11,281 line long RPR-FOM FDD into 176 proxy classes with 21,499 non-comment source statements and 2,049 lines of Javadoc comments. The XRTI controller modules use only a small portion of this interface to transmit the position, orientation, velocity, and acceleration of their target models. To reduce the amount of bandwith that they use, the controller modules employ a dead-reckoning algorithm to extrapolate the model's position over time, transmitting updates only when the model diverges from its predicted path.

VII. Performance Testing

To compare the XRTI's performance to that of commercial RTIs, the author performed a simple test in a LAN environment. With two clients running an NPSNET-V virtual world consisting of a shark swimming in a circle, the author measured the performance of the DMSO RTI-NG 1.3v6, the MÄK RTI v2.0.3, and the XRTI prototype in terms of a comprehensive set of metrics: the average frame rate in frames per second, the average best-effort and reliable interaction latencies in milliseconds, the average incoming and outgoing network transfer rates in kilobytes per second, and the time in seconds spent in the representative `emitEntityState` method. The following table contains the results.

	DMSO	MÄK	XRTI
Average frame rate (FPS)	42.4	48.1	49.3
Average best-effort interaction latency (ms)	32.8	17.3	6.5
Average reliable interaction latency (ms)	38.7	17.3	5.7
Average incoming network transfer rate (KB/s)	25.8	16.4	8.1
Average outgoing network transfer rate (KB/s)	16.1	11.0	15.0
Time spent in `emitEntityState` method (s)	6.0	3.0	3.0

Table 1. Results of the performance test. The four rows of the data represent the different performance metrics; the three columns represent the RTIs tested.

These results indicate that, for a small-scale NPSNET-V shared virtual world, the XRTI matches or exceeds the performance of the commercial RTIs. However, the fact that both the XRTI and NPSNET-V are written in Java, whereas the other RTIs use the Java Native Interface (JNI) to invoke native code from Java bindings, may affect the results. Using the JNI may introduce added latency to Java method calls.

VIII. Future Work

The XRTI will require a substantial amount of work to turn it from a research prototype into a full-featured RTI, and finally into a complete solution to the problem of dynamically extensible networking. It must support the complete HLA standard, including the ownership management, time

management, and data distribution management services, as well as the ability to manage distributed save, load, and synchronize operations. Once it supports the IEEE 1516 standard completely, the XRTI will be a candidate for DMSO verification, which will establish it as an officially certified HLA RTI. In order to use the XRTI in languages other than Java, it will require special bindings, as well as versions of the proxy compiler that target different languages. Reducing the bottleneck created by the XRTI Executive will require the investigation of peer-to-peer techniques, with the eventual goal of eliminating altogether the need for a central server application. Finally, making the XRTI's proposed extensions part of the HLA standard will require a significant amount of interaction with the HLA community.

IX. Obtaining and Contributing to the XRTI

The XRTI Web site [NPSNET 03] provides access to all of the software developed and documentation written in support of the XRTI project. The source code for the XRTI is available under the Berkeley Systems Distribution (BSD) license from SourceForge, a Web site devoted to the open-source community. The author welcomes the contributions of anyone interested in participating in the XRTI's development, and encourages them to contact him to coordinate their efforts with his.

X. Summary

By addressing the shortcomings of the HLA, the XRTI provides part of a means to realize the longstanding vision of an Internet-scale network of shared virtual worlds held by many in the virtual environment, simulation, and gaming communities. The XRTI combines the HLA's middleware basis and simulation-specific functionality with its own contributions: a proxy compiler that converts FDDs into sets of easy-to-use, type-safe Java proxy classes; a message protocol described by a standardizable FOM; and a reflection mechanism that allows applications to introduce new object and interaction classes into running simulations. The XRTI is open-source and freely redistributable, yet even its initial prototype provides a level of performance comparable to that of closed source RTIs. Future versions of the XRTI will provide all of the functionality required by the HLA standard, and will include support for languages other than Java and networking profiles other than the client-server model. To demonstrate to the HLA community the need for and feasibility of the improvements proposed by the XRTI project, the author hopes to enlist the help of the open-source development community to advance the XRTI's development and to test the XRTI in new applications.

References

[DMSO 03] Defense Modeling and Simulation Office. "Runtime Infrastructure (RTI)." [https://www.dmso.mil/public/transition/hla/rti/]. October 2003.

[IEEE 1278] The Institute of Electrical and Electronics Engineers, Inc., IEEE 1278-1993. *IEEE Standard for Information Technology—Protocols for Distributed Interactive Simulation Applications.* 18 March 1993.

[IEEE 1516] The Institute of Electrical and Electronics Engineers, Inc., IEEE Std 1516-2000. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Framework and Rules.* 11 December 2000.

[Kapolka 02] Kapolka, A., McGregor, D., and Capps, M. "A Unified Component Framework for Dynamically Extensible Virtual Environments." *Proceedings of the Fourth ACM International Conference on Collaborative Virtual Environments*. 30 September-2 October 2002.

[Kapolka 03] Kapolka, A. *The Extensible Run-Time Infrastructure (XRTI): An Experimental Implementation of Proposed Improvements to the High Level Architecture.* Master's Thesis. Naval Postgraduate School. Monterey, California. December 2003.

[MÄK 03] MÄK Technologies. "MÄK Technologies High Performance RTI." [http://www.mak.com/rti.htm]. October 2003.

[NPSNET 03] The NPSNET Research Group. "XRTI: Extensible Run-Time Infrastructure." [http://www.npsnet.org/~npsnet/xrti/]. October 2003.

[Oliveira 02] Oliveira, M., Crowcroft, J., and Slater, M. "TreacleWell: Unraveling the Magic 'Black Box' of the Network." *Proceedings of the Sixth World Multiconference on Systemics, Cybernetics, and Informatics*. Orlando, Florida. 14-18 July 2002.

[Serin 03] Serin, E. *Design and Test of the Cross-Format Schema Protocol (XFSP) for Networked Virtual Environments.* Master's Thesis. Naval Postgraduate School. Monterey, California. March 2003.

[Wilson 01] Wilson, S., Sayers, H., and McNeill, M.D.J. "Using CORBA Middleware to Support the Development of Distributed Virtual Environment Applications." *Proceedings of the 9th WSCG International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision*. Plzen, Czech Republic. 5-9 February 2001.