

# P2P with JXTA-Java pipes

Jean-Marc Seigneur  
Distributed Systems Group  
Department of Computer  
Science  
Trinity College Dublin  
seigneurj@tcd.ie

Gregory Biegel  
Distributed Systems Group  
Department of Computer  
Science  
Trinity College Dublin  
Greg.Biegel@cs.tcd.ie

Christian Damsgaard  
Jensen  
Informatics and Mathematical  
Modelling  
Technical University of  
Denmark  
cdj@imm.dtu.dk

## ABSTRACT

The peer-to-peer (p2p) paradigm is attracting increasing attention from both the research community and software engineers, due to potential performance, reliability and scalability improvements. This paper emphasizes that JXTA can help teachers to teach p2p with Java. This paper also presents an approach for performance analysis of JXTA pipes - one of the key abstractions in JXTA, which has not yet been fully evaluated. It explains how to assess a pipe and demonstrates performance results of the JXTA-Java implementation. In doing so, this paper assists software developers in estimating the overall performance and scalability of JXTA, and the suitability of choosing JXTA for their specific application.

## Categories and Subject Descriptors

D.1.0 [Programming Techniques]: General

## General Terms

Performance, Experimentation

## 1. INTRODUCTION

Despite attracting increasing attention from both the research community and software engineers [6, 17, 11], the p2p paradigm is neither well defined nor well understood at the moment. The term peer-to-peer, or p2p, is applied to a large range of technologies. Such p2p technologies are more likely to adopt a highly distributed network-based computing style without centralized control. In addition to improving the performance of information discovery, content delivery, and information processing, such a p2p manner may also enhance the overall reliability and fault-tolerance of applications. For instance changing the implementation of an existing discussion forum from a client-server system to a p2p system increased the availability and reliability of the system [12]. JXTA [9, 16] aims at providing a network programming and computing platform based on the peer-to-peer paradigm that can be implemented on every smart device, where such devices would communicate and

collaborate in a p2p manner. It is a new alternative to existing approaches to implementing distributed systems such as CORBA or Java RMI. Other documents fully describe JXTA [21], however, from a teaching perspective, it is worth detailing some aspects of JXTA. Since the JXTA platform defines a set of protocols designed to address the common functionality required to allow peers on a network to form p2p systems, JXTA provides a useful basis for teaching the p2p paradigm. JXTA encapsulates the mechanisms for a peer to be part of a comprehensive p2p solution, which should provide:

- discovery of other peers and their services
- publishing of the peer's available services
- exchange of data with another peer
- routing of messages to other peers
- query of other peers for status information
- grouping of peers into peer groups

JXTA encapsulates each of these mechanisms in a specific protocol. Therefore it is possible to focus on one aspect of the p2p paradigm, e.g. peer discovery. Different implementations are available, but the Java implementation is the most complete and thus the most convenient implementation to be used for teaching. This standard p2p platform, where the separation of concerns is respected and the active open source community continues to provide reusable software, should save time since there is no need to build core p2p software for teaching. JXTA defines six core protocols [15]: the peer discovery protocol, the peer resolver protocol, the peer information protocol, the peer membership protocol, the endpoint routing protocol and the pipe binding protocol (PBP).

One of the main abstractions in JXTA is the concept of pipe, which describes a connection between a sending endpoint - encapsulation of the native network interfaces provided by a peer - and one or more receiving endpoints. As it is familiar for users of Unix to use a pipe to connect the output from one command to the input of another command, pipes may be used intensively by JXTA developers. A pipe - a kind of virtual communication channel - is used to conveniently connect peers and to send messages between them because a network transport can be accessed without interacting directly with the endpoint abstraction. Any transport capable of unidirectional asynchronous unreliable communication can be used; indeed JXTA specifications specify that the default service pipe provides unidirectional asynchronous unreliable communication. Different endpoint transport implementations are available, e.g. TCP or HTTP. The JXTA implementation provides that kind

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PPPJ 2003, 16-18 June 2003, Kilkenny City, Ireland.  
Copyright 2003 ISBN: 0-9544145-1-9 ...\$5.00.

of basic core pipe but other pipes with better quality of service can be built by developers. At present there are other types of pipes implemented: bi-directional pipe, bi-directional/reliable pipe. Other pipes are planned, e.g. streaming pipes providing efficient data transfer over a flow-controlled channel. The PBP dynamically resolves the set of currently listening peers to specific pipes. As for other resources in JXTA, pipes are uniquely identified by a pipe advertisement, which is used to discover which pipes are available and to resolve the input pipe and output pipe endpoints. The pipes support two modes of communication: point-to-point, i.e. connecting exactly two pipe endpoints; propagate pipe, i.e. connecting one output pipe to multiple input pipes. JXTA implements TLS [20] to secure the communication through pipes. The following keywords are used to easily switch between the different types of pipes:

JxtaUnicast: unicast, unreliable and unsecure pipe

JxtaUnicastSecure: unicast and secure pipe

JxtaPropagate: propagated, unreliable and unsecure pipe

Developers often have to make a choice concerning which underlying technology to use. Performance assessment of distributed software technologies is necessary to make a well-founded decision about which technology to use in a given application domain. It is important that different criteria be considered in order to make the right choice of technology [10]. Performance is one of the main criteria. This paper focuses on pipes performance of the Java implementation of JXTA. Developers are most likely to use them to send and receive messages between peers due to their convenience. Knowing their performance therefore helps to estimate the overall performance of JXTA applications. In doing so, this paper assists software developers in estimating the overall performance and scalability of JXTA, and assists them in evaluating JXTA for their specific application. The performance analysis method that we have used is based on the “performance assessment framework for distributed object architectures” [13]. The following section presents our approach for performance analysis of JXTA-Java pipes.

## 2. JXTA-JAVA PIPES PERFORMANCE

This section first describes the testing method. Then the results and their interpretations are presented.

### 2.1 Testing Method

As said above, the performance analysis method is based on a specific assessment framework [13]. Nevertheless, some changes have been made to get results more appropriate for the JXTA pipe paradigm.

The JXTA model is not a distributed object model as such. There is a JXTA community project [2] that allows the use of JXTA pipes as the underlying communication technology for remote method invocation rather than Java RMI. Nevertheless, it is worth quantifying the performance of JXTA pipes, especially to estimate how the type of pipe impacts performance, e.g. whether the type is JxtaUnicastSecure or not. It can also be used to check whether new builds of JXTA improve pipe performance. A pipe may span different peers. This testing method may be used to assess the performance of pipes connecting two peers separated by context-dependent number of peers with context-dependent types of transport between them. Practically, two different configurations were assessed. The first configuration consists of direct physical connection between two peers using TCP, which is the closest configuration to other tests carried out with the performance assessment framework [13]. The second configuration consists of the use of a

relay peer and HTTP between the peers, which may be mandatory in presence of firewalls. Many other practical configurations are possible but it was beyond the scope of this paper to present all practical cases. The same testing method may be applied for other configurations though.

It is not really possible to accurately compare the results of this paper with the ones in Juric, et al. [4] even if the performance analysis method is the same and all is Java-based: JXTA/Java, CORBA/Java and Java RMI. For instance, the hardware parts of the testing environment are not strictly the same. If one thinks at a job level, in this paper the job is a string of text which has to be sent to a client and returned. This is due to the fact that the aim of this work was to assess the Round Trip Time (RTT) of a message sent from one peer to another. Juric, et al. [4] focused on method invocation and the goal was to measure the overhead of the distributed architectures, by invoking methods without parameters returning different types. Their job could be seen in JXTA as sending an empty message to a peer which would return a message with a string of text. To get a true RTT, the message should contain the string on both trips, as it is done in this paper. By reading these papers, software developers can only get a rough idea of performance comparison between JXTA/Java, CORBA/Java and Java RMI. Nevertheless, developers have to assess performance at the end-user level, where the underlying technology is transparent as long as the function is achieved. For instance, next generation appliances may be CORBA-based or JXTA-based but end-users will be more concerned by the time taken to switch on their appliance through their smart home system than by knowing which underlying technology is used, e.g. remote method-based or message parsing-based.

In contrast, we have respected the fact that the target peer should not carry out any other processing than required for sending back the message after reception.

Among the different criteria for quantitative evaluation of performance, four criteria are relevant for evaluating JXTA pipes: RTT, throughput, data throughput and scalability. The definition of each criteria had to be adapted though:

**RTT** — measures the time needed from the point when the Initiator peer sends a message to the Target peer to the point when the Initiator receives the message back. Any other processing than needed for sending and receiving the message on both peers is minimized as much as it is possible.

**Throughput** — measures the number of message round trips in a given time interval. Throughput and round trip time are inversely proportional. This is the reason that this paper presents only RTT results.

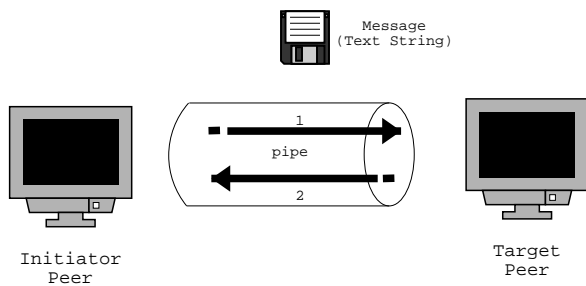
**Data throughput** — measures the efficiency of data transfer. The data is transferred as elements in the message sent and returned. This paper gives results for data as string of text.

**Scalability:** In distributed object technologies, scalability is the evaluation of the performance degradation when multiple clients simultaneously interact with a single server object. For this criteria, it would be more relevant to use JxtaPropagate type of pipes and design another test with multiple target peers.

The same hardware was used for all tests. Listed below are the different pieces of hardware referred to in this document:

“LocalOrInitiatorPC”: PentiumIV 1.7GHz with 256MB RAM

“LANTargetPC”: PentiumII 450MHz with 128MB RAM



**Figure 1: Simplified diagram describing the test environment**

“LANRelayPC”: PentiumIII mobile CPU 866MHz with 256MB RAM

“LAN”: The three computers above were connected to a 10Mb Ethernet hub.

In order to minimize the impact of the environment to get relevant performance results that are reproducible, other tasks running on computers used for testing were kept minimal and were the same for all tests. In addition, the network was free of other traffic by using an isolated LAN and the network configuration was the same for all tests.

All Java sources were compiled and run with the Sun Microsystems JDK1.4.0 and the JVM under Hotspot mixed mode. Two stable builds of the JXTA platform were studied: the build 49b of February 2002 and the build 65e of July 2002. Two types of operating systems were studied separately: Windows and Linux. For the Windows configuration, LocalOrInitiatorPC and LANRelayPC were running WindowsXP and LANTargetPC was running WindowsNT4. For the Linux configuration, Red Hat Linux 7.3 was used.

The same Java source code was used for the different tests. Time was measured with the `System.currentTimeMillis()` method. The precision of this method is 1ms under Linux and 10ms under Windows. We repeated the message round trip one thousand times, though, to get the necessary level of accuracy. We found that 50 repetitions of such observation gave a strong level of confidence in the results. The end of this subsection details how we processed to be confident in our results. Figure 1 depicts a simplified view of the basic unit of work.

We measured the elapsed time for 1000 round trips of one message between two peers<sup>1</sup>. On the Initiator peer side, before starting that loop of round trips, the message is created and a string of text of a specified size is embedded into that message. In doing so, data throughput can be studied by measuring the impact of sending different size of text strings: 1, 100, 1000, 2000, 3000, 4000, 5000, 10000, 20000 or 30000 characters. Then the loop is started: one message is sent to the Target peer and the Initiator peer waits for the reply from the Target peer. Below is the simplified code on the Initiator peer side:

```
Message message, receivedMessage;
message = pipe.createMessage();
message.setString( ' ' +
                  textStringOfXXCharacters );
long startTime, endTime;
for(int j=0; j < 50 ; j++)
```

<sup>1</sup>It is worth mentioning that we had to create two pipes between peers due to the fact that basic JXTA pipes are unidirectional.

```
{
    startTime = System.currentTimeMillis();
    for(int i=0; i<1000; i++)
    {
        outputPipe.send(message);
        receivedMessage =
            inputPipe.waitForMessage();
    }
    endTime = System.currentTimeMillis();
    System.out.println(j + " " +
                      (endTime-startTime));
}
```

On the Target peer side, the message to send back is created and a string of text of the same size as for the Initiator peer is created and embedded into that message. Then, it starts to continuously listen for a message and sends back a message in response. The processing is kept to a minimum to only measure the round trip time. Following is a simplified version of the code achieving that on the Target peer:

```
Message message, receivedMessage;
message = pipe.createMessage();
message.setString( ' ' +
                  textStringOfXXCharacters );
while(true)
{
    receivedMessage =
        inputPipe.waitForMessage();
    outputPipe.send(message);
}
}
```

The tests were executed in two scenarios:

1. local: The Initiator and Target peers ran on the same host.
2. LAN: The Initiator and Target peers ran on two separate hosts connected to a 10Mb Ethernet hub.

A variant of the second scenario has been used in order to use a JXTA relay peer and HTTP-only on the Initiator and Target peers. A third computer, where the relay peer ran, was connected to the hub in that case. “TCP-only” means that the JXTA platform is configured with TCP-only settings on each peer. Neither rendez-vous peer nor relay peer is specified in that configuration. “HTTP-only” means that the JXTA platform is configured with HTTP-only settings on each peer, and that they use a third peer as a relay, which is listed as a HTTP rendez-vous peer. This third peer is run on LANRelayPC and is set to be a HTTP rendez-vous peer and a relay peer as well. All local tests were done on LocalOrInitiatorPC. Concerning tests over LAN, for all tests, the Initiator peer was run on LocalOrInitiatorPC and the Target peer was run on LANTargetPC. The secure option of the pipes was also studied to see the overhead of securing the communication thanks to TLS [20]. Thus, tests were done with two types of JXTA pipes: JxtaUnicast and JxtaUnicastSecure. The default shipped options for the cipher suite used for TLS were used because the only change was to use the keyword JxtaUnicastSecure instead of JxtaUnicast in the pipe advertisement.

Two issues arose when we assessed the accuracy of our results. The first issue concerned the calculation of the necessary number of observations. The second issue involved the analysis of the steady state. We had to define how many observations were needed to get an appropriate level of confidence. There are two requirements on Xi, one occurrence of what is observed, to allow reliable data analysis [14]:

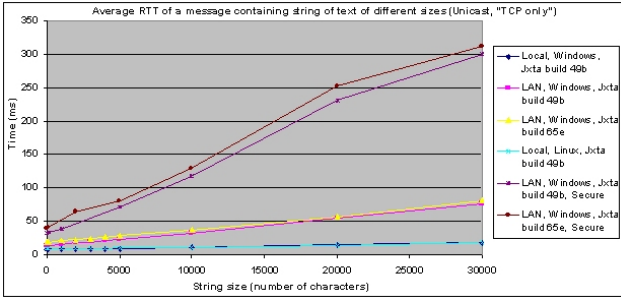


Figure 2: Data throughput

1. Each  $X_i$  should be normally distributed. To satisfy this first requirement we invoked the CLT (Central Limit Theorem): if we obtain  $X_i$  as the arithmetic mean of a sufficient large batch of individual observations, we can expect its distribution to be close to normal. This is the reason that we measure the time of one thousand round trips.
2. All  $X_i$  should be independent. This requirement introduces the second concern, the analysis of the steady state. We assume that each round trip of message is independent, though.

Since the two requirements are fulfilled, we can now calculate  $n$  the number of observations necessary to obtain an accuracy of  $\pm r=1\%$  on loops of one thousand round trips of message with a confidence level of  $100(1-\alpha)=99\%$ . We ran 400 repetitions of one thousand round trips of message in order to be able to apply the following formula:

$$\frac{\bar{X} - \mu}{s/\sqrt{n}} = z \quad (1)$$

where  $\bar{X}$  is the mean,  $s$  is the standard deviation and  $z$  is the inverse of the standard normal cumulative distribution with a probability of  $(1-\alpha)$ . Hence, we validated that fifty repetitions of one thousand round trips is enough to get this level of confidence in our results for the whole range of string sizes. Concerning the steady state, ideally, any data obtained from the transient period must be discarded, but practically it is not easy to detect [14]. Nevertheless, in the case of JXTA pipes, it looks like the first one thousand round trips of messages are processed slightly more slowly than the rest of the round trips. The standard deviation is also higher, showing that it is not a steady state. The slow start may be due to the initial creation of objects needed for processing the requests. However, by following the same approach as for the necessary number of observations, we found out that it is not really significant. In fact, we validated that fifty repetitions of one thousand round trips is enough to get an accuracy of  $\pm 1\%$  on the time spent for a loop of 1000 round trips of messages with a confidence level of 99% for the whole range of string sizes. That is the reason that most of the tests were tripled.

## 2.2 Results and Interpretations

The first set of results in Table 1 lists RTT for TCP-only local configurations. The second and third sets of results in Table 1 list RTT for TCP-only LAN configurations. In all the tables, n/a means that the result is not available, i.e. it was not considered essential to carry out the specific test in question.

As expected, to communicate over a LAN introduces an overhead. If we focus on the results for the JXTA build 49b, the overhead of the network grows with the string size from 62% for a string

Table 2: RTT(Stringsize) linear functions

	offset (in ms)	k (in ms/character)
Local, Windows, Jxta build 49b	7.6718	0.0003
Local, Linux, Jxta build 49b	8.1387	0.0003
LAN, Windows, Jxta build 49b	12.3042	0.0021
LAN, Windows, Jxta build 65e	17.7732	0.0020
LAN, Windows, Jxta build 49b, Secure	30.1358	0.0092
LAN, Windows, Jxta build 65e, Secure	39.8799	0.0095

Table 3: HTTP-only LAN results (Average RTT time in ms)

HTTP-only LAN	JXTA build 49b	JXTA build 65e
Number of characters in message	1	1
Windows JxtaUnicast pipe	200.71	68.97
Windows JxtaUnicastSecure pipe	422.48	136.33

of 1 character to 331% for a string of 30000 characters. In all cases, the RTT looks linearly dependent on the string size, as it presented in Figure 2. It can be approximated with a linear function in the form:

$$(2)$$

In all cases, to use secure communication via JXTA unicast pipes implies an overhead as expected. Of course, secure communication is more useful under a LAN configuration, hence we did not carry out tests for all string sizes. Roughly speaking, the secure overhead grows from 125% for a string of 1 character to 300% for a string of 30000 characters. It is worth mentioning that  $k$  for secure communication is approximately 4.6 times bigger than without security. For LAN scenarios, the RTT also looks linearly dependent on the string size. Nevertheless, there are more fluctuations than without the use of TLS.

In Table 2, there is an estimation of the RTT function for each configuration calculated by linear regression analysis by using the “least squares” method to fit a line through the set of observations.

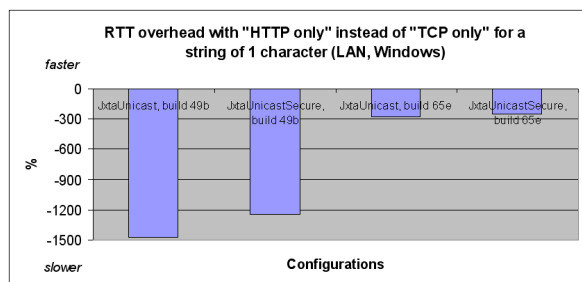
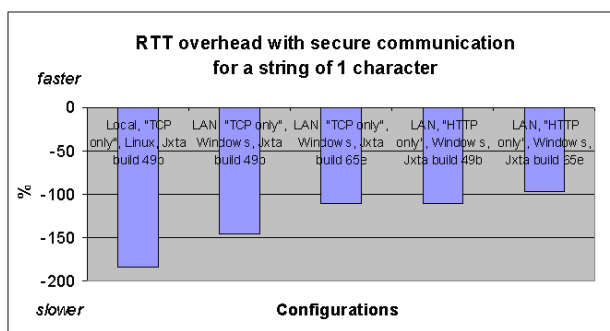
It underlined that newer stable JXTA builds do not automatically improve performance since the older JXTA build used in this paper outperforms the newer build for all TCP-only configurations.

Concerning the results of this testing method for HTTP-only configurations, they show that this test cannot successfully be run for such configurations. JxtaUnicast pipes are unreliable and under such configurations messages were easily dropped. Thus, a new test should be designed to cope with dropped messages. No test at all could be run successfully for HTTP-only under local tests, i.e. Initiator, Relay and Target peers running on the same machine. Some results for LAN tests follow in Table 3 even though their level of confidence is not statistically guaranteed as for TCP-only configurations. In fact, less than 50 repetitions were carried out for the following results. Moreover, only messages with a string of 1 character were tested due to the lower level of confidence in the results. Again, another testing method should be designed to get the same level of confidence as for TCP-only results. For this new test, it would make sense to carry out test for all string sizes.

In all cases, to use HTTP-only configurations implies an overhead as shown in Figure 3. Secure communication does not really

**Table 1: TCP-only results (Average RTT in ms)**

Number of characters in message	1	100	1000	2000	3000	4000	5000	10000	20000	30000
local (JXTA build 49b)										
Linux JxtaUnicast pipe	8.48	8.52	8.59	8.74	8.89	9.06	9.33	10.68	13.93	17.21
Linux JxtaUnicastSecurePipe	24.40	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Windows JxtaUnicast pipe	7.89	7.97	8.09	8.25	8.49	8.98	9.10	10.59	14.14	17.56
LAN (JXTA build 49b)										
Windows JxtaUnicast pipe	12.78	12.91	14.96	16.71	18.62	20.48	22.21	31.72	54.03	75.68
Windows JxtaUnicastSecure pipe	31.42	33.10	38.39	n/a	n/a	n/a	71.06	117.95	230.65	299.66
LAN (JXTA build 65e)										
Windows JxtaUnicast pipe	18.13	18.46	20.31	21.81	23.75	26.01	28.32	36.80	56.87	80.53
Windows JxtaUnicastSecure pipe	38.7	39.99	n/a	64.14	n/a	n/a	79.53	128.12	252.03	311.43

**Figure 3: HTTP-only vs. TCP-only****Figure 4: Secure communication overhead**

increase the percentage of this overhead - roughly speaking, it takes 15 times more time with the old build and 3.7 times with the new build. For both JxtaUnicast and JxtaUnicastSecure, the overhead has been decreased by approximately 80% between the old JXTA build and the new one. That time, the newer stable JXTA build is up to 200% faster than the old build. The introduction of a faster Web server in the relay implementation and other performance work between the two JXTA releases successfully improved the RTT.

Finally, Figure 4 summarizes the different secure overheads that were obtained for a string of 1 character.

### 3. CONCLUSION

In this paper, we have described how we assessed the performance of JXTA pipes - one of the key abstractions in JXTA - and presented results for its Java implementation under different configurations. In doing so, this paper assists software developers in estimating the overall performance and scalability of JXTA, assists

them in choosing JXTA for their specific application. By reading this and other related papers [7, 8, 4, 13, 3], software developers can get a rough idea of performance comparison between JXTA/Java, CORBA/Java and Java RMI. Our results for "TCP-only" have an accuracy of +/-1% on the time spent for a loop of 1000 round trips of messages with a confidence level of 99% for the whole range of string sizes. The interpretation of our results has shown that the RTT of "TCP-only" configurations evolves linearly with the size of the text string embedded into the message. If we take into account that 0.1 second is about the limit for having the user feel that the system is reacting instantaneously and 1.0 second is about the limit for the user's flow of thought to stay uninterrupted [18], we can say that building applications considered as responsive by humans may be an issue with the actual Java implementation of JXTA. If a message has to go through different chained pipes to carry out an action initiated by a user, the time to process may reach the limits above. The introduction underlined that JXTA is handy for teaching the p2p paradigm, especially when the Java implementation of JXTA is used since it is the most complete. Indeed, JXTA provides a comprehensive p2p platform where separation of concerns is respected, each key p2p mechanism is encapsulated in a specific protocol, and the JXTA open-source developers community continuously makes available more reusable software. However, if performance is key, especially for some specific software engineering projects, the Java implementation of JXTA may not be the best choice.

As far as we know there are no in-depth analysis of the performance of JXTA pipes. This paper contributes to the JXTA Benchmark project [5]. A lot of work has been done in distributed objects models performance using Java/RMI, Java/CORBA, or C++/CORBA [7, 8, 4, 13, 3]. Obviously, the same tests should be done with the next builds of the Java implementation of JXTA to check whether performance improves or not. A test coping with dropped messages should be designed to assess HTTP-only configurations. Other scenarios where pipes span different peers would be interesting as well as for JxtaPropagate pipes. Finally, JXTA pipes of other implementation should be studied. Indeed, the c implementation of JXTA [19] and the J2ME implementation [1] are under scrutiny.

### 4. ACKNOWLEDGEMENTS

We are grateful to the JXTA developers community and the members of the DSG p2p working group.

### 5. ADDITIONAL AUTHORS

Additional authors: Yong Chen (Distributed Systems Group, Department of Computer Science, Trinity College Dublin, email: chen-y@tcd.ie), Vinny Cahill (Distributed Systems Group, Department of Computer Science, Trinity College Dublin, email: Vinny.-

Cahill@cs.tcd.ie) and Elizabeth Gray (Distributed Systems Group, Department of Computer Science, Trinity College Dublin, email: Liz.Gray@cs.tcd.ie).

## 6. REFERENCES

- [1] C. W. Akhil Arora and K. S. Pabla. JXTA J2ME Implementation Project. <http://jxme.jxta.org>, February 2003.
- [2] A. Blakey and J. Ernst. JXTA RMI Project. <http://jxta-rmi.jxta.org>, February 2003.
- [3] A. Z. B.M. Juric and I. Rozman. Are Distributed Objects Fast Enough. *SIGS Java Report*, 5:29–38, May 1998.
- [4] R. I. B.M. Juric and H. Marjan. Performance comparison of CORBA and RMI. *Information and Software Technology*, 42(13):915–933, October 2000.
- [5] K. S. P. Brian Sayrs and B. Traversat. JXTA Benchmark Project. <http://bench.jxta.org>, February 2003.
- [6] J. Crowcroft and I. Pratt. Peer to Peer: peering into the future. In *Advanced Lectures on Networking, NETWORKING 2002, Pisa, Italy*, pages 1–19, May 2002.
- [7] A. Gokhale and D. Schmidt. Measuring the performance of Communication Middleware on High Speed Networks. In *Proceedings of SIGCOMM '96, Stanford, CA*, pages 306–317, August 1996.
- [8] A. Gokhale and D. Schmidt. Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks. In *IEEE 17 International Conference on Distributed Systems (ICDCS 97)*, May 1997.
- [9] L. Gong. Project JXTA: A Technology Overview. <http://www.jxta.org/project/www/docs/TechOverview.pdf>, April 2001.
- [10] J. Gyorkos. Measurements in Software Requirements Specification Process. *Microprocessing and Microprogramming*, 40:893–896, December 1994.
- [11] M. Haahr. PISEIA: P2P Infrastructure Simulation, Evaluation and Implementation Architecture. <http://www.dsg.cs.tcd.ie>, March 2003.
- [12] E. Halepovic and R. Deters. Building a P2P Forum System with JXTA. In *Proceedings of the 2 International Conference on Peer-to-Peer Computing*, pages 41–49, September 2002.
- [13] B. Juric, T. Welzer, I. Rozman, M. Hericko, B. Brumen, T. Domanjko, and A. Zivkovic. Performance Assessment Framework for Distributed Object Architectures. In *Proceedings, Advances in Databases and Information Systems Conference ADBIS '99*, pages 349–366, September 1999.
- [14] K. Kant. *Introduction to Computer System Performance Evaluation*. McGraw-Hill, 1992.
- [15] S. Microsystems. JXTA v1.0 Protocols Specification. <http://www.jxta.org/project/www/docs/ProtocolSpec.pdf>, June 2001.
- [16] S. Microsystems. Project JuXTApose. <http://www.jxta.org>, February 2003.
- [17] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-Peer Computing. Hewlett-Packard Technical Report, HPL-2002-57, March 2002.
- [18] J. Nielson. *Usability Engineering*. Morgan Kaufmann, San Francisco, 1994.
- [19] K. S. Pabla and B. Traversat. JXTA C Implementation Project. <http://jxta-c.jxta.org>, February 2003.
- [20] I. E. Taskforce. TLS: Transport Layer Security. <http://www.ietf.org/html.charters/tls-charter.html>, January 2003.
- [21] B. J. Wilson. *JXTA*. New Riders Publishing, 2002.