Java RMI and .Net Remoting Performance Comparison

Willem Elbers, Frank Koopmans and Ken Madlener
Radboud Universiteit Nijmegen

December 2004

Abstract

Java and .Net are both widely used for creating Middleware solutions. There are many interesting aspects which can be compared between Java and .Net. In this paper we examine the architecture of Java RMI and .Net Remoting and test their performance as they come "out-of-the-box". The performance was measured over a series of 3 different tests using .Net and Java's high performance timers and our own modules to measure CPU utilization and Memory usage. In all our tests Java RMI had the best performance times.

1 Introduction

The creation of software has been evolving strongly over the past few years. High level programming languages are increasingly popular amongst software developers because there is a great need for software quality and lower development times, while not compromising on performance. Also, the Internet has become increasingly popular and the need for networking applications is greater then ever.

Java and .Net are both widely used for creating Middleware solutions. There are many interesting aspects which can be compared between Java and .Net, this paper focuses specifically on Middleware performance.

First there will be an overview of the Java and .Net middleware implementations, Java RMI and .Net Remoting. These will be described on an abstract level which will help understanding and explaining the results of the performance measurement later on. After the introduction of Java and .Net middleware, their performance measurement will be introduced. An overview of which measurements were done, and why, followed by the results of these measurements will reveal the performance aspects of Java and .Net middleware.

By looking back at the architecture of both Java and .Net Middleware, conclusions can be drawn about the consequences for performance.

2 Java and .Net middleware explained

There are many differences and similarities between Java RMI and .Net Remoting. This chapter introduces the general architecture of both and a few differences which are relevant to the performance aspect of middleware.

2.1 .Net Remoting and Java RMI in general

.Net Remoting and Java RMI are mechanisms which allow the user to invoke methods in another address space. The other address space could be on the same machine or a different one. These two mechanisms can be regarded as RPC in an object-oriented fashion.

- When the client object calls a method on the server object, a proxy passes the call information to the RPC system on the client. This system in turn sends the call over the channel to the Remoting system on the server.

- The RPC system on the server receives the call information and, on the basis of it, invokes the method on the actual object on the server (creating the object if necessary).
- The RPC system on the server collects the result of the method invocation and passes it back to the client through the connection of the RPC system on of the client.
- The RPC system at the client receives the server's response and returns the results to the client object through the proxy.

Distributed objects Java and .Net use distributed objects, these are objects whose instances can be used remotely, they can be referenced in two different ways:

- Within the address space where the object was constructed; The object is an ordinary object which can be used like any other object.
- Within other address spaces (remote); The object can be referenced using an object handle, and is treated like a remote interface. For the most part object handles can be used in the same way as an ordinary object.

Java and .Net distribute their objects in the same way, but the terminology is different. .Net named their remote objects Remote Interfaces while Java mostly speaks of Remote Classes.

2.2 .Net Remoting and Java RMI differences

The following differences between .Net Remoting and Java RMI affect their performance in our tests:

Distributed directory Java RMI uses a RMI Registry as their distributed directory, RMI clients and servers register their "address" in the RMI Registry. In .Net Remoting a Windows native register is used, hence there is no distributed directory within the .Net framework.

Available protocols .Net Remoting can communicate using HTTP, TCP or SOAP. Java RMI is able to use All URL protocols (HTTP, FTP, etc.), IIOP, JRMP and ORMI. These different protocols serve different means, HTTP is used by .Net and Java (as standard) for compatibility and raw TCP-sockets are used for optimized speed. Java has a better choice in protocols for RMI, resulting in greater compatibility with other applications.

Proxies Another important difference between .Net Remoting and Java RMI is that the proxies in Remoting are created during runtime whereas RMI creates it's stubs during compile-time.

Boxing This subject is not directly related to the middleware mechanisms but affects the performance of .Net. While Java's type system is almost completely object oriented, it still uses some primitive types such as int and char. In .Net there are no primitive types. Through a built-in mechanism called "Boxing" all types can be considered object and thus making it a fully object oriented type system.

3 Java and .Net middleware performance measurement

3.1 Measurement specification

The goal of the measurements is to accurately compare the middleware performance between Java RMI and .Net Remoting. We do not want to measure any other performance differences between Java and .Net.

The three most interesting aspects for middleware performance are:

- Remote method invocation with little data
- Remote method invocation with large data
- Exchange an array of objects between client and server while both remove one object and send the array to the other.

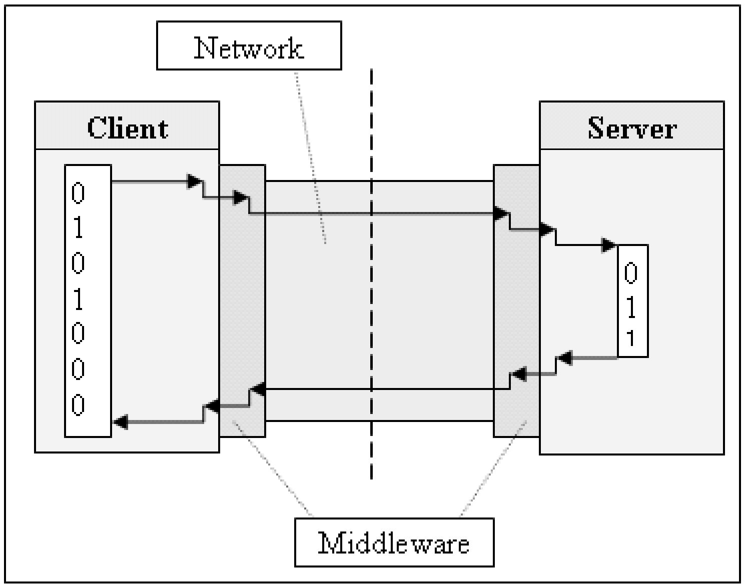
Repeatedly invoking a remote method with little data (as parameter) will reveal how the middleware handles a "plain" method invocation.

This is often used in practice. Repeatedly invoking a remote method with large data (as parameter) will reveal how the middleware handles the transportation and buffering of data. The last measurement reveals how the middleware will handle a variation of the transport data size in combination with the disposal of garbage objects. A serialized object containing an array of dummy objects is sent from client to server. The server removes one object from the array and sends the serialized object back to the client. The client also removes an object from the array, and sends the serialized object to the server. This is repeatedly done until the array is empty.

The first and second measurement cannot be predicted very well. There is no good reason on forehand to believe Java or .Net is faster. Especially in the data invocation measurement we expect both Remoting and RMI to be equally as fast. Where the socket implementations are slightly faster, because sockets won't have any middleware overhead such as marshalling and unmarshalling. For the third measurement, we expect Java to be faster because. In a Java RMI environment the client and server share the virtual machine. This is making the disposal of garbage and exchanging of data fast.

3.2 Measurement criteria

We will use a client-server situation for implementing middleware. Figure 1 illustrates how the client and server programs communicate.

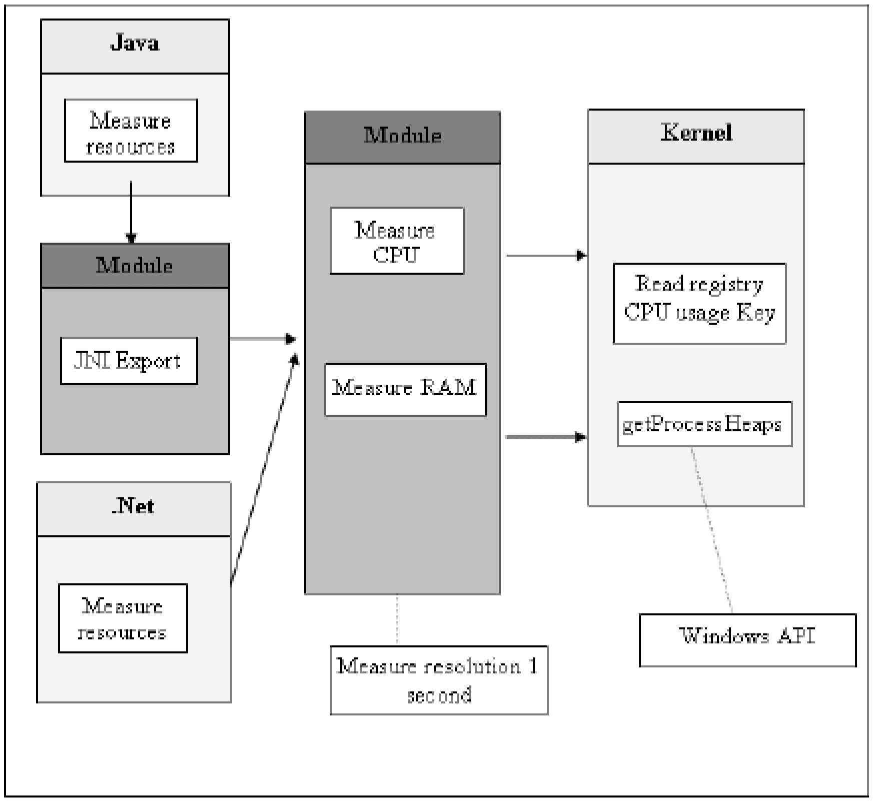


To make sure that only the middleware performance is being measured, the measurements are implemented as ”clean” as possible. The execution time of the program code is as minimal as possible, making its influence on the measurement neglectable.

It is important to note that we are trying to compare two different platforms. This means the Timers used to measure intervals and durations are (most likely) of a different resolution. Usage of a (not by Sun supported) high-performance timer in Java, we achieved 10 nano-second-accurate timing. The default timing in Java was only 10 milliseconds accurate. The timer which is normally used in .Net has an accuracy of 100 nanoseconds, according to the specifications. Unfortunately, the implementation is not that accurate and in practice it is about 15 milliseconds accurate due to thread context switching. Using a special Windows native timer, we can achieve 10 nano-second-accurate timing in .Net too. The implementation of the middleware itself in Java and .Net are considered to be blackbox. During the measurement we do not alter or tweak it. Both Java RMI and .Net Remoting were implemented using HTTP as communication protocol. For both middleware implementations, this is the most common used technique.

3.3 Measurement implementation

To make a good and representative measurement we will use a separate computer system for both client and server. To reduce interference from other applications and unpredictable behavior, the client and server system are both provided with a new installation of Microsoft Windows. A null measurement is used to set our reference mark. The null measurement is implemented (on both Java and .Net) by calling an empty remote method using sockets. The usage of Sockets is considered to use only trivial networking components. Therefore this implementation of a null measurement will only measure the time it takes to send data through a network. All measurements will be corrected by the corresponding null measurements leaving an accurate reading of the middleware performance. To make a good measurement the measurements take at least 10 seconds. This rules out minor differences caused by unexpected circumstances and makes the timer inaccuracy neglectable. Three different kinds of middleware performance measurement are done. Each of these measurements is implemented in the same way in both Java and .Net. The measurements exchange data between the server and client using middleware, the time and resources the measurement takes are measured. In both Java and .Net there are no facilities to accurately measure the system resources used. We have created our own DLL using the C++ programming language which measures the system resources by calling Windows API's and a registry key to query the system resources. By making a uniform library for measuring system resources we can be sure that the system resources are measured in the same way for both Java and .Net. Without a uniform library there is a chance that Java and .Net measure the used CPU and RAM in a different way, this would cause false measurements.

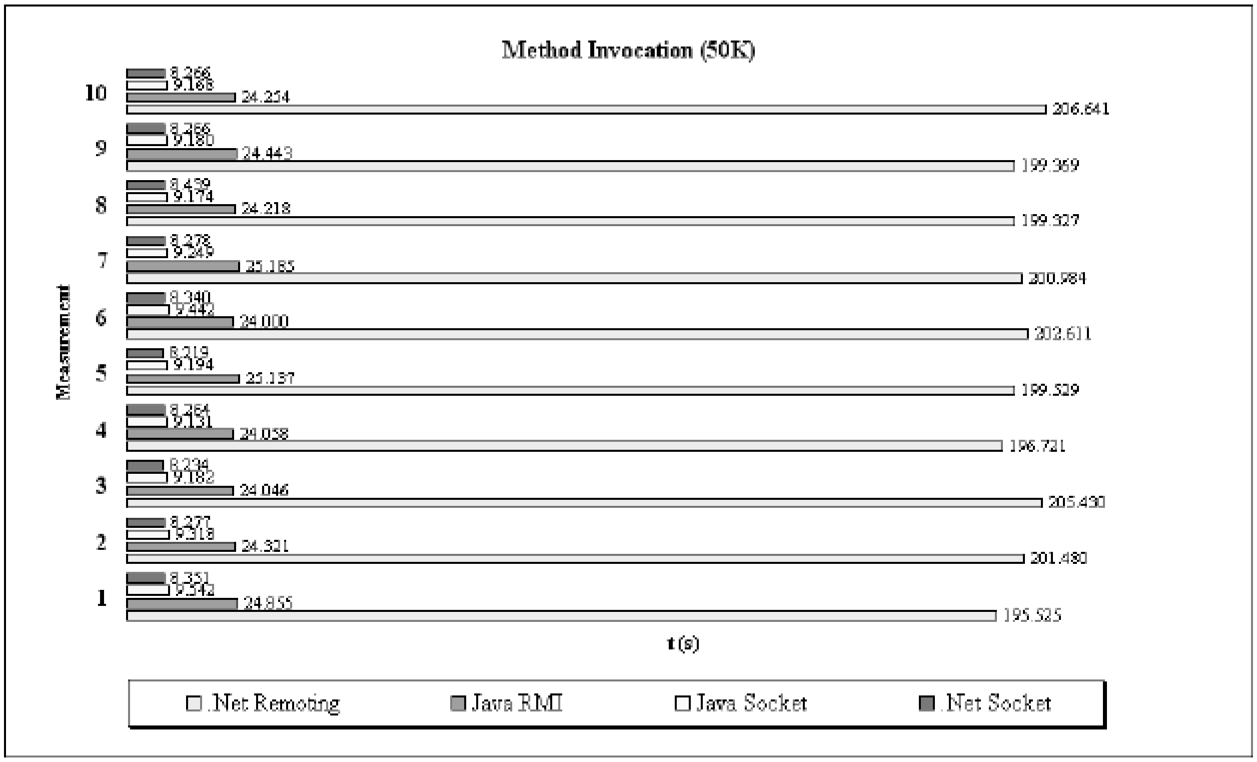


The measurement of system resources by the DLL has a resolution of 1 second. It takes a few milliseconds to call the DLL methods for measuring. The resolution cannot be higher because that would be a performance slowdown, it would intervene with the measurement results. The measurement of our performance is done in two places now, in Java/.Net a high-performance timer is called for measuring the time needed and in the DLL we measure the CPU and RAM usage. The reason that these measurements are not all done in the DLL is the big difference in measurement resolution between the timers and the resource-measurement. If we would implement a high-resolution timer in the DLL with a resolution of 10nanoseconds it would be called by the Java/.Net program after 5 milliseconds. This would cause severe inaccuracy. However, measuring the resource usage in the DLL is acceptable because the 5 milliseconds delay is relatively small to the resolution of 1 second.

4 Measurement results

Null measurement All measurements were performed locally on one machine to confirm our expectations of local performance. This was true for all measurements. All local measurements were performed within a millisecond. These were performed with the same amount of invocations and/or data.

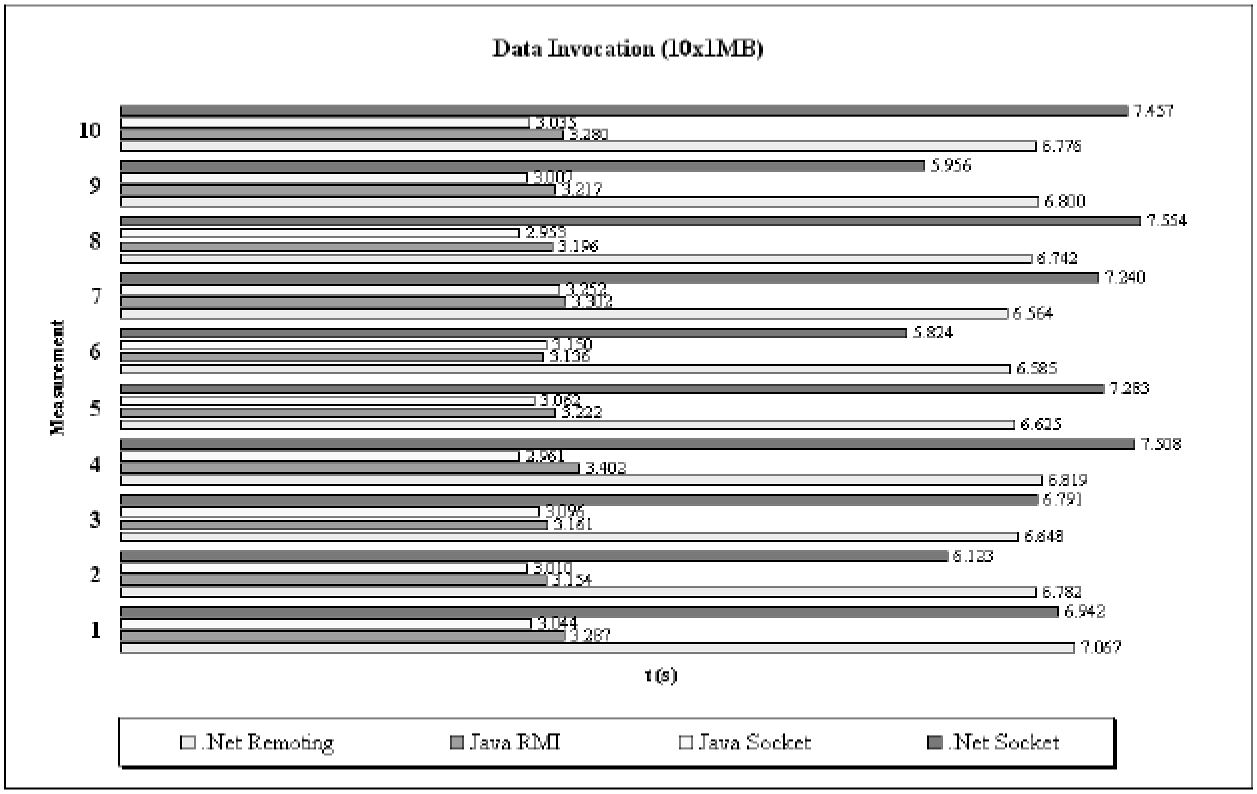
Method invocation This test consists of 50.000 remote method calls with just a single int parameter.



This graph reveals an unexpected result. The overall performance of .Net Remoting is a factor 7 worse than Java RMI. The resource graphs show us that there is no bottleneck at either CPU or Network utilization. A logical explanation to this unexpected behavior of Remoting could be the buffer polling rate or amount of roundtrips required to invoke a method. The boxing mechanism, as described in chapter 2, should also affect the performance in a negative way, although it is not very likely to perform this bad. To make sure both our test applications were working correctly we have added some debug code to show a message on the screen when the method was invoked by the server.

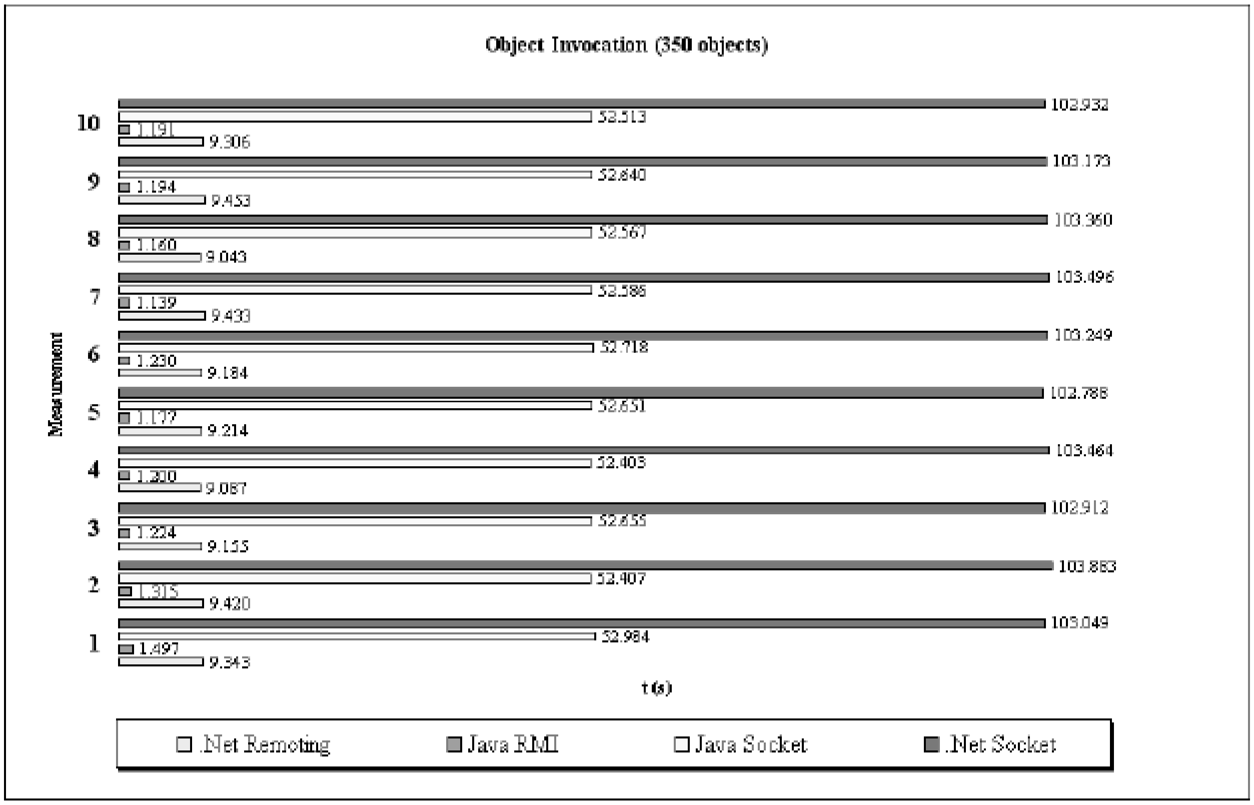
Data invocation This test consists of 10 remote method calls with a 1 megabyte char array parameter. In this test, again, we see an unexpected bad performance of .Net Remoting. In the network utilization graphs we see RMI peaking at 100% while Remoting does a maximum of about 50%.

The interval between these peaks however is the same for both.



During these measurements .Net Remoting had no remarkable amount of CPU Utilization. In the implementation of the .Net Socket a Network-Stream wrapper was used in order to get these large blocks of data transferred across the application domains. This probably has some severe consequences for the performance of this implementation, because a sequence of smaller blocks of data (10 megabyte divided into blocks of 8192 bytes, which is apparently the maximum of a byte array passed as a parameter to our remote method) outperforms the Java Socket.

Object invocation The object invocation test consists of ”bouncing” an array of 350 objects between the 2 application domains while emptying one element each time the client or server receives this array.



Here, RMI is a factor 7 faster than Remoting. From the results of this test we can conclude that .Net Remoting has problems when it comes to transferring relatively big objects between the application domains, as Microsoft warns about this issue in it's documents on .Net architecture. Increasing the size of the object the difference between .Net Remoting and Java RMI, the difference between these two mechanisms can go up to a factor of 20 in favor of RMI. We had expected the network traffic would decrease towards the end of the test, where the number of objects left in the array approaches zero. This is for both RMI and Remoting not the case. Halfway the test both RMI and Remoting even allocate some extra memory. Our own socket implementations of this test are much slower than both RMI and Remoting. This is not very strange, considering the fact the whole structure of each object in the array has to be transferred for each method invocation.

5 Conclusions

The most obvious and expected result of these measurements is that the middleware for both Java and .Net causes a performance decrease compared to local system calls. This is still true when disregarding the networking cost, because there is an extra layer of functionality within middleware.

RMI had the best performance in the 3 performed tests. It is up to 20 times faster than .Net Remoting on object invocation. This was more or less expected, as Microsoft warns for a bad performance in this specific test. The result was unexpected in the method invocation and data invocation tests. On method invocation RMI outperformed Remoting by a factor 7 and on data invocation by a factor 2.

In the following table we have summarized the results of our tests.

	Time		CPU		Memory	
	RMI	Remoting	RMI	Remoting	RMI	Remoting
Invocation	+	–	+	-	-	+
Data	+	-	+/-	+/-	-	+
Objects	+	-	+/-	+/-	-	+
Total	**+**	**-**	**+**	**-**	**-**	**+**

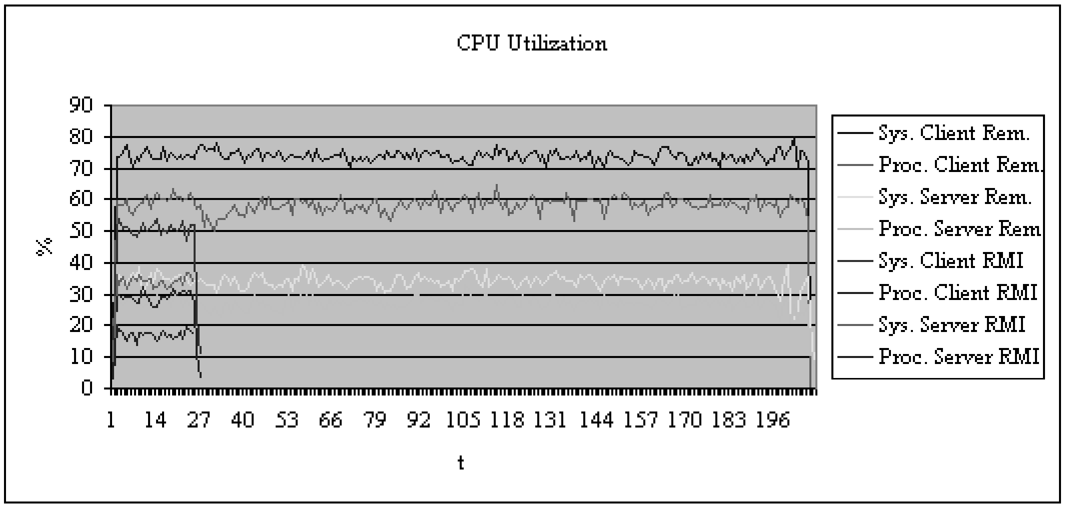
To measure the performances we have used the mechanisms ”out-of-the-box”, so we have probably not used it under the optimal configurations.

Some interesting aspects of this middleware performance comparison still remain uncovered in this paper. To achieve a greater understanding of the differences it would be useful to research the influence of marshalling and garbage collection. Also, the behavior of the tested middleware mechanisms under their optimal configurations and under a greater variation in size of the test sets is open to further research.

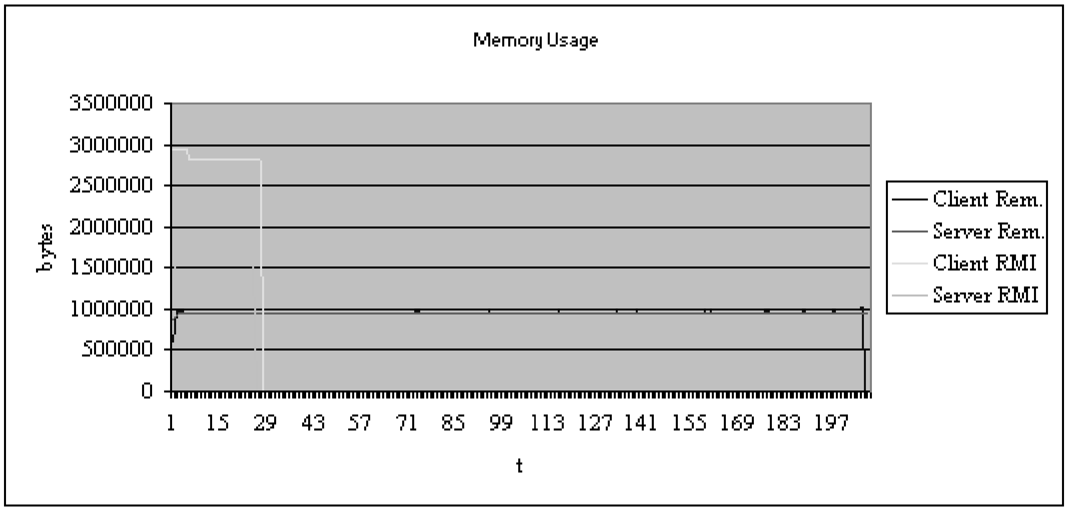
6 Appendix A

6.1 Graphs

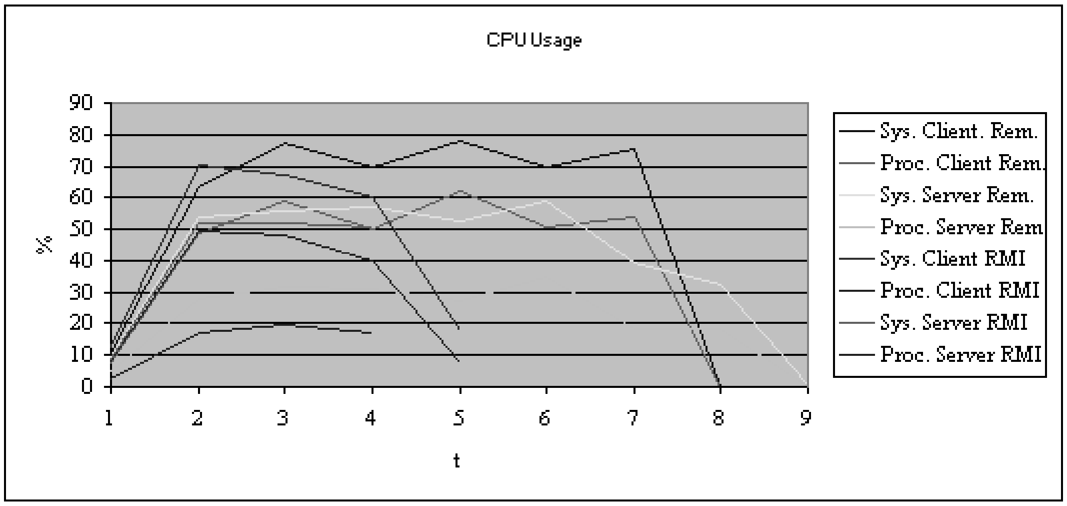
Method invocation - CPU utilization



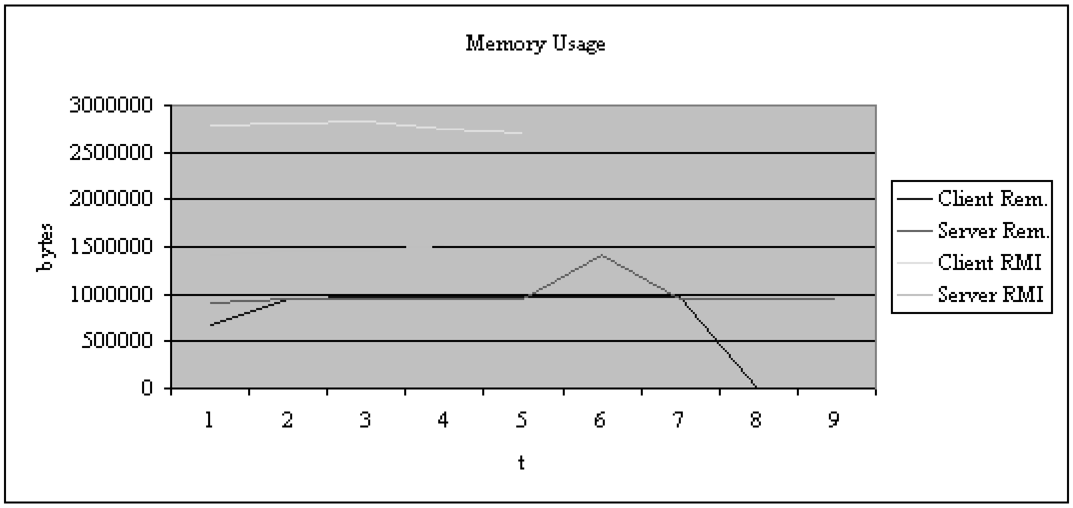
Method invocation - Memory usage



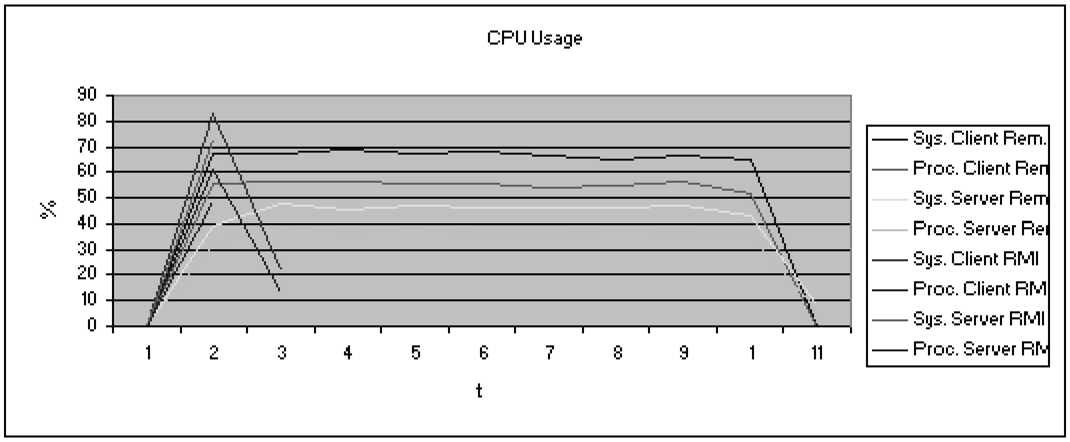
Data invocation - CPU Utilization



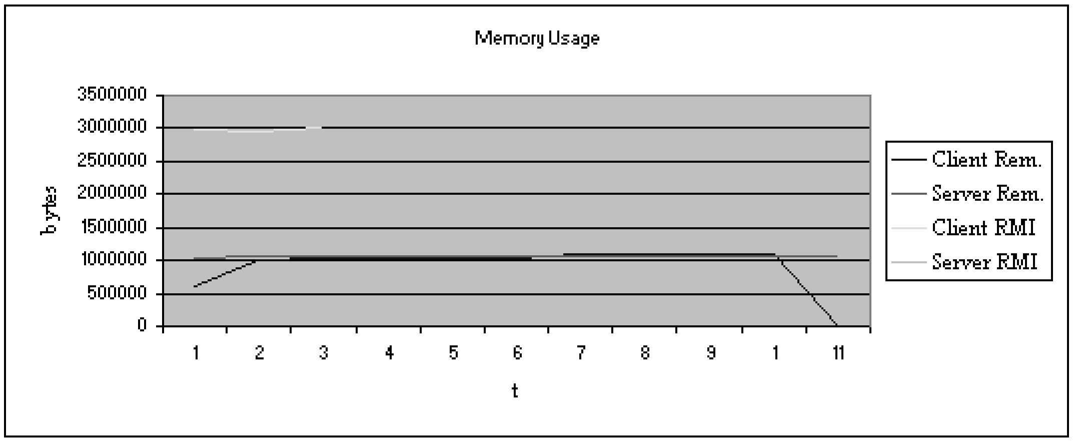
Data invocation - Memory usage



Object invocation - CPU utilization



Object invocation - Memory usage



7 Appendix B

7.1 Technical specifications

All relevant technical specifications used during this research are provided for reference and replay.

Hardware The systems used for both client and server are two Fujitsu Siemens Amilo-D-6800 Laptops with extra RAM. System specs:

- Intel Pentium 4 2,4MHz
- 768MB RAM
- 40GB Harddisk
- 100Mbit Intel LAN adapter

Software The operating system used is Microsoft Windows XP Professional. Both systems were formatted and provided with a new Windows installation and service-pack 1. For Java, virtual machine version 1.4.2_06 was used. For .Net, version 1.1 was used.

References

[1] Comparing file transfer and encryption performance of Java and .NET http://www.lore.ua.ac.be/Publications/pdf/Jagers2004.pdf

[2] .Net Remoting Overview http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconNETRemotingOverview.asp

[3] Parallel computing Vol. 30 ISSN 0167-8191 LNCS (Lecture Notes on Computer Science) 2047

[4] Measurement process - Glasgow university http://www.ece.eps.hw.ac.uk/~amc/msc-projects/lorna_kirkhope/intro_m_p.htm

[5] The Mono Project http://www.mono-project.com

[6] Java website http://java.sun.com

[7] The Server Side, a Java Middleware and Web technologies website http://www.theserverside.com

[8] The Object Management Group http://www.omg.org

[9] The World Wide Web consortium http://www.w3.org

[10] .Net timer research http://www.eggheadcafe.com/articles/20021111.asp

[11] Mailing from Florian Bomers (Sun microsystems employee) on high performance timing in Java