RMIX: A Dynamic, Heterogeneous, Reconfigurable Communication Framework*

Christian Engelmann and Al Geist

Computer Science and Mathematics Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6164, USA
{engelmannc,gst}@ornl.gov
http://www.csm.ornl.gov

Abstract. RMIX is a dynamic, heterogeneous, reconfigurable communication framework that allows software components to communicate using various RMI/RPC protocols, such as ONC RPC, Java RMI and SOAP, by facilitating dynamically loadable provider plug-ins to supply different protocol stacks. With this paper, we present a native (C-based), flexible, adaptable, multi-protocol RMI/RPC communication framework that complements the Java-based RMIX variant previously developed by our partner team at Emory University. Our approach offers the same multi-protocol RMI/RPC services and advanced invocation semantics via a C-based interface that does not require an object-oriented programming language. This paper provides a detailed description of our RMIX framework architecture and some of its features. It describes the general use case of the RMIX framework and its integration into the Harness metacomputing environment in the form of a plug-in.

1 Introduction

Collaborative environments for heterogeneous distributed computing strive to enable research institutions and universities world-wide to pool their computing and storage resources in order to enable joint research in computational sciences, such as nanoengineering and quantum chemistry. Frameworks for metacomputing, data and computational grids, and peer-to-peer environments help to facilitate resource and data sharing among collaborating sites using standardized interfaces and interoperable software components.

Remote Method Invocation (RMI) is the most important communication paradigm for heterogeneous distributed collaborative environments as it extends the semantics of local method calls to networked systems. RMI is an object oriented analogy to the Remote Procedure Call (RPC) concept. It enables client components to invoke methods of objects that have been previously exported on

* This research is sponsored by the Mathematical, Information, and Computational Sciences Division; Office of Advanced Scientific Computing Research; U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725.

local or remote server components using a protocol stack that defines connection management, message formats and data encoding.

Traditional RMI communication frameworks typically implement one specific protocol stack, where connection management and message formats are defined by client and server libraries, and data encoding is defined using client-side and server-side stubs for each class of objects.

With this paper, we present a native (C-based), flexible, adaptable, multi-protocol RMI/RPC communication framework that complements the Java-based RMIX variant [1–4] previously developed by our partner team at Emory University as part of the Harness research effort [5–9].

RMIX is a dynamic, heterogeneous, reconfigurable communication framework that allows software components to communicate using various RMI/RPC protocols, such as ONC RPC, Java RMI and SOAP, by facilitating dynamically loadable provider plug-ins to supply different protocol stacks. While the RMIX base library contains functions that are common to all protocol stacks, like networking and thread management, RMIX provider plug-ins contain protocol stack specific functions for connection management, message formats and data encoding. Since it is up to the provider plug-ins to reuse base library functions, implementations may range from lightweight to heavyweight. Moreover, client- and server-side object stubs are very lightweight and protocol independent as they only perform an adaptation to the RMIX system.

Furthermore, RMI semantics have been expanded within RMIX to support the RPC paradigm and its protocols. In addition to standard synchronous RMI/RPC mechanisms, RMIX also offers advanced RMI/RPC invocation semantics, such as asynchronous and one-way.

This paper is structured as follows. First, we briefly discuss related past and ongoing research. We continue with a detailed description of the RMIX framework architecture and some of its features. We describe the general use case of the RMIX framework and its integration into the Harness lightweight metacomputing environment in the form of a plug-in. This paper concludes with a short summary of the presented research and its current status.

2 Related Work

The research in heterogeneous, adaptable, reconfigurable, networked systems (Harness) is a collaborative effort among Oak Ridge National Laboratory (ORNL), University of Tennessee, Knoxville, and Emory University focusing on the design and development of technologies for flexible, adaptable, reconfigurable, lightweight environments for heterogeneous distributed computing.

The C-based RMIX framework developed at ORNL and described in this paper follows a similar architectural design approach to the Java-based variant by offering the same interface and similar functionality. However, the C-based implementation is not a one-to-one translation of the Java-based solution as this would require reimplementing many of the Java language features in a non-Java

language, *i.e.*, reinventing the Java Virtual Machine (JVM) and parts of the Java Standard Development Kit (SDK).

Our approach is to offer the same multi-protocol RMI/RPC services and advanced invocation semantics via a C-based interface that does not require an object-oriented programming language. Instead of mapping RPC to RMI calls, like in the Java-based variant, we actually map RMI to RPC calls by emulating an object-oriented interface. Any object-oriented application just needs to implement the C-based stubs as they were method calls, with the exception that the first call argument is always a reference to the object itself. Polymorphism is guaranteed as the server-side stub calls the real object method.

The following related work has already been previously discussed in the context of the Java-based RMIX research [1–4].

XSOAP [10] (a.k.a. SoapRMI) is a RMI system based on the SOAP protocol and offers Java and C++ implementations to create and access Web Services. While it supports multiple transports and custom data encoding, it is not a universal multi-protocol RMI system.

JavaParty [11] is a drop-in replacement for standard Java RMI, written in pure Java and exploiting optimized serialization. While JavaParty supports non-TCP/IP communication networks, e.g., Myrinet, it is not interoperable with ordinary RMI applications and services.

The Manta [12] approach sacrifices Java portability and uses native code to achieve the best possible performance. Manta is a native Java compiler that compiles Java source code to Intel x86 executables. It does not offer any multi-protocol support nor does it provide a native equivalent.

Web Services [13, 14] have become a de facto standard for simplifying integration and access of heterogeneous networked services by presenting the user with a much lower level of abstraction for distributed computing. Web Service protocols are XML-based and are not designed for efficient data transfer. However, a RMI layer on top of Web Services, as exemplified by XSOAP and the Java-based RMIX variant, can offer simple and elegant access to heterogeneous resources over the Web.

3 RMIX Framework Architecture

The main goal of the RMIX communication framework is to provide efficient and interoperable RMI/RPC capabilities to system software, middleware and applications in a heterogeneous, distributed, collaborative computing environment. Conceptually, the RMI (and RPC) paradigm is based on a client-server architecture, where a client invokes a method (or function) at a server-side object. The RMIX approach allows each client-server pair to choose at compile time or even to negotiate at runtime the most efficient RMI protocol stack that is supported by both sides, while client-side and server-side object stubs remain the same as they only perform an adaptation to the RMIX framework and are not involved in the protocol stack.

The RMIX framework architecture (Figure 1) consists of two parts: a base library and a set of provider plug-in software modules. While the base library contains functions that are common to all protocol stacks, like advanced RMI semantics, networking and thread management, provider plug-ins contain protocol stack specific functions for connection management, message formats and data encoding. Since it is up to the provider to reuse base library functions, implementations may range from lightweight to heavyweight.

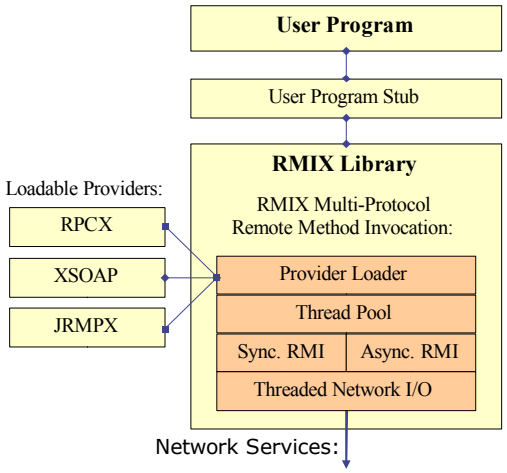

Fig. 1. RMIX Framework Architecture

The base library reuses technology developed earlier for the lightweight Harness run time environment [15], such as a thread pool to simplify thread management and a provider plug-in loader to allow dynamic adaptation. It also provides TCP/IP based networking support and advanced invocation semantics, such as asynchronous and one-way. Due to the pluggable nature of RMIX, provider plug-ins are able to extend these basic functions by supplying their own implementation or by loading other plug-ins.

RMI calls are mapped to RPC calls in part by the base library using an object registry that stores necessary object interface information at the server side. When exporting an object, the user has to specify the object interface including method names, signatures and respective stub function pointers. Server-side object stubs adapt incoming RPC calls to method invocations and client-side stubs adapt outgoing RMI calls to RPC calls. The adaptation performed by both stubs also includes transposing the RMI/RPC call arguments between C-function and argc/argv style in order to be able to pass them through the RMIX base library for encoding and decoding in provider plug-ins. The mapping of RMI to RPC calls is independent from the protocol stack.

3.1 General Use Case

In order to enable an application to perform remote method invocations using RMIX, it just needs to link the RMIX base library and supply the necessary lightweight client- and server-side object stubs.

First, an object needs to be exported at the server-side base library with protocol parameters, object interface and object pointer in order to accept incoming RMI calls. The base library registers the object, loads the appropriate provider using the specified protocol, and calls the provider via a standardized interface to evaluate the protocol parameters and to export the object. The provider returns a local reference for the exported object back to the user. Part of this local object reference is the remote object reference.

Remote object references may be dynamically stored at name servers, such as the RMIX registry, or statically assigned at compile time using advanced protocol parameters to force a specific export behavior.

When calling a remote method, the user invokes the client-side method function using the remote object reference as first argument followed by the original arguments. The client-side stub transposes the arguments to an argc/argv style and forwards them and additional object interface information to the base library, which loads the appropriate provider using the protocol specified in the remote object reference and calls it via a standardized interface.

Providers are free to use the base library components for encoding and networking, but are not forced to do so. On the server side, the provider receives and decodes the RMI call and looks up the server-side object interface and object pointer at the object registry in the base library. The provider calls the server-side method function with object pointer and argc/argv style arguments either directly or using a separate thread. The server-side stub transposes the arguments back to C-function style and calls the appropriate function or method. Any return values are passed from the server-side stub and provider back to the client-side provider and stub in a similar fashion.

Provider plug-ins may be preloaded before exporting an object or calling a remote method to improve performance. Furthermore, a provider registry file is used to store information about providers and supported protocols. Protocol parameters specified for exporting an object may be used to configure protocol stack parameters, such as type mapping. They are also part of the remote object reference to ensure symmetric protocol stack configuration.

3.2 Advanced RMI Semantics

The RMI paradigm is based on a request/response model, where the request message contains the call input and the response message holds the call output. However, applications are not interested in response messages if they communicate using the message passing paradigm. Furthermore, each RMI call takes a certain amount of time for remote processing causing the caller to wait idle for the response message.

RMIX supports one-way invocations via a separate invocation interface that allows the caller to continue after the request message has been sent and accepted. Any response is eliminated at the client-side provider plug-in in a separate thread to maintain RMI/RPC protocol compliance.

Asynchronous invocations are offered by RMIX via another separate interface that also allows the caller to continue after the request has been sent and accepted. The caller obtains an invocation reference in order to retrieve the response later. The client-side provider uses a separate thread to wait for the response and to store it locally. Multiple method invocations may be interleaved, *i.e.*, called in succession without retrieving the response in between. The server-side protocol plug-in guarantees the invocation order.

Asynchronous and one-way invocations have to be explicitly supported by a provider plug-in using separate implementations for each invocation style.

3.3 Remote Object Registry

Our C-based RMIX variant provides a name server style registry to dynamically associate remote object references with names similar to the Java RMI registry. However, in contrast to the Java RMI registry, the RMIX registry has multi-protocol support to improve interoperability. In fact, the RMIX registry is itself just a name server object that is exported using RMIX.

4 Harness Integration

Harness is a pluggable heterogeneous Distributed Virtual Machine (DVM) environment for parallel and distributed scientific computing. Conceptually, the Harness software architecture consists of two major parts: a runtime environment (RTE) and a set of plug-in software modules. The multi-threaded userspace RTE manages the set of dynamically loadable plug-ins. While the RTE provides only basic functions, plug-ins may provide a wide variety of services needed in fault-tolerant parallel and distributed scientific computing, such as messaging, scientific algorithms and resource management. Multiple RTE instances can be aggregated into a DVM.

The C-based RMIX variant has been integrated into the C-based lightweight Harness RTE [15] in form of a plug-in (Figure 2) to provide multiprotocol RMI/RPC capabilities to the RTE and to plug-ins. This effort also complements earlier work in integrating the Java-based RMIX solution into the Java-based Harness run time environment H2O [16].

The use case scenario of the C-based RMIX variant within the Harness context has already been discussed in [15]. While the RMIX base library and Harness RTE stubs are wrapped into a Harness-RMIX plug-in, stubs for other plug-ins are also implemented as plug-ins. Since the Harness RTE supports plug-in dependencies, a plug-in requiring RMIX automatically loads its stub plug-in(s), which subsequently load the RMIX plug-in.

Ongoing work in this area focuses on using both lightweight Harness RTEs to provide adaptability and interoperability in a heterogeneous collaborative environment for distributed scientific computing. Furthermore, we are also currently investigating parallel plug-in programming paradigms using the Harness/RMIX combination as a backbone for adaptive, fault tolerant, distributed, component-based scientific applications.

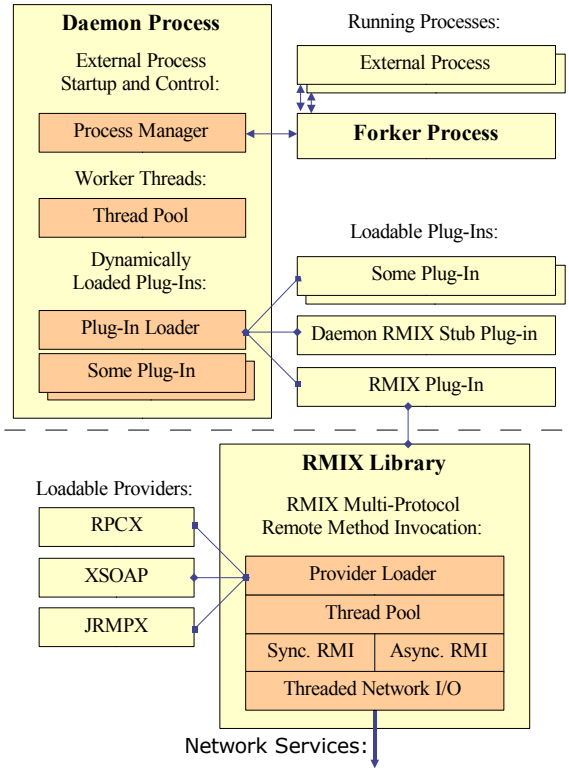

Fig. 2. RMIX Plug-in for the Harness Metacomputing System

5 Conclusions

With this paper, we presented a native (C-based), flexible, adaptable, multi-protocol RMI/RPC communication framework that complements the Java-based RMIX solution. We described the RMIX framework architecture, its general use case and some of its features, such as advanced invocation semantics, in more detail. We also explained recent integration efforts with the Harness lightweight metacomputing environment.

RMIX is part of the Harness software distribution package from Oak Ridge National Laboratory. Currently, we supply the Harness runtime environment together with the RMIX base library and the fully functional RPCX provider plug-in, which offers a ONC RPC compliant protocol stack using XDR encoding. The RPCX provider also supports one-way and asynchronous invocations. Ongoing work focuses on SOAP, IIOP and Java RMI (JRMP) provider plug-ins to further improve heterogeneity. Future work will target security related issues, such as authentication, authorization and encryption.

References

1. Kurzyniec, D., Wrzosek, T., Sunderam, V.S., Slominski, A.: RMIX: A multiprotocol RMI framework for Java. Proceedings of IPDPS (2003) 140
2. Kurzyniec, D., Wrzosek, T., Sunderam, V.S.: Heterogeneous access to service-based distributed computing: The RMIX approach. Proceedings of IPDPS - HCW (2003) 100
3. Kurzyniec, D., Sunderam, V.S.: Semantic aspects of asynchronous rmi: The RMIX approach. Proceedings of IPDPS - JavaPDCW (2004) 157
4. Wrzosek, T., Kurzyniec, D., Sunderam, V.S.: Performance and client heterogeneity in service-based metacomputing. Proceedings of IPDPS - HCW (2004) 113
5. Geist, G.A., Kohl, J.A., Scott, S.L., Papadopoulos, P.M.: HARNESS: Adaptable virtual machine environment for heterogeneous clusters. Parallel Processing Letters **9** (1999) 253–273
6. Sunderam, V., Kurzyniec, D.: Lightweight self-organizing frameworks for metacomputing. Proceedings of HPDC (2002) 113–124
7. Emory University, Atlanta, GA, USA: Harness project at http://www.mathcs.emory.edu/harness
8. Oak Ridge National Laboratory, TN, USA: Harness project at http://www.csm.ornl.gov/harness
9. University of Tennessee, Knoxville, TN, USA: Harness project at http://icl.cs.utk.edu/harness
10. Indiana University, Bloomington, IN, USA: XSOAP project at http://www.extreme.indiana.edu/xgws/xsoap
11. University of Karlsruhe, Karlsruhe, Germany: JavaParty project at http://www.ipd.uka.de/javaparty
12. Maassen, J., van Nieuwpoort, R., Veldema, R., Bal, H.E., Plaat, A.: An efficient implementation of Java's remote method invocation. Proceedings of PPoPP (1999) 173–182
13. Vasudevan, V.: A Web Services primer. http://www.xml.com/pub/a/2001/04/04/webservices (2001)
14. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl (2001)
15. Engelmann, C., Geist, G.A.: A lightweight kernel for the Harness metacomputing framework. Proceedings of IPDPS - HCW (2005) 120
16. Kurzyniec, D., Drzewiecki, D., Sunderam, V.S.: Towards self-organizing distributed computing frameworks: The H2O approach. Parallel Processing Letters **13** (2003) 273–290