Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL

Matjaz B. Juric *, Ivan Rozman, Bostjan Brumen, Matjaz Colnaric, Marjan Hericko

Institute of Informatics, Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova 17, 2000 Maribor, Slovenia

Abstract

This article analyses two most commonly used distributed models in Java: Web services and RMI (Remote Method Invocation). The paper focuses on regular (unsecured) as well as on secured variants, WS-Security and RMI–SSL. The most important functional differences are identified and the performance on two operating systems (Windows and Linux) is compared. Sources of performance differences related to the architecture and implementation are identified. The overheads related to the usage of security and the influences of JCE (Java Cryptography Extension) security providers on the performance of secured remote invocations are identified. Finally, the impact of distributed models on design and implementation of distributed applications is identified and guidelines for improving distributed application performance in design and implementation stage are provided. The paper contributes to the understanding of functional and performance related differences between Web services and RMI and their secure variants, WS-Security and RMI–SSL.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Web services; WS-Security; RMI–SSL; RMI; Java; Performance

1. Introduction

Development of distributed applications on Java platform requires us to select between the XML (Extensible Markup Language) based Web services or a distributed object model such as RMI (Remote Method Invocation). Web services are a commonly accepted model, which enables interoperability between different software platforms, such as Java and .NET. RMI is limited to Java platform, although interoperability with CORBA (Common Object Request Broker Architecture) can be achieved using the IIOP (Internet Inter-ORB Protocol).

For developers using primarily Java platform the selection between Web services and RMI, and their secure variants, WS-Security and RMI–SSL, can become a relatively difficult decision. This is why we have done a functional comparison. When selecting the appropriate model the performance criterion can also be important. Therefore, we have done a detailed performance comparison on two operating systems, Windows and Linux. An increasingly important issue in the development of distributed applications is security. In Web services the authentication and the message-level security is achieved using WS-Security, while in RMI the SSL (Secure Socket Layer) is used. To identify the performance overhead of using security we have done a comparison of regular (unsecured) Web services and RMI with secured Web services (WS-Security) and RMI–SSL, and measured the influence of JCE security providers on performance.

The review of related work has shown that not much has been done on the analysis and performance comparison of Web services and RMI. There are some articles

---

* Corresponding author. Tel.: +386 2 235 5113; fax: +386 2 235 5134.
E-mail address: matjaz.juric@uni-mb.si (M.B. Juric).

with the objective to improve performance and efficiency of RMI without the need to preserve compatibility with standard RMI. These works do not compare RMI to other distributed models: Rivera et al. (1997) have presented improved performance of RMI with reducing network delays. They have used Illinois Fast Messages as the alternative transport layer and implemented Java sockets interface. Vijaykumar et al. (1998) have explored transport protocols and object caching to improve the performance of RMI. They have presented a more efficient prototype system with new transport protocols and the possibility to cache objects at client nodes. Maassen et al. (1999) have studied the overhead of RMI related with passing and processing object type information at runtime, and compared it to the overhead of RPC (Remote Procedure Call). They have developed a RMI implementation based on native compilation and compile time generation of marshallers, presented an optimized protocol for RMI communication, and proved better efficiency with performance measurements. Nester et al. (1999) have presented a more efficient implementation of RMI achieved through optimizations in object serialization and RMI stack, such as slim encoding of type information, better buffering, less use of reflection, and local optimizations. They have also presented a set of benchmarks for RMI, which compared to the benchmarks in this work use only a limited set of data types (integer and float). Kono and Masuda (2000) have described a more efficient object serialization in RMI and presented performance measurements. Their approach dynamically specializes serializing routine for the receiving platform to directly convert the in-memory representations of objects. All these works demonstrate that RMI can be further optimized. We show the same but in contrast to these works preserve the compatibility with standard RMI. We also compare the functionality and the performance of different distributed models using more complex measurements, identify the bottlenecks, and provide guidelines for developing distributed applications. There are also some related open source implementations: Transparent RMI (SourceForge, 2004) is an extension to RMI that allows any interface to be used remotely and provides centralized recovery from remote exceptions. Mantaray (SourceForge, 2005) is a performance optimized distributed messaging middleware that supports connectivity through different standards, including RMI and SOAP. No performance measurements existed at the time of writing this article. Zeadally et al. (2004) have provided an empirical performance evaluation of native sockets, Java sockets, and RMI on Windows, Solaris, and Linux. They have measured latency and throughput, have shown that latency and throughput of RMI is not as good as native sockets, but have not provided optimizations and have not compared RMI–SSL, Web services, or WS-Security. The measurements method differs

considerably from ours. We measure round trip and instantiation times. In our earlier research we have compared the performance of RMI and CORBA (Juric et al., 2000a; Juric and Rozman, 2000). We have identified performance bottlenecks of RMI–IIOP and applied optimizations (Juric et al., 2000b) through which RMI–IIOP outperformed regular RMI. We have also compared RMI, RMI tunneling, and the transition to Web services (Juric et al., 2004). Our earlier articles have been based on a set of benchmarks for distributed object models, proposed in Juric et al. (1999). We use an updated version of these benchmarks in this paper. At the time of writing there was no performance analysis of Web services, WS-Security, RMI, and RMI–SSL. There was also no comparison of regular and secure usage of Web services and RMI.

The article is organized as follows: In Section 2, a functional comparison of Web services and RMI is given. In Section 3, the performance analysis method is presented. In Sections 4 and five local and remote performance analysis and comparison of RMI, RMI–SSL, Web services, and WS-Security on Windows and Linux is presented. In Section 6, the analysis and interpretation of the results is given with possible optimizations and the identification of consequences a distributed model has on application design and implementation.

2. Functional comparison of Web services and RMI

There are several important differences between Web services and RMI. RMI is a distributed object model and enables the development of distributed Java applications in which the methods of remote objects can be invoked from other Java virtual machines, possibly on remote hosts. The communication in RMI is based on the notion of distributed objects and follows the object paradigm. A distributed object exposes a remote interface through which the clients can invoke methods. RMI supports synchronous request/response method invocation. Distributed objects in RMI can be stateful and maintain their identities. The RMI system consists of three layers: the stub/skeleton layer, the remote reference layer and the transport layer (Sun Microsystems, 2003). The stub/skeleton layer provides support for client-side stubs and server-side skeletons, the remote reference layer for reference and invocation behavior and the transport layer for connection setup and management as well as remote object tracking. The RMI transport layer usually opens direct sockets to remote object hosts and uses binary protocol for communication. This can be either JRMP (Java Remote Method Protocol) which is a Java proprietary protocol; or the IIOP which enables interoperability with CORBA. The RMI communication is usually blocked by firewalls therefore it is appropriate mainly for communication

within LANs. The binary communication in RMI is not secured by default. Because it uses the TCP/IP as the underlying protocol, SSL or TLS (Transport Layer Security) can be used to secure the communication and to enable authentication of involved parties. The authentication is determined using the PKI (Public Key Infrastructure) digital certificates using RSA. The encryption of the communication is achieved using a symmetric encryption algorithm such as DES, Triple-DES, or AES (Sun Microsystems, 1998).

Web services are based on the SOA (Service Oriented Architecture) concepts. The major difference compared to RMI is related to the definition of information flow between the service and the client. Instead of object based, in Web services the information flow is based on XML formatted payloads (W3C, 2003) which act as input, output, and/or fault messages. The combination of these messages determines the operation types, which can be request/response or one-way (but also solicit response and notification) (W3C, 2001). Messages are defined using XML Schema. Web services support synchronous and asynchronous interactions. They use the XML-based SOAP (Simple Object Access Protocol) for communication. SOAP builds on top of existing Internet protocols, such as HTTP, FTP or SMTP, therefore it can transverse firewalls seamlessly. By default SOAP is unsecured. If it is used over TCP/IP we could use SSL or TLS for security, but this would lead to point-to-point security solutions. Therefore for Web services the WS-Security specification has been developed, which provides message-level security and is the preferred choice. WS-Security is a new specification which has only recently been supported in software. Similar as RMI–SSL it supports authentication (using X.509 certificates or other methods, such as Kerberos) and communication security using symmetric encryption.

Because Web services are not based on the object paradigm they lack some features found in RMI, particularly support for object references and stateful objects, dynamic class downloading, support for distributed garbage collection, and remote object activation. In contrast to RMI Web services are not bound to a particular programming language or software platform therefore they enable interoperability with services developed on other platforms (such as Microsoft .NET). Web services are described with the WSDL (Web Services Description Language), which enables language-independent description of types, messages, port types, operations, and ports. Instead of the RMI Registry Web services use the UDDI (Universal Description, Discovery and Integration). Table 1 shows comparison of RMI and Web services.

JWSDP (Java Web Services Developer Pack) provides APIs related to XML and Web services development in Java. JAX-RPC (Java API for XML based Remote Procedure Call, will be renamed to JAX-WS)

Table 1
Functional comparison of RMI and Web services

Functionality	RMI	WS
Synchronous operations	Yes	Yes
Asynchronous operations	No	Yes
Remote method/procedure invocation	Yes	Yes
Document oriented	No	Yes
One-way operations	No	Yes
Stateful objects/services	Yes	No
Dynamic class downloading	Yes	No
Automatic distributed garbage collection	Yes	No
Language independent wire protocol	No	Yes
Language indp. object/service description	No	Yes
Location of objects/services	Registry	UDDI
Dynamic binding to instances	No	Yes
Remote object/service activation	Yes	No
Interoperability through SW platforms	Partial (IIOP)	Yes
Security	SSL or TLS	WS-Sec

enables the development of Web services, which is very similar to the development of RMI remote objects. JAX-RPC Web service consists of an interface extending the java.rmi.Remote and a class implementing the interface. In contrast to RMI distributed objects Web services are typically executed within the web container as servlets. The choice is to transform Web service classes directly to servlets or to use a servlet handler which forwards the requests to appropriate classes. JWSDP uses the later technique.

Web services are a serious alternative to RMI. On one hand they provide interoperability with other platforms, on the other hand the development of Web services using JAX-RPC does not differ considerably from the development of RMI remote objects. They both use stubs on the client side and skeletons on the server side to mask the complexity of remote communication from the developer. Used that way the differences between RMI and Web services are not obvious to the developers at first sight. Therefore the performance question becomes important. In the next sections we present performance analysis and comparison results of Web services and RMI.

3. Performance analysis method

We have done a performance comparison of RMI, RMI–SSL, Web services, and WS-Security using a subset of the performance assessment framework (Juric et al., 1999). For the purposes of this article we have measured the round trip method invocation times and the instantiation times. Round trip method invocation time (RTT) is the time that elapses from the initiation of a method invocation by the client until the results are returned. Because the methods have not performed any processing, the round trip time expresses the overhead of remote method invocation. It is important to

understand how different data types influence the result; therefore we have measured round trip times for eight data types: integer, short, long, float, double, boolean, byte, and string. Instantiation time is the time needed for a client to achieve a connection to the remote object/service instance. The instantiation time does not include a registry lookup such as UDDI or advanced Web services retrieval (Zhuge and Liu, 2004), which is based on a multi-valued specialization relationships between services, measurement of similarity degree between services, and an SQL-like query language. We have achieved the comparability of the results between RMI, RMI–SSL, Web services, and WS-Security with identical implementations that have differed only in necessary details by obtaining the initial references. Further, we have used a consistent mapping between Java and Web services (XML Schema) data types (Sun Microsystems, 2004). For the Web services implementation we have used SOAP RPC/encoded data representation.

To measure the performance, we have defined the RTTTester interface (Listing 1) and developed the implementation classes. For each test 1000 invocations have been measured to gain the necessary resolution. Each test has been repeated 12 times, the maximal and minimal times have been discarded. The average time of 10 repetitions has been calculated. Variation and the coefficient of variation have been calculated too. For RMI–SSL and WS-Security we have used server and client-side key store with authentication on both sides. We have used X.509 certificates, RSA for digital signature, and Triple-DES for securing the communication.

Listing 1. Remote interface for performance tests

```
public interface RTTTester
                  extends Remote {
  int returnRTTforInt()
          throws RemoteException;
  short returnRTTforShort()
          throws RemoteException;
  long returnRTTforLong()
          throws RemoteException;
  float returnRTTforFloat()
          throws RemoteException;
  double returnRTTforDouble()
          throws RemoteException;
  boolean returnRTTforBool()
          throws RemoteException;
  byte returnRTTforByte()
          throws RemoteException;
  String returnRTTforString()
          throws RemoteException;
}
```

We have accomplished the measurements on identical equipment and environment. We have used two identically configured computers for each operating system, one acting as the client and one as the server. The computers had Intel Pentium 4 processors running at 2.4 GHz (FSB 133 MHz), with 512MB RAM, running only essential services, and restarted before each test to ensure the same starting conditions. They have been connected to a 100 Mb/s switched network, free of other traffic. To perform the measurements we have used the Java 2 Platform Standard Edition version 1.4.2_05 and Java Web Services Developer Pack version 1.4. Web server has been the Apache Tomcat v5.0. We have used the following three JCE (Java Cryptography Extension) security providers: Bouncy Castle Crypto version 1.24, Wedgetail JCSI Provider version 2.3, and OpenSource HBCI Toolkit for Java version 0.0.6. We have performed the measurements on Windows XP Professional with SP2, and Whitebox Enterprise Linux 3.0 (Respin 1, kernel-2.4.21-20.EL).

4. Performance analysis on single computer

4.1. Performance on Windows

First, we have measured the performance using single computer on which we have deployed the client and the server functionality (Fig. 1). This way we have avoided the network overhead. Instantiation has been considered separately. The times for basic data types (int, short, long, float, double, boolean, and byte) have not differed considerably, therefore we have calculated the geometric averages. This is because the data type payload represents only a small portion of the whole JRMP or SOAP message payload exchanged between the client and the server.

From Fig. 2, we can see that RMI and Web services behave as we have expected. RMI offers results which are an order of magnitude better than Web services. Using basic data types Web services are on average ~12.3 times slower than RMI, using strings they are
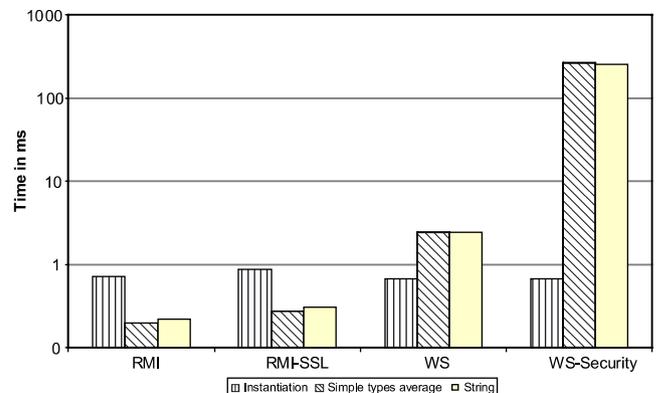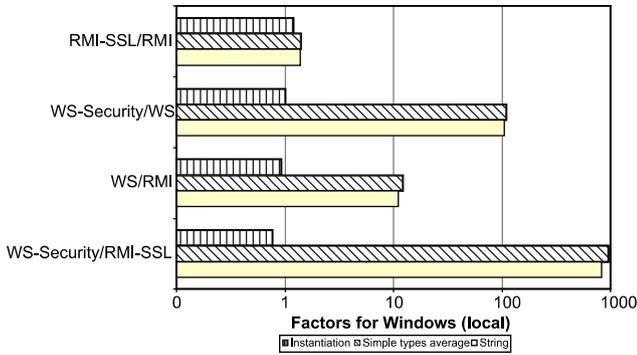

Fig. 1. Local performance results on Windows.

Fig. 2. Local performance factors on Windows.

~11 times slower. It is interesting that by the instantiation Web services are ~8% faster than RMI. Compared to our previous work (Juric et al., 2000a), RMI performance improved significantly from an average of 1.54 ms for simple data type local RMI calls to 0.21 ms in this research for the same data types. This is more than 7 times faster. Web services performance today is comparable to RMI performance, as measured in Juric et al. (2000a). We can also see that using secure communication and authentication slows the performance. RMI–SSL is on average ~40% slower than unsecured RMI. Using basic data types the slowdown is ~40.5%, using strings ~38.9%. The instantiation is ~20% slower.

Comparing WS-Security to unsecured Web services however gives a different picture. We can see that WS-Security is by factor 100 slower than Web services (~110 times for simple types, ~105 times for string, no differences by instantiation). To investigate the reasons for such slowdown we have repeated the measurements using three different JCE security providers: Bouncy Castle Crypto version 1.24, Wedgetail JCSI Provider version 2.3, and OpenSource HBCI Toolkit for Java version 0.0.6. The results are shown on Fig. 3. We can see that the differences between the JCE providers are minimal and not responsible for the slowdown. This is
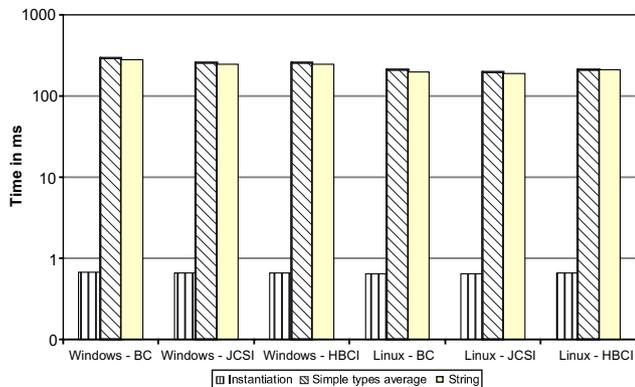
why we show the average results for WS-Security from all three security providers. JCSI and HBCI provide almost identical performance, while Bouncy Castle is slower by ~15% using simple types and strings and ~4% slower for instantiation. Comparing WS-Security to RMI–SSL shows that WS-Security is slower using simple data types by factor ~957 than RMI–SSL and using strings by factor ~830. It is however ~23% faster for instantiation. These numbers show that WS-Security, as supported by the products used in this test, shows poor performance and is only of limited use in real-world scenarios. WS-Security should be used only where security has the highest priority over performance and other approaches such as RMI–SSL cannot be used e.g. due to interoperability.

4.2. Performance on Linux

To investigate the performance difference between operating systems we have repeated the performance measurements on Linux. The results are shown on Figs. 4 and 5. We can see that the general picture on Linux has not changed. RMI is faster than RMI–SSL, Web services are an order of magnitude slower than RMI, while WS-Security is considerably slower. Using basic
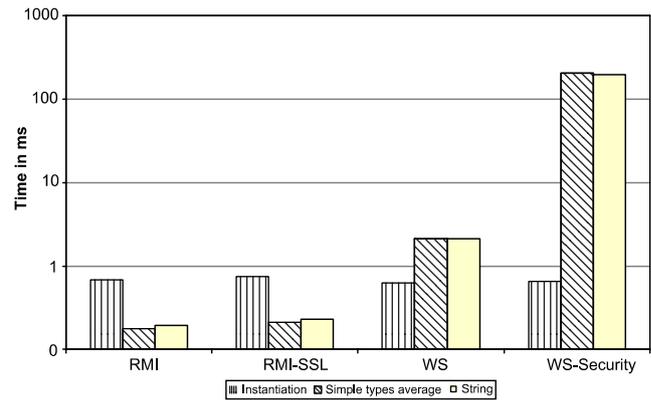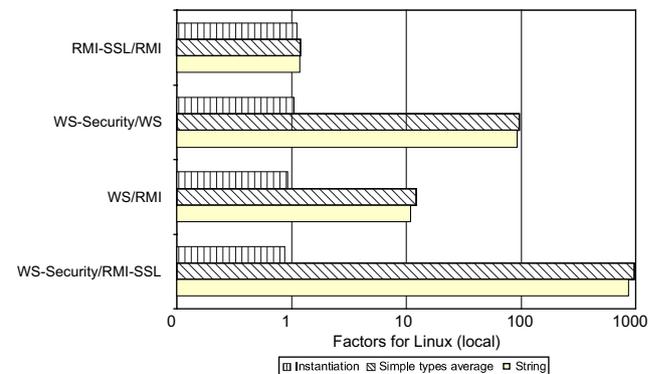

Fig. 4. Local performance results on Linux.

Fig. 3. Comparison of JCE security providers on Windows and Linux.

Fig. 5. Local performance factors on Linux.

data types Web services are on average ~12.1 times slower than RMI, using strings they are ~11 times slower. By the instantiation Web services are ~8% faster than RMI. RMI–SSL is on average only ~20% slower than unsecured RMI. Using basic data types the slowdown is ~20.5%, using strings ~17.7%. The instantiation is ~10% slower.

Comparing WS-Security to unsecured Web services shows that it is ~96 times slower using simple types, ~92 times using strings, and almost no difference by instantiation. On Fig. 3 we have also shown the comparison of the three Linux JCE security providers: Bouncy Castle Crypto version 1.24, Wedgetail JCSI Provider version 2.3, and OpenSource HBCI Toolkit for Java version 0.0.6. Here the differences between them are even smaller than on Windows. JCSI is ~5% faster than the other two using simple types, using strings JCSI is ~5% faster than BC, which is ~5% faster than HBCI. Comparing WS-Security to RMI–SSL shows that WS-Security is slower using simple data types by factor of ~969 than RMI–SSL and using strings by factor of ~860. It is however ~13% faster by instantiation.

4.3. Windows to Linux comparison

A comparison of local performance results between Windows and Linux is shown on Fig. 6. We can see that the local performance on Linux is always considerably better than on Windows. RMI is faster by ~6.6% for instantiation, ~11.8% using simple types and ~12.7% using strings. RMI–SSL is faster by ~15.2% for instantiation, ~30.5% using simple types and ~32.9% using strings. Web services are faster by ~6.9% for instantiation, ~12.9% using simple types and ~13.0% using strings. WS-Security is faster by ~2.4% for instantiation, ~28.8% using simple types and ~28.3% using strings. The analysis has shown that nether RMI or RMI–SSL nor Web services or WS-Security perform local optimizations. Therefore from the results we can
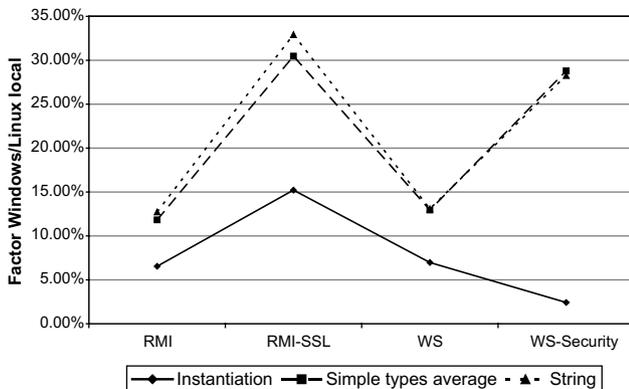
conclude that Linux better optimizes local network calls than Windows.

5. Performance analysis on networked computers

5.1. Performance on Windows

In real-world scenarios distributed applications are usually deployed on network-connected computers. To include network overhead, we have repeated the tests on two network-connected computers. The server-side functionality was installed on one computer and the client-side functionality on the other computer. From Fig. 7, we can see that RMI still performs much faster than the other alternatives. Web services are ~9.6 times slower than RMI for simple types and strings and ~14% slower for instantiation. RMI–SSL is ~25% slower than RMI for simple types, strings, and instantiation. The huge difference between WS-Security and Web services still exists and is even larger than in local scenario. WS-Security is ~124 times slower using simple types, and ~116 times slower using strings. By the instantiation WS-Security is only ~3% slower. WS-Security is compared to RMI–SSL ~900 times slower. This is
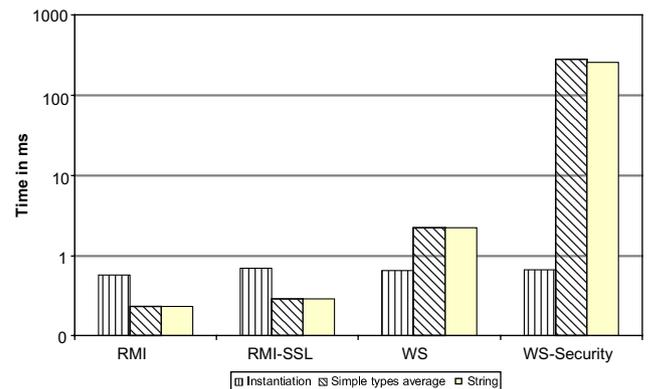

Fig. 7. Remote performance results on Windows.

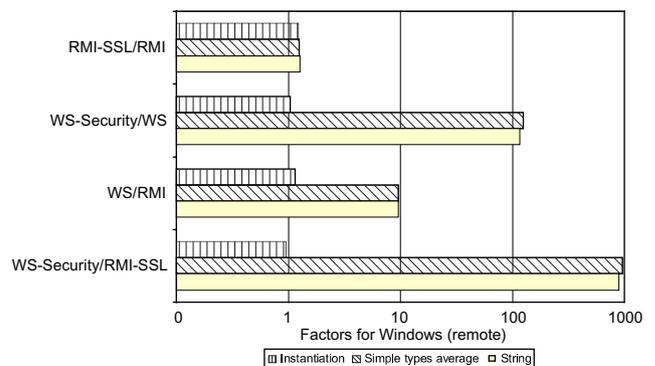Fig. 6. Local performance comparison between Windows and Linux.

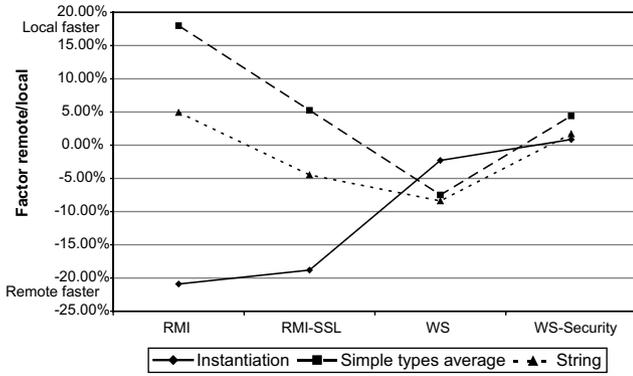Fig. 8. Remote performance factors on Windows.

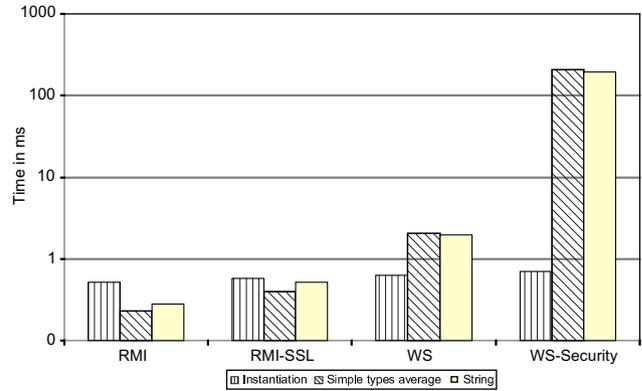Fig. 9. Comparison of remote vs. local performance results on Windows.

Fig. 10. Remote performance results on Linux.

shown on Fig. 8. We have also measured the difference between the three JCE providers, but have figured out that they performed almost identical to the local scenario and have not shown the results again.

Performance impact of remote network connections on Windows is relatively small and is shown on Fig. 9. Using RMI it adds ∼18% for simple types and ∼5% for strings. The instantiation is ∼21% faster. RMI–SSL is in remote configuration only ∼5% slower using simple types, ∼4% faster using strings, and ∼19% faster for instantiation. Web services are faster is all three cases: ∼7% for basic types, ∼8% for strings, and ∼2% for instantiation. WS-Security is slightly slower: ∼4% for basic types, ∼2% for strings, ∼1% for instantiation. The reason for such results is the load distribution. The CPU utilization for RMI–SSL is higher than for RMI, therefore the additional network overhead is lower than the improvement of using two CPUs. This is the case for Web services too where the web container places additional load on the server side. For WS-Security however the large message size which is transferred over the network is responsible for a slight slowdown in remote scenario. We have analyzed the message size in Section 6.

5.2. Performance on Linux

The performance results using two networked computers with Linux operating system are shown in Fig. 10. Web services are ∼8.8 times slower than RMI for simple types, ∼6.9 times for strings and ∼21% slower for instantiation. RMI–SSL is ∼70% slower than RMI for simple types, ∼82% for strings, and ∼23% for instantiation. WS-Security is ∼101 times slower using simple types, and ∼97 times slower using strings. Instantiation times of WS-Security are ∼14% slower. WS-Security on Linux is compared to RMI–SSL ∼520 times slower for simple types and ∼370 times for strings. This is shown on Fig. 11. We have also measured the difference between the three JCE providers, but have figured out that they performed almost identical to the local scenario and have not shown the results again.
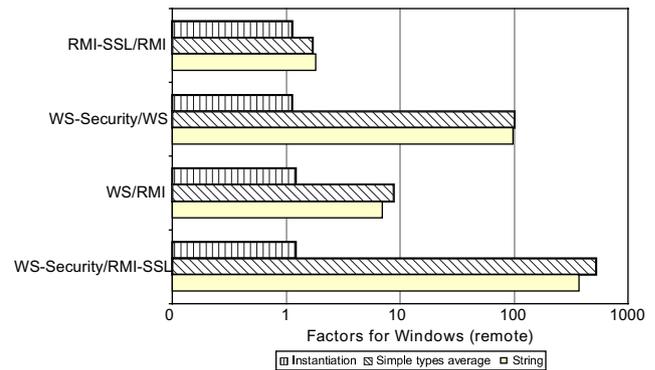

Fig. 11. Remote performance factors on Linux.

The performance impact of network connections on Linux differs considerably from Windows and is particularly important when using RMI–SSL. As shown on Fig. 12 remote RMI is slower than local for ∼32% for simple types and ∼46% for strings. The instantiation is ∼24% faster. RMI–SSL is in remote configuration ∼87% slower using simple types, ∼125% slower using strings, and ∼22% faster for instantiation. Web services are faster in all three cases: ∼4% for basic types, ∼8% for strings, while instantiation is almost identical. WS-Security is almost identical for simple types, ∼3% faster
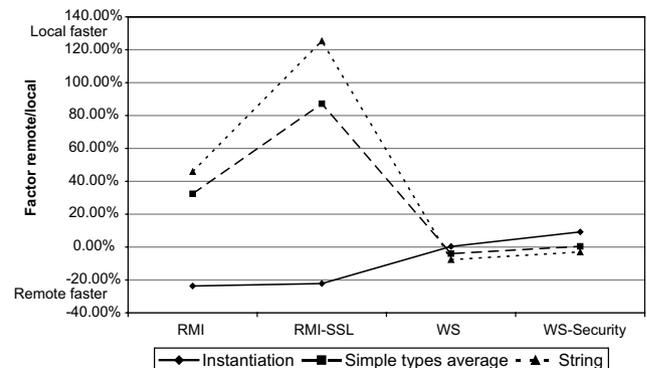

Fig. 12. Comparison of remote vs. local performance results on Linux.

for strings, while instantiation is ∼9% slower. In contrast to Windows Linux performs much more efficient local optimizations. This is why in most cases Linux remote performance is considerably slower than local. This is especially true for RMI and RMI–SSL. For Web services and WS-Security the Web server on the server-side adds considerably to the CPU utilization. Therefore, the difference between Web services and WS-Security for local and remote scenario is smaller.

5.3. Windows to Linux comparison

On Fig. 13, we have shown the comparison of results in remote networked scenario between Windows and Linux. In contrast to the local scenario where Linux has constantly outperformed Windows, in remote scenario Windows are faster for basic types and strings using RMI and RMI–SSL. In other scenarios Linux is faster. RMI is faster on Linux for instantiation by ∼10%, almost identical on simple types and slower using strings by ∼19% than Windows. RMI–SSL is on Linux faster for instantiation by ∼20% and slower using simple types by ∼27% and strings by ∼44%. Web services are faster on Linux: ∼4% by instantiation, ∼9% using simple types and ∼12% using strings. WS-Security is also faster on Linux by ∼34% using simple types and ∼35% using strings. It is ∼5.5% slower for instantiation. The reasons for differences are related to the implementation of network and threading in both operating systems.

6. Analysis and interpretation of results

The major difference between RMI and Web services from the performance perspective is related to the message protocol used for communication between distributed objects/services. RMI uses a binary protocol with binary encoding and makes use of the Java object serialization for call marshalling and returning the data. Web services on the other hand use SOAP which is a XML-based protocol. Instead of binary SOAP uses textual encoding of data with the added metadata. The metadata can be classified into:

- Start and end tag names, used to denote elements.
- Attributes containing the XML Schema types (xsi:type).
- XML-related headers.

While the size of the RMI binary messages is related to the actual binary size of the data, the size of SOAP messages is related to the lexical length of data, data type and variable names (which are used for tag names and for xsi:type attributes). For creating and reading of SOAP messages Web services use XML serialization. Hericko et al. (2003) have shown that binary serialization is an order of magnitude more efficient than XML serialization. The results in this research show the same. As we can see from Fig. 14, Web services SOAP messages were on average ∼4.3 times larger than RMI JRMP messages. When transferred over HTTP additional header information is added which further increases the message size. The problems with large size of SOAP messages can be partially solved using open source protocols such as Burlap (Caucho, 2004) or Hessian (Caucho, 2005), but at the cost of interoperability, which is rarely an option. Burlap uses a restricted subset of XML to reduce the size and improve the processing performance of messages. Hessian is a self-describing and portable binary protocol for connecting Web services. Burlap and Hessian are incompatible with standard Web services.

There are also several differences related to achieving authentication and secure communication. In our tests RMI–SSL and WS-Security used digital certificates to authenticate the client and the server side. To encrypt the communication we have used Triple-DES in both scenarios. Generally Triple-DES does not increase the message size, however when used with SOAP it can influence the message size, because it uses textual encoding. In RMI–SSL after the initial authentication binary
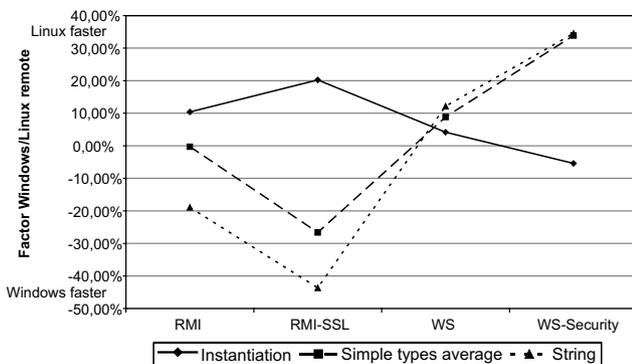

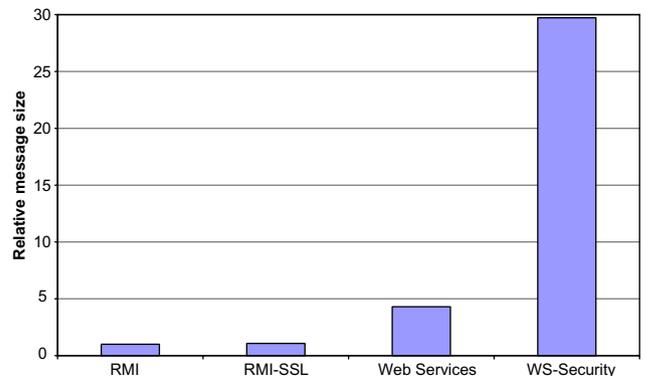Fig. 13. Remote performance comparison between Windows and Linux.

Fig. 14. Relative average message sizes comparison.

JRMP messages are encrypted on the sender side and decrypted on the receiver side. The size of the messages is comparable (Fig. 14) to the unsecured RMI; the small difference (RMI–SSL messages are ∼8% larger than RMI) is related to the initial overhead of the authentication, and the block size used with Triple-DES encryption.

In addition to the client and server authentication WS-Security offers message-level security which requires that binary security token, token reference, signature cipher data, and encrypted chipper data are included with each message (Oasis, 2004). In addition WS-Security time-stamps each message. To digitally sign data WS-Security uses XML Digital Signature; for encryption it uses XML Encryption. The encrypted data is base64-encoded to retain the textual format of SOAP messages, which increases the size of messages. As we can see from Fig. 14, SOAP messages when using WS-Security are on average ∼6.9 times larger than unsecured SOAP messages and do not differ considerably between different data types. Additional overhead of WS-Security is related to the message-level security, which requires verification of the digital signature of each message. While SOAP message-level security provides additional flexibility such as the possibility to encrypt parts of the message with different keys (this is currently not supported in JWSDP) it also places a large performance overhead compared to RMI–SSL. Comparing WS-Security message size to RMI–SSL shows that WS-Security messages are ∼27.5 times larger.

6.1. Bottleneck identification

To investigate the relation of message sizes, performance, and the possible bottlenecks, we have profiled the test scenarios and identified the methods and packages in which the majority of execution time has been spent. Then we have calculated the percentage of time and compared the methods between the scenarios. We have used Borland OptimizeIt Enterprise Suite 6 profiler and Enerjy Performance Profiler.

From the results in Table 2, we can see that a considerable amount of total round trip time in Web services

and WS-Security is related to XML serialization and deserialization. For RMI and RMI–SSL, which use binary serialization, the amount of time related to binary serialization and deserialization is lower. This confirms our previous results (Juric et al., 2004) and confirms that binary serialization, which produces smaller stream sizes, requires fewer resources for parsing and demarshalling of streams as XML serialization. XML serialization has to perform more string parsing, string copying, comparison and data type casts than binary serialization. RMI has thus the advantage of smaller message sizes and faster processing, marshalling and dispatching. The performance overhead of Web services is ∼9 times larger than RMI, which is twice as much as is the difference in message sizes (∼4.3 times). The tests in this research have been limited to basic data types and strings. We believe that with larger payload exchanged between the client and the server the advantage of RMI would be even more evident.

In addition to Web services and WS-Security deficits related to SOAP protocol, parsing, marshalling and string manipulation, implementation related deficits influence the performance (including Java platform related and operating system related). Additional code analysis and profiling of JWSDP have shown that there are several bottlenecks present, particularly: repeated authentication and acquisition of keys from key-storage, no results buffering, unnecessary data copying, unoptimized range checking, repeated local method invocations, large methods with several branches which negatively influence the performance, and repeated computation of invariant values. The tested version of JWSDP has been the first version that supported WS-Security therefore we can expect performance improvements in the future versions. The operating system related performance factors include the network subsystem and the integration with the Java virtual machine. We have seen that for local scenarios Linux provides much better local optimizations, which considerably improve performance over Windows.

6.2. Impact on application design and implementation

The identified differences in functionality and performance should be considered when designing distributed applications. Distributed models hide the complexity of remote communication from the developer. The fact that remote method/service invocation is considerably slower than local should be considered in the design phase already. We can influence the number of remote method invocations (incoming, outgoing), interface design, amount of data transferred over the wire, message structures and their processing overhead, state management, and location of objects/services, serialization performance, message processing and encoding, etc. Performance also depends on the distributed model

Table 2
Comparison of relative time consumption

Package (%)	RMI	RMI–SSL	WS	WS-Sec
java.net.socket	27.86	32.16	17.02	39.76
sun.rmi.transport	19.58	21.76	N/A	N/A
sun.net.www	<0.10	<0.10	5.07	8.16
sun.nio.cs	<0.10	<0.10	1.93	1.15
java.io.ObjectInputStream/ ObjectOutputStream	11.87	12.48	<0.10	<0.10
com.sun.xml.rpc	N/A	N/A	27.79	36.33
sun.rmi.server	3.36	4.15	N/A	N/A
java.lang.SecurityManager	<0.10	1.17	N/A	N/A
sun.security.pkcs	N/A	2.90	N/A	1.34

implementation and the selection of platform and hardware.

The primary design rule is to minimize the number of remote invocations. This is achieved using coarse-grained interfaces, which reduce the number of remote invocations. They however increase the size of parameter and return data, which should be packed in value objects and structures. In RMI and RMI–SSL value objects use binary serialization when transferred over the network. To improve the performance we should design value objects with only the necessary state (set of attributes and relations). Special attention has to be placed on relations which heavily influence the serialization performance (Adatia et al., 2001). When serializing a value object the whole graph of related objects gets serialized. Java provides ways to control serialization. We can mark attributes as transient and they will not be included in serialization. This is particularly useful for derived attributes. We can also write custom methods for reading and writing the state of the objects. For complex objects such custom methods can be more efficient than default serialization. Serialization in Java is related to the dynamic class downloading, which is used to transfer the object behavior (code) to the remote computers. Pre-installing the necessary classes on all involved computers thus improves the performance.

If using Web services or WS-Security we should use coarse-grained interfaces too and reduce the number of remote invocations. Instead of value objects we use complex XML types to transfer parameters and return values. Our measurements have shown that XML processing is an important overhead factor therefore to improve the performance we can simplify the schemas. We should use short tag names and unaligned XML. Attention should be paid to the XML parsing techniques where SAX often shows performance benefits compared to DOM. In Java we should also be careful selecting the appropriate JAXP engine. If we have several requests/responses sent to the same service we should use persistent connections.

Distributed objects and services are often used as middle-tier components and contain business logic. A large performance penalty of distributed designs is related to input validation. If input validation is done on the middle-tier this requires several remote method invocations, which reduce the responsiveness of the user interface. Moving the input validation on the client thus improves performance. In RMI value objects with methods for input validation can be used for this purpose. Value objects can also contain pre-fetched and cached data form the persistence tier to further speed-up the validation. In the later approach, we however have to introduce the necessary synchronization mechanisms. With Web services we can use complex types instead of value objects for pre-fetched and cached data form

the persistence tier. We should deploy the input validation code to the client-side in advance.

Number of remote method invocations can be further reduced using the façade pattern (Gamma et al., 1995) and the collocation (deployment on the same physical computer) of distributed objects/services. A distributed façade containing the control logic can invoke the related objects/services locally and thus reduce the remote invocation overhead. Here generally two choices exist: we can make use of the local optimizations provided by distributed models and operating systems; or we can design objects/services with local interfaces. The first approach is more flexible as it allows easier modifications of physical architecture, but depends on the local optimizations. Our measurements have shown that only the performance of RMI and RMI–SSL is better in local scenarios, and that important differences exist between operating systems (Windows and Linux). Relying on local optimizations is thus in most cases not a good choice. Therefore, to take advantage of distributed façades we should carefully plan the architecture in advance and make the decisions which components should have local and which remote interfaces already in the design phase.

Particular attention should be placed on the selection of operations. Here we should decide whether it is more appropriate to use synchronous or asynchronous and one-way or two-way operations. Most applications that require reliable communication will most likely use two-way operations. RMI provides support for two-way synchronous operations only while with Web services we can use one-way and two-way, synchronous and asynchronous operations. Using one-way operations can improve the performance but this is a tradeoff with reliability and is hardly a choice. The selection between synchronous and asynchronous operations will depend on the type of processing done in the operations and the requirement for the immediate response. Long-lasting operations and operations where clients do not require immediate responses are best modeled as asynchronous. Asynchronous operations are often used with callbacks to notify the client about the end of an operation and submit results. Callbacks in Web services are related with the additional effort where clients have to be able to expose ports (endpoint references). Depending on the implementation technology, this requires a Web server or a HTTP stack on the client. With JWSDP it is necessary to install a Web server on the client.

Web services are stateless services. This requires clients to include id for each operation invocation. To improve performance the ids should be compact. We should avoid using composed ids if possible. When using Web services over Internet protocols such as HTTP, we should also be aware that requests can arrive out of order, can be lost, or can arrive more than once. There-

fore, we should design the operations of Web services as idempotent.

When we require secure communication, we should use RMI–SSL if possible. If we require WS-Security we should limit the number of remote invocations as much as possible. We should also identify the data and operations where security is necessary and do not use it all over. To improve the performance we can use plain XML Encryption instead. This will limit inter-operability however.

7. Conclusion

In this article, we have done a functional and performance analysis and comparison of Web services and RMI. We have analyzed regular (unsecured) as well as secured variants, WS-Security and RMI–SSL. We have identified and described the major functional differences and figured out that RMI is suitable for distributed applications, which require synchronous remote method invocations only, make use of stateful objects, object references, dynamic class downloading, distributed garbage collection, and remote object activation. Web services on the other hand are suitable for synchronous as well as asynchronous operation invocations, provide interoperability with other platforms and environments, and are better suitable for dynamic service binding and communication through firewalls. Differences also exist between secured versions. RMI–SSL offers point-to-point security while WS-Security offers message-level security. Developing distributed applications using JAX-RPC further masks the differences between RMI and Web services.

To identify the differences in performance we have done an in-depth performance analysis on Windows and Linux and have included local and remote scenarios. The measurements have shown that RMI has been an order of magnitude faster than Web services in all scenarios. Web services have been on average $\sim$9 times slower than RMI. RMI–SSL has been on average $\sim$40% slower than RMI. WS-Security on the other hand has been considerably slower than Web services, on average more than 100 times. Selection of different JCE security providers has influenced the performance only marginally, within $\sim$6%. The comparison of performance on Windows and Linux has shown that Linux has provided better results in local scenarios by $\sim$10% to $\sim$30%. In remote scenarios Windows has provided faster performance for RMI (by $\sim$5%) and RMI–SSL (by $\sim$15%), while Linux has provided better performance for Web services (by $\sim$8%) and WS-Security (by almost 30%).

The reasons for slower performance of Web services and WS-Security compared to RMI and RMI–SSL can be found in message sizes transferred over the network, processing overhead of the messages, and implementation related overheads. The differences between binary JRMP and XML-based SOAP messages are considerable. Unsecured SOAP messages are on average $\sim$4.3 times larger than unsecured JRMP messages, secured SOAP messages are more than 27 times larger than secured JRMP. While RMI–SSL adds only a minor overhead ($\sim$8%) to the network traffic compared to RMI, WS-Security's overhead compared to Web services is considerable ($\sim$690%). The major portion of the overhead is related to the concept of the message-level security and to the textual encoding of the message content (payload). To identify the sources of overhead we have profiled the code.

The differences between Web services and RMI in functionality and performance influence the design and implementation of distributed applications. We have identified the most important design and implementation principles, which lead to performance improvements and/or prevent bottlenecks. These principles can be used to adapt the application design and accommodate the advantages and disadvantages of selected distributed model.

References

Adatia, R., Juric, M.B., Sarang, P.G., Gabhart, K., et al., 2001. Professional EJB. Wrox Press, Birmingham.

Caucho, 2004. Burlap Web Service Protocol, Version 2.1.12. Available from: <http://www.caucho.com/burlap/index.xtp>.

Caucho, 2005. Hessian Binary Web Service Protocol, Version 3.0.13. Available from: <http://www.caucho.com/hessian/>.

Gamma, E., Helm, R., Johnson, R., Vlissides, R., 1995. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley.

Hericko, M., Juric, M.B., Rozman, I., Beloglavec, S., Zivkovic, A., 2003. Object serialization analysis and comparison in Java and .NET. ACM SIGPLAN Notices 38 (8), 44–54.

Juric, M.B., Rozman, I., 2000. Java 2 RMI and IDL comparison. Java Report 5 (2), 36–48.

Juric, M.B., Hericko, M., Rozman, I., 2000a. Performance comparison of CORBA and RMI. Information and Software Technology Journal 42 (13), 915–933.

Juric, M.B., Rozman, I., Nash, S., 2000b. Java 2 distributed object middleware performance analysis and optimization. ACM SIGPLAN Notices 35 (8), 31–40.

Juric, M.B., Kezmah, B., Hericko, M., Rozman, I., Vezocnik, I., 2004. Java RMI, RMI tunneling and Web services comparison and performance analysis. ACM SIGPLAN Notices 39 (5), 58–65.

Juric, M.B., Zivkovic, A., Hericko, M., Brumen, B., Welzer, T., Rozman, I., 1999. Performance assessment framework for distributed object architectures. ADBIS'99, LNCS 1691. Springer, pp. 349–366.

Kono, K., Masuda, T., 2000. Efficient RMI: Dynamic specialization of object serialization. In: International Conference on Distributed Computing Systems. IEEE Computer Society Press, pp. 308–316.

Maassen, J., Nieuwpoort, R., Veldema, R., Bal, H., Plaat, A., 1999. An efficient implementation of Java's remote method invocation. In: ACM SIGPLAN symposium on Principles and practice of parallel programming. ACM Press, pp. 173–182.

Nester, C., Philippsen, M., Haumacher, B., 1999. A more efficient RMI for Java. In: ACM Conference on Java Grande. ACM Press, pp. 152–159.

Oasis, 2004. Web Services Security: SOAP Message Security 1.0 (WS-Security). Available from: <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>.

Rivera, L., Zhang, L., Sampemane, G., Krishnamurthy, S., 1997. HP-RMI: High performance Java RMI over FM. Available from: <www-csag.ucsd.edu/individual/achien/cs491-f97/projects/hprmi_paper.ps>.

SourceForge, 2004. Transparent RMI, Version 0.1.4. Available from: <http://sourceforge.net/projects/trmi/>.

SourceForge, 2005. Mantaray, Version 1.8. Available from: <http://sourceforge.net/projects/mantaray/>.

Sun Microsystems, 1998. Introduction to SSL. Available from: <http://docs.sun.com/source/816-6156-10/contents.htm>.

Sun Microsystems, 2003. Java Remote Method Invocation Specification 1.4. Available from: <ftp://ftp.java.sun.com/docs/j2se1.4/rmi-spec-1.4.pdf>.

Sun Microsystems, 2004. Java API for XML-Based RPC (JAX-RPC) Specification 1.1. Available from: <http://java.sun.com/xml/downloads/jaxrpc.html>.

Vijaykumar, K., Walther, D., Bhola, S., Bommaiah, E., Riley, G., Topol, B., Ahamad, M., 1998. Efficient implementation of Java remote method invocation (RMI). In: 4th USENIX Conference on Object-Oriented Technologies and Systems, Usenix, New Mexico. pp. 144–148.

W3C, World Wide Web Consortium, 2001. Web Services Description Language, WSDL, W3C Note. Available from: <http://www.w3.org/TR/wsdl>.

W3C, World Wide Web Consortium, 2003. SOAP Version 1.2, W3C Recommendation. Available from: <http://www.w3.org/TR/soap/>.

Zeadally, S., Zahang, L., Zhu, Z., Lu, J., 2004. Network APIs performance on commodity operating systems. Information and Software Technology Journal 46 (6), 397–402.

Zhuge, H., Liu, J., 2004. Flexible Retrieval of Web Services. Journal of Systems and Software 70 (1–2), 107–116.

Matjaz B. Juric, Ph.D., is an associate professor at the University of Maribor. He has authored or co-authored the following books: Business Process Execution Language, Professional J2EE EAI, Professional EJB, J2EE Design Patterns Applied, and .NET Serialization Handbook. He has also published chapters in books and several articles in journals.

Ivan Rozman, Ph.D., is the rector of University of Maribor and head of the Institute of Informatics. His research interests include engineering management, software development methodologies, software process, and software quality. He is the leader of COT—Object Technology Centre.

Bostjan Brumen, Ph.D., is a researcher at the University of Maribor. His research interests are focused on data mining, data classification, and data security. He has published on several conferences and journals.

Matjaz Colnaric, Ph.D., is a professor at the University of Maribor. His research interests include embedded hard real-time systems, microprocessors and data structures. He is a reviewer for journals such as IEEE Parallel and Distributed Technology, International Journal on Real-Time Systems, Euromicro Journal, Computer Journal, etc.

Marjan Hericko, Ph.D., is an associate professor at the University of Maribor. His research interests include object technology with emphasis on development methods and tools, quality assurance and object oriented metrics. He has published on several international conferences and journals. He is program committee head of the OTS conference and the technical coordinator of COT.