

Enabling Remote Method Invocations in Peer-to-Peer Environments: RMIX over JXTA

Pawel Jurczyk¹, Maciej Golenia¹, Maciej Malawski¹, Dawid Kurzyniec²,
Marian Bubak^{1,3} and Vaidy S. Sunderam²

¹ Institute of Computer Science, AGH, Mickiewicza 30, 30-059 Kraków, Poland

² Emory University, Atlanta, USA

³ Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland

Abstract. In this paper, we present a peer-to-peer (P2P) system with remote method invocations, combining RMIX and JXTA technologies, and underpinning the H2O distributed resource sharing platform. We show that the integration of RMIX and JXTA was possible due to extensibility of the former, which allowed to plug in the JXTA-based socket implementations. The result of this integration is a fully operational RMI implementation running on top of the JXTA P2P network, where methods can be invoked on remote objects located behind firewalls or NATs. We present results of tests showing that our implementation can be used to connect peers in different LANs that cannot interact directly, while in the case of direct connection the performance is comparable to that of RMI using standard sockets.

Keywords distributed computing, P2P systems, JXTA, remote method invocation, H2O system

1 Introduction

Scientists and companies are looking for new ways of accessing the computational power needed for solving large-scale computing problems. This issue is addressed by Grid systems development which provides middleware for running applications on distributed resources shared between institutions. On the other hand, huge potential computing power is located inside millions of computers connected to the Internet. These resources can be used in a Peer-to-Peer (P2P) fashion, as was demonstrated by the SETI@home [1] project. The common feature of Grid and P2P systems is the resource sharing and distributed nature of the environment. However, large administrative effort is required to set up the Grid infrastructure and to establish virtual organization security. On the other hand, P2P systems are more suitable for ad-hoc collaborations, characterized by more dynamic participation patterns than those observed in Grid systems.

Avoiding the administrative burden related to using Grid systems was one of the goals of the H2O [2] resource sharing platform. H2O proposes and implements the model, where the roles of resource providers, service deployers and users can be separated. In this model, providers can independently offer the CPU power of their machines by running H2O kernels, but the responsibility of service

(application) deployment is delegated to others. This makes resource sharing easier for providers, in the spirit of the P2P model. Communication in H2O is facilitated by RMIX, which is an extensible RMI-based framework offering the RMI programming model to H2O-based distributed applications.

In order to enhance the H2O with the ability to operate within a P2P environment, we are integrating H2O with JXTA P2P system. The goal of this work is to exploit JXTA mechanisms in order to enable resource sharing among peers which may be hidden behind NAT or firewalls, and which may dynamically join and leave the P2P network, possibly changing their locations. Our approach decomposes into two separate tasks: (1) enabling communication in the P2P environment and (2) enabling resource discovery in the P2P network. In our previous paper [3], we presented the basic concepts of our solution. In this paper, we describe in detail the RMIX-JXTA integration, which brings the RMI programming model to P2P systems and provides foundations for peer-to-peer resource sharing via the H2O framework.

This paper is organized as follows: First, we give the background on RMI as a programming model for distributed computing and RMIX as its specific instance. Next, we overview P2P technologies and JXTA as the proposed standard. Subsequently, we describe our approach of integrating RMIX with JXTA and we present results of preliminary tests.

2 RMI as a successful programming model and RMIX

The diversity of distributed programming approaches results in multiplicity and heterogeneity of distributed computing infrastructures. Remote Procedure Calls paradigm, exemplified e.g. by ONC-RPC and XML-RPC [4], gained popularity as it is based on common and well-understood function invocation semantics. Similarly, recent systems such as CORBA [5] or Java RMI [6] extend the notion of object-oriented programming to distributed environments via the concept of *distributed objects* whose methods can be invoked from remote locations.

Such Remote Method Invocation (RMI) model is natural and convenient in object-oriented languages such as Java. However, interoperability between different environments becomes an issue. For example, standard Java RMI implementation is based on the Java Remote Method Protocol (JRMP) that is sophisticated and full-featured, but limited in practice to pure Java systems. There are some other Java RMI implementations based on other protocols (e.g. RMI-IIOP [7] that enable connectivity with CORBA or JAX-RPC [8] using SOAP/HTTP for Web services connectivity) but usually these solutions sacrifice functionality. In these circumstances the RMIX project [9] has been initiated.

The main objective of the RMIX communication library is to enable using multiple, independent RMI protocol service providers within a single, RMI paradigm-based framework. RMIX is flexible and extensible, allowing integration of existing RMI implementations. Service provider implementations are pluggable at run-time and hot-swappable. Additionally, RMIX features several general-purpose enhancements over Java RMI, including dynamic stubs,

SSL support, runtime binding and customizable virtual endpoints that allow the same remote object to be accessed via different protocols such as SOAP, JRMP, SunRPC. The framework opens up a possibility of dynamic access control - the interceptor allows or denies method invocations depending on custom-defined logic. RMIX provides a set of new features in the method invocation model providing an asynchronous calls and a one-way calls [10], with precise semantics.

Importantly from the P2P environment point of view, RMIX communication library enables pluggability also at the byte transport level, via potentially user-supplied socket factories. RMIX socket factory model is more generic than the one defined by standard Java RMI, naturally supporting non-IP networks. This makes it possible to plug in a transport which uses non-host-port based addressing scheme, such as the JXTA P2P overlay.

3 P2P networks and JXTA P2P network implementation

One of the first incarnations of the P2P paradigm was the Napster application [11], used for file sharing purposes. Next generations of P2P networks tried to avoid centralized servers, and they provided mechanisms for distributed lookup (Gnutella [12]), also introducing the concept of peer hubs for more efficient operation (Morpheus [13]).

The idea of exchanging files in a Peer-to-Peer manner inspired people to share free CPU cycles and resources, since it has been realized that there are millions of mostly idle machines that could be linked and used as one big supercomputer. Example projects which tried to tap on that resources are SETI@home [1], GPU Project [14], JNGI [15] and Parabon [16].

There exist many libraries allowing to create custom Peer-to-Peer systems but most of them are platform- or protocol-dependent. They are tailored for specific types of networks and applications, such as e.g. file sharing, and thus lack the necessary flexibility to serve as a general purpose sharing middleware. That is why JXTA [17] has been introduced. JXTA is designed to be a set of open, language independent protocols that allows any connectible devices, from cell phones, PDAs, and notebooks to big servers, to communicate with each other as peers [18]. Currently, implementations of JXTA protocols exist in PERL, C and, of course, Java.

A *peer* is a central JXTA abstraction. When a peer connects to the network, it publishes its interface (the *endpoint*) for communication with the outside world. Peers can collect information about the network, available resources, and services. They may also gather in *peer groups* to form controlled and focused collaborations. A peer may belong to multiple groups at any given time.

One of the most important features of JXTA are its communication capabilities. Peers can create communication channels called *pipes*, which are used in the JXTA network to exchange messages. These virtual channels do not require a direct connectivity between the participating peers. For instance, if peers are behind NATs or firewalls, they can still communicate with the help of special *routing* peers that propagate necessary messages. Pipe messages can carry any

type of data, such as e.g. text messages, binary data, or even Java objects. In the Java implementation of JXTA, pipes are unidirectional. Peer endpoints can include multiple network interfaces, and JXTA tries to use the best one available. For instance, when peers are in the same subnetwork, pipes typically use direct TCP connections which greatly increases efficiency.

JXTA pipes are very useful, but they represent a relatively low level of abstraction. However, in the recent versions of JXTA, the socket API very similar to that of standard Java sockets has been implemented on top of pipes.

4 Advantages of a P2P system for distributed computing

Distributed resource sharing systems like H2O can draw many advantages from using a P2P RMI framework. In particular, such approach allows to extend resource sharing to environments spanning networks with NATs and firewalls. This is achieved by using flat addressing type in P2P systems. Moreover, independence between object address and its host IP address opens up a possibility of location-independent resource references, supporting mobility and dynamic load balancing. These new features in P2P distributed application frameworks will yield new possibilities for building global computing systems:

- simplicity in deploying distributed applications (no need of specialized configurations of routers or firewalls),
- wider distributed application range (users from private networks can participate in any distributed application),
- clients can use resources independently from their location,
- enabling ad-hoc collaborations and virtual computing groups (using peer groups in P2P network).

Motivation for using RMIX and JXTA technologies stem from the flexibility and genericity of JXTA on the one hand, and from the RMIX features, facilitating the integration with the P2P environment, on the other hand. Furthermore, since the JXTA Socket API is similar to Java Socket API used at the RMIX transport level, integration of both systems can be achieved relatively easily, bringing the advantages presented above.

5 Integration of RMIX with JXTA

The integration of JXTA and RMIX is achieved by implementing client socket factory and server socket factory interfaces defined by the RMIX communication library. By using JXTA Sockets abstraction in the mentioned socket factories, RMIX framework will be communicating on top of the P2P network.

To denote RMIX endpoint address in the P2P environment we have decided to use the following addressing model:

```
<string endpoint address>[@<group>[(param-1;param-2;...;param-n)]]
```

In its simplest form, an address can be a simple string such as *rmixEndpoint*.

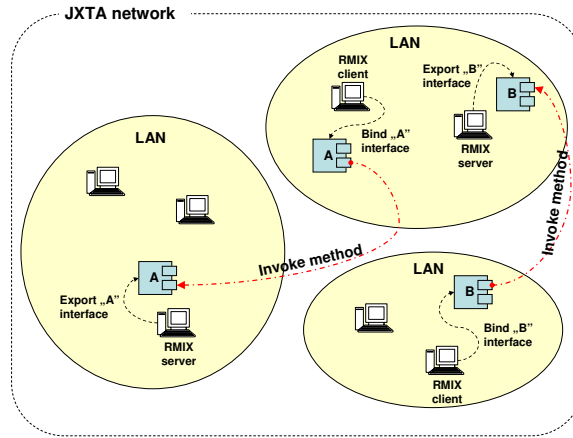
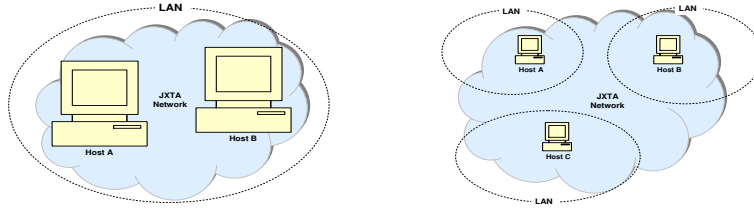


Fig. 1. RMI communication library in P2P environment.

Such addresses belong to JXTA *NetPeerGroup* and are accessible globally. Alternatively, the address can be narrowed to a user-specified group, by providing two optional parts - the first defining a group name and the second providing additional parameters used when joining the group. This approach enables flexibility (e.g. custom security controls) at the group membership level, by supporting groups using custom implementations of Membership Service. For example, address *myEndpoint@cGroup* denotes an RMI endpoint with address *myEndpoint* that is located in the *cGroup* JXTA group, with no security control. When one would like to use group with the security control, a valid address of RMI endpoint may look like *myEndpoint@cGroup(groupInterfaceImpl;p1;p2)*. Here, when joining *cGroup*, the custom security control defined in *groupInterfaceImpl* implementation of our *JxtaGroupInterface* interface will be used.

From the RMI user's point of view, the JXTA functionality is completely hidden. The user responsibility is limited to provide the JXTA socket address (usually embedded into a serialized stub received from a naming service or from another method invocation). The RMI recognizes the endpoint as JXTA-specific and delegates transport-related activities to JXTA socket factories. The system automatically connects to the JXTA network (becoming a JXTA peer), joins a group (if needed), manages rendezvous status of the current JXTA peer, and connects to the JXTA socket specified via the provided address.

The approach to combining RMI and P2P paradigms, described in this section, creates a possibility of using the RMI semantics in a P2P environment (as shown on Fig. 1). We argue that this approach creates new possibilities for developing distributed applications, by providing a simple remote method invocation model that can be used across any firewalls, NATs or private networks, in a location-independent fashion.



(a) Configuration that uses only a Local Area Network (b) Configuration that uses the JXTA network spanning several Local Area Networks

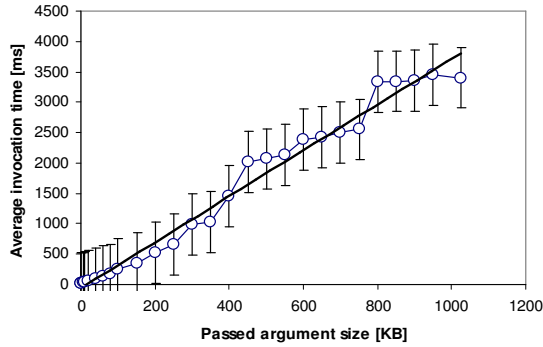
Fig. 2. Test configurations of the JXTA Socket Factories

6 Experimental Evaluation

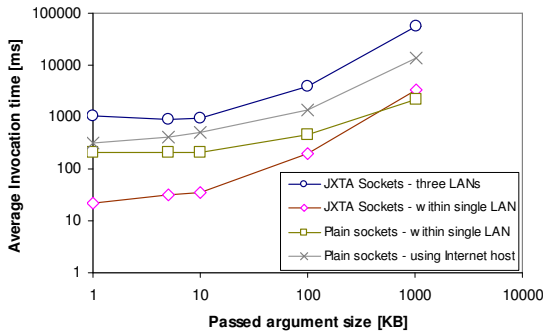
We have executed a simple benchmark to measure the performance of the RMIX communication framework in the P2P environment. We measure the round-trip time of a remote method call that accepts one argument of type `byte[]`, and returns a `String` object. The size of the byte array argument is a parameter of the benchmark. For each table size, we invoke the method 50 times and take the arithmetic average. The measurement was performed for two JXTA network configurations. In the first one (shown in Fig. 2(a)) JXTA network connects two computers from the same LAN. In this situation JXTA Sockets use direct TCP connections. The second test configuration 2(b) assumes that JXTA spans three LANs. Two of them contain the client and the server of our benchmark, while the third LAN contains a JXTA rendezvous peer providing connectivity between the client and the server. Additionally, to compare JXTA Socket with TCP socket performance, we run the same test using RMIX plain socket factories.

Results for JXTA Socket performance in a single LAN (the configuration shown in Fig. 2(a)) indicate linear increase of invocation time as a function of payload, which suggests that the communication is throughput-bound. Further experiments, illustrated in Fig. 2, show JXTA results for different setups, compared with invocation times for plain RMIX socket factories in two configurations: the first one is for two hosts inside a single Local Area Network, the second involves a host within a LAN connecting to a server that is somewhere on the Internet. Note that the chart presented in Fig. 3(b) has logarithmic scales for better clarity.

As could be expected, when the payload is large, JXTA Sockets are consistently slower than plain sockets. The reason is that despite using direct socket communication, there is still overhead induced by JXTA pipes and their messaging layer. Surprisingly, however, we observed that for smaller data sizes, JXTA Sockets performed faster than plain sockets. We attribute this phenomenon to the connectionless nature of the HTTP 1.0 protocol used by the Apache AXIS (SOAP 1.1 implementation used by RMIX), incurring a TCP handshake



(a) Single Local Area Network.



(b) Different network configurations

Fig. 3. Test results of RMIX using JXTA.

7 Summary

In this paper we discussed the role of a P2P-enabled RMIX communication library in the context of distributed resource sharing, and the H2O platform in particular. The need for a generic platform allowing to run arbitrary code on resources shared via a P2P network can be satisfied by the combination of H2O and JXTA technologies. Since RMIX is an underlying communication substrate of H2O, exploiting JXTA communication capabilities in H2O reduces to integrating them with RMIX. We have shown that this integration is possible because of the RMIX extensibility which makes it possible to use JXTA sockets as the transport mechanism. The result of this integration is the fully operational RMI implementation running over JXTA P2P network, where methods can be invoked on remote objects located behind firewalls or NATs, which is not possible in traditional RMI systems. Preliminary test results show that our implementation can be indeed used to connect peers in different LANs that cannot

overhead on each method call. In contrast, JXTA pipes internally use TCP connection pools, reducing the connection establishment overhead. This trend is reversed when we use a connection-oriented RMI protocol, such as JRMP/TCP. Nonetheless, we point it out as an interesting observation that performance of widely used communication technologies and implementations (such as the Apache AXIS) can sometimes be improved by *inserting* an additional layer into the invocation protocol stack.

On the other hand, we observed a significant drop of JXTA communication performance a "three-LAN" communication scenario. The causes of this overhead are twofold: 1) routing through an intermediary, and 2) necessity to perform additional HTTP 1.0 tunneling to traverse firewalls.

interact directly, while maintaining performance at near-native level when direct connectivity is achievable.

The proposed RMI over JXTA implementation provides a foundation to enable H2O resource sharing in a P2P environment. In particular, it exploits the P2P *connectivity* and allows H2O services to be accessible via JXTA. Our current work is focused on the complementary *discovery* aspects. By reusing decentralized P2P lookup mechanisms, coupled with global addressing and universal connectivity, powerful general-purpose P2P-enabled distributed computing platform can be enabled.

Acknowledgments This research is partly funded by the EU IST Projects CoreGRID, KWf-Grid and the Polish State Committee for Scientific Research SPUB-M grant. The authors are grateful to Piotr Nowakowski for his remarks.

References

1. SETI@home: Search for extraterrestrial intelligence at home (2003) <http://setiathome.ssl.berkeley.edu/>.
2. Kurzyniec, D., Wrzosek, T., Drzewiecki, D., Sunderam, V.: Towards self-organizing distributed computing frameworks: The H2O approach. *Parallel Processing Letters* **13**(2) (2003) 273–290
3. Jurczyk, P., Golenia, M., Malawski, M., Kurzyniec, D., Bubak, M., Sunderam, V.S.: A system for distributed computing based on H2O and JXTA. In: *Cracow Grid Workshop, CGW'04, December 13–15, 2004, Kraków, Poland* (2005) 257–268
4. Scripting News, Inc.: XML-RPC Home Page (2005) <http://www.xmlrpc.com/>.
5. Object Management Group, Inc.: CORBA (2005) <http://www.corba.org/>.
6. Sun Microsystems, I.: Java RMI (2005) <http://java.sun.com/products/jdk/rmi/>.
7. Sun Microsystems, Inc.: Java RMI over IIOP (2005) <http://java.sun.com/products/rmi-iiop/>.
8. Sun Microsystems, Inc.: Java API for XML-Based RPC (JAX-RPC) (2005) <http://java.sun.com/xml/jaxrpc/>.
9. Kurzyniec, D., Wrzosek, T., Sunderam, V., Słomiński, A.: RMIX: A multiprotocol RMI framework for Java. In: *Proc. of the Intl. Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France, IEEE Computer Society* (2003) 140–146
10. Kurzyniec, D., Sunderam, V.S.: Semantic aspects of asynchronous RMI: The RMIX approach. In: *Proc. of 6th Intl. Workshop on Java for Parallel and Distributed Computing, IPDPS 2004, Santa Fe, New Mexico, USA, IEEE Computer S.* (2004)
11. Napster, LLC.: NAPSTER Music Service (2005) <http://www.napster.com>.
12. OSMB, LLC: Gnutella File Sharing Network (2001) <http://www.gnutella.org>.
13. StreamCast Networks, Inc.: Morpheus File Sharing System (2005) <http://www.morpheus.com/>.
14. GPU Team: GPU project (2005) <http://gpu.sourceforge.net/>.
15. Verbeke, J., Nadgir, N., Ruetsch, G., Sharapov, I.: Framework for peer-to-peer distributed computing in a heterogeneous, decentralized environment. *Lecture Notes in Computer Science* **2536** (2002) 1–12
16. Parabon Computation, Inc.: Parabon distributed computing platform (2005) <http://www.parabon.com/index.jsp>.
17. CollabNet, Inc.: JXTA Home Page (2005) <http://www.jxta.org/>.
18. Sun Microsystems: JXTA v2.3.x: Java Programmers Guide. (2005) http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf.