Improvement of JXTA protocols for supporting reliable distributed applications in P2P systems

Fatos Xhafa[1], Raul Fernandez[2], Thanasis Daradoumis[2], Leonard Barolli[3], and Santi Caballé[2]

[1] Polytechnic University of Catalonia
Department of Languages and Informatics Systems
Barcelona, Spain. fatos@lsi.upc.edu
[2] Open University of Catalonia, Department of Information Sciences
Barcelona, Spain. adaradoumis@uoc.edu
[3] Department of Information and Communication Engineering
Fukuoka Institute of Technology (FIT)
Fukuoka, Japan. barolli@fit.ac.jp

Abstract. In any distributed application, the communication between the distributed processes/nodes of the distributed systems is essential for both reliability and efficiency matters. In this work we address this issue for distributed applications based on JXTA protocols. After a careful examination of the current version of JXTA protocols, we observed the need for improving the original JXTA protocols, such as pipe services, to ensure reliable communication between peer nodes and the discovery and presence service to increase the performance of the applications. The re-implemented protocols have been validated in practice by deploying a P2P network using nodes of PlanetLab platform and testing each of the extended protocols using this real P2P network.

1 Introduction and motivation

The reliability of distributed applications has been largely investigated by researchers of the distributed computing community. While reliability issues are generally well understood for operating systems and classical distributed systems and LANs (see e.g. [3]), there is still few work in addressing these issues for the emergent computational systems. With the development of Internet and other new technologies, distributed systems and applications are becoming the indispensable approach for solving complex problems. Therefore, the reliability issue is nowadays being investigated for Web-based distributed systems / applications (e.g. [6, 13, 11]) and Grid-based computing (e.g. [2, 7]). Moreover, P2P systems are evolving towards an important paradigm for distributed computing. Each time more, P2P systems are used beyond file/data sharing applications, for developing large scale distributed applications that benefit from the immense computing power contributed by millions of peers worldwide. Projects such as "Folding@Home on the PS3" by seti@home for studying the protein folding by utilizing the new Cell processor in Sony's Playstation 3 are allowing to achieve performance previously only possible on supercomputers. The main challenge in developing large-scale P2P applications is how to develop efficient, scalable and reliable distributed systems from inexpensive unreliable computers contributed to P2P network by millions of individuals. One of the today's technologies used in developing P2P systems is JXTA [1, 9, 10]. This is a recent technology, which has been used in several P2P projects [5] and is currently drawing the attention of many researchers and developers of the P2P and Grid computing. In particular the reliability and efficiency issue is being studied for JXTA-based applications (e.g. [4, 12]).

This work is motivated by the need to support the development of efficient and reliable P2P applications using JXTA protocols. To this end, we have carefully analyzed the current JXTA protocols and report here several limitations of most important protocols of JXTA. After examining and pointing out such limitations, we further propose a solution to them through re-implementation/extensions without damaging the genericity of JXTA protocols. More precisely, in this work we have considered the following protocols and services: *Peer Discovery Protocol*, *Discovery Service*, *Peer Information Protocol*, *Peer Information Service*, *Peer Resolver Protocol*, *Resolver Service*, *Pipe Binding Protocol* and *Pipe Service*, *Endpoint Routing Protocol* and *Endpoint Service*. Our approach is validated in practice by deploying a real P2P network using the nodes of the PlanetLab platform [8], a planetary scale distributed infrastructure, and have experimentally evaluated the performance and reliability of the re-implemented protocols.

The rest of the paper is organized as follows. We give in Section 2 the evaluation of the JXTA protocols and their limitations. In Section 3 we present the re-implementation of the JXTA protocols; their experimental evaluation is given in Section 4. Finally, we conclude in Section 5 with some remarks and indicate directions for future work.

2 The JXTA protocols and their limitations

2.1 The JXTA protocol and services

The services offered by JXTA library are based on its own protocols and serve as the starting point in the development of a P2P network using JXTA. Certainly, it is upon the developer to extend/implement these services according to his needs. JXTA offers essentially six protocols, each one with its corresponding services. In fact, in developing JXTA-based P2P applications, not all these protocols and services are necessarily used. These protocols and services are the following:

- *Peer Discovery Protocol* and *Discovery Service*, which serve to advertise the proper resources and to discover the resources of other peers.
- *Peer Information Protocol* and *Peer Information Service*, which are in charge of obtaining state information of local or remote peers.
- *Peer Resolver Protocol* and *Resolver Service*, which are used for sending a general query to one or more peers and receiving one or more responses in order to interchange desired information.

– *Pipe Binding Protocol* and *Pipe Service*, which serve for establishing a virtual communication channel, that is a *pipe*, to enable sending and receiving data from other peers.
– *Endpoint Routing Protocol* and *Endpoint Service* for routing to other peers. The route information contains an ordered sequence of relay peers that must be used to send a message to the specified destination peer.
– *Rendezvous Protocol* and *Rendezvous Service*, which constitute the mechanisms through which peers can be subscribed to a propagated service or a source peer can propagate a message to all peers subscribed to the service.

2.2 Limitations of JXTA protocols

In any P2P network, independently of its specific purposes, there are several key issues to tackle: (a) publishing and receiving services; (b) the network connection; and (c) message sending/receiving. JXTA offers protocols and services to support these needs, however, as shown in this work, they have several limitations that we overcame by adding new functionalities to the original services. We present next these limitations of JXTA protocols.

Managing the presence mechanism. The presence of a peer node in a peerGroup of the JXTA network is provided by the document of presence notification. This document is published by each peer and reaches the local cache of all peers, where it will "stay" for a certain time. The presence mechanism is very important for the JXTA network in order for a peer's advertisement and therefore its services, to reach all peers that are present. Most importantly, the rendezvous must know all peers connected to the peergroup because the rendezvous is in charge of synchronizing the local caches of peers and compute the routes to be used by any peer to reach all the connected peers of the group. The advertisement document is of the type *PeerAdvertisement* and must periodically be published in order for other peers of the group to know the updated presence information of the peer that publishes the advertisement document. However, this publishing is not done automatically in JXTA; hence any application using JXTA must manage it by its own! Essentially, any JXTA-based application has to manage the information of the updated presence (a sort of presence refreshment) of any peer node in any peerGroup it belongs to.

Managing the connection to groups. In any mixed P2P network, that is, a P2P network in which we have *broker peers* and *client peers* [12], any client peer can be connected only to one peerGroup by discovering and establishing communication with a rendezvous. Therefore it is necessary to have a broker peer in any peerGroup, which must act as a rendezvous in order to achieve that client peers get connected and the broker could notify the presence of the peer to the rest of the nodes of the peerGroup. It should be noted that the connection to a peerGroup is done differently depending on whether the peerGroup is specific or is a general group. The general group is the *NetPeerGroup*, the peerGroup to which must get connected any peer to join the JXTA network and to collaborate/work with other peers. On the other hand, the connection to other peerGroups is not indispensable for the peer to start working in the JXTA network, but rather it serves to make much more specific the peer's characteristics and functionalities. Moreover, using broker peers and brokerage services, the number of peers that could be used for a certain purpose can be reduced. In JXTA, there are no such mechanisms for broker peers.

Managing the communication mechanism. In JXTA, pipes are used as a basic direct communication mechanism between two peers belonging to the same peerGroup. Pipes are of two types: InputPipe and OutputPipe. Clearly, if a peer A wants to send a message to a destination peer B, it is necessary that an instance of A's outputPipe must be connected to an instance of B's inputPipe. In order to establish this connection, the peer A must know the B's inputPipe and the peer B must accept messages from the A's outputPipe. This is achieved in JXTA by using descriptions of pipes in a way that they are unique, that is an inputPipe can only receive messages from an outputPipe having the same description. The description is done in XML documents, called PipeAdverstisement, which is of type advertisement and has defined, besides its unique identifier, several parameters such as name, type and description. Therefore in order for the peer A to send a message to the peer B, both inputPipe and outputPipe must be defined in the same PipeAdvertisement and both peers must agree which one to use. Thus, we have to find the appropriate protocol to assure the communication among all peers of the same peerGroup. Again, this is not assured by JXTA pipes. Moreover, JXTA pipes do not assure nor notify the failure that might occur in message delivery and once a message is sent out, it is not attempted again in case of failure.

3 Improvements of JXTA protocols

In order to overcome the limitations of the JXTA protocols shown in the previous section, we re-implemented the JXTA protocols/services by adding new functionalities and control to the original services offered by JXTA.

Managing the presence and service mechanisms. Recall that the presence is done through the presence notification document. This document must be periodically published in order for other peers of a peerGroup to know the presence of the peer. This is one of the limitations of JXTA that we have solved by adding to peer's functionalities, the publishing of its own PeerAdvertisement periodically. In this way, the peers of the peerGroup will know the presence of the peer and hence they will send to the peer their future advertisements. Moreover, in order for the peer to know all services of the P2P network, it has to discover such services, which are found in different advertisements and are stored in local cache.

Again, the peer's cache must be frequently updated. One way to do this, is to look up local caches of peers in the peerGroup by sending queries. A query is a request sent to all peers of the peerGroup which, upon reaching a peer, examines its local cache, and recollects all the advertisements of the specified type. When such query reaches a rendezvous, it recollects all advertisements of all peers of the peerGroup.

Finally, the results of the query are stored in the local cache of the peer, which can use them later on for its purposes. The main issue here is thus the synchronization of the local cache of a peer with local caches of other peers of the peerGroup. To this end, in our implementation, a query is sent periodically to all peers of the peerGroup. More precisely this is done in a new module[4], called DiscoveryOverlay, as shown in Fig. 1.

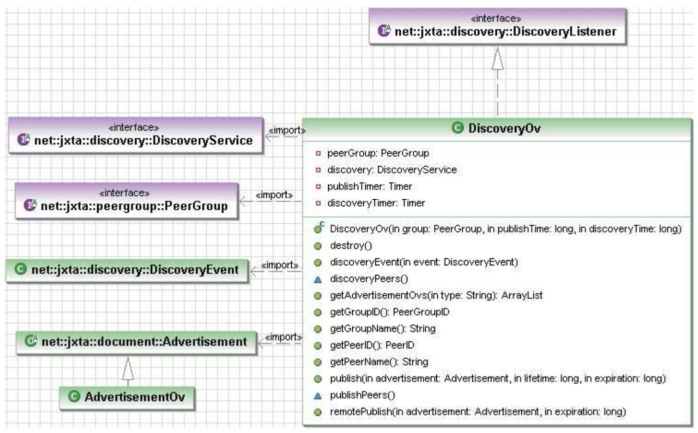
«interface»
net::jxta::discovery::DiscoveryListener
«interface»
net::jxta::discovery::DiscoveryService
«import»
DiscoveryOv
peerGroup: PeerGroup
discovery: DiscoveryService
publishTimer: Timer
discoveryTimer: Timer
DiscoveryOv(in group: PeerGroup, in publishTime: long, in discoveryTime: long)
destroy()
discoveryEvent(in event: DiscoveryEvent)
discoveryPeers()
getAdvertisementOvs(in type: String): ArrayList
getGroupID(): PeerGroupID
getGroupName(): String
getPeerID(): PeerID
getPeerName(): String
publish(in advertisement: Advertisement, in lifetime: long, in expiration: long)
publishPeers()
remotePublish(in advertisement: Advertisement, in expiration: long)
«interface»
net::jxta::peergroup::PeerGroup
«import»
net::jxta::discovery::DiscoveryEvent
«import»
net::jxta::document::Advertisement
«import»
AdvertisementOv

Fig. 1. UML diagram of the overlay module

Managing the connection to groups. Recall that any peerGroup needs a rendezvous peer. The principal peerGroup is that of NetPeerGroup, the connection to which is done according to the peer's configuration: (a) connection to the default rendezvous of JXTA. This is certainly inefficient since we cannot control the rendezvous; (b) connection to rendezvous pre-specified by the concrete JXTA application; (c) connection to rendezvous specified in a rendezvous list. In this later case, the JXTA application just needs to know the address where to find the rendezvous list. Note that the rendezvous list can be changed independently of the application, which is desirable in P2P applications. Moreover, once a peer is connected to the P2P network through the principal peerGroup, it can join any peerGroup by knowing the GroupAdvertisement of such groups. However, for this to be done, the peerGroups must exist, there must be at least one rendezvous and the GroupAdvertisements must be propagated. In our P2P network of broker peers and client peers, the brokers are the governors of the

[4] The abbreviation ***Ov*** stands for *Overaly*, which refers to the new implementation.

network and the organization of the groups. Thus broker peers are in charge of creating the groups and propagate the GroupAdvertisement accordingly.

Managing the communication mechanism. Recall that in order for a peer to send a message to another peer, there must exist a connection between an outputPipe and inputPipe defined in the same PipeAdvertisement and therefore both peers have to agree on which PipeAdvertisement to use. One possible solution would be to create a PipeAdvertisement for any possible connection between any two peers of a peerGroup according to a predefined protocol. This is not efficient given the large number of different PipeAdvertisements needed to manage the connections.

We decided then to create a unique PipeAdvertisement by a concrete peer in a peerGroup, having thus operative only one inputPipe through which to receive messages from all the rest of peers of the peerGroup. It is necessary to find a way to create such a unique PipeAdvertisement per peer and how other peers can know the peer's PipeAdvertisement. Since in JXTA, each peer has its unique PeerID, we associate a PipeAdvertisement with a peer by including the PeerID in PipeAdvertisement variables. Once the PipeAdvertisement is created, the source peer has to create an outputPipe and the destination peer has to create an inputPipe, both of them defined with the same PipeAdvertisement.

We have thus implemented this approach for the case of receiving and sending messages and pipe services (details are omitted). We show in Fig. 2 the UML diagram of the re-design of the JXTA pipe services mechanisms.

Managing the peer's state information. As already mentioned, keeping updated information of the network is crucial in P2P applications. To this end, we have implemented functionalities that allow to recollect information on the nodes of the network and publish it periodically through advertisements. More precisely, each time a node changes its state, due to any network event or application event, the node's state information is updated. This information is defined at three levels: for the last k hours, for the current session and for all sessions (historical data). The peer node publishes this information using advertisements, which are periodically published. Moreover, the peer node computes several statistics, which are sent together with the peer's state information; thus, any peer can use statistical information of other peers for efficient decision-taking. To this end, we have implemented a module LocalCache, which has functionalities to control the local cache of the peer (see Fig. 3).

4 Testing the re-implementations of the JXTA protocols

4.1 Deployment of the P2P network using PlanetLab nodes

In order to evaluate the performance of the re-implemented protocols, first we deployed the P2P network using nodes of the PlanetLab [8], a planetary-scale

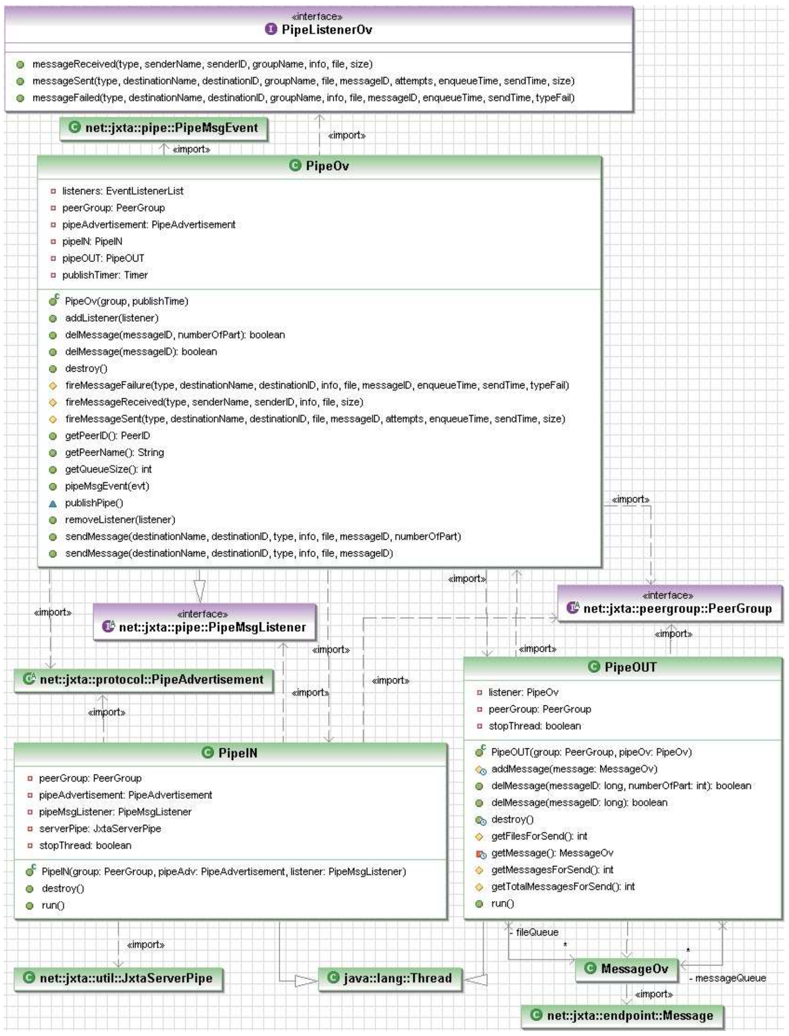
«interface»
PipeListenerOv
messageReceived(type, senderName, senderID, groupName, info, file, size)
messageSent(type, destinationName, destinationID, groupName, file, messageID, attempts, enqueueTime, sendTime, size)
messageFailed(type, destinationName, destinationID, groupName, info, file, messageID, enqueueTime, sendTime, typeFail)
net::jxta::pipe::PipeMsgEvent
«import»
PipeOv
listeners: EventListenerList
peerGroup: PeerGroup
pipeAdvertisement: PipeAdvertisement
pipeIN: PipeIN
pipeOUT: PipeOUT
publishTimer: Timer
PipeOv(group, publishTime)
addListener(listener)
delMessage(messageID, numberOfPart): boolean
delMessage(messageID): boolean
destroy()
fireMessageFailure(type, destinationName, destinationID, info, file, messageID, enqueueTime, sendTime, typeFail)
fireMessageReceived(type, senderName, senderID, info, file, size)
fireMessageSent(type, destinationName, destinationID, file, messageID, attempts, enqueueTime, sendTime, size)
getPeerID(): PeerID
getPeerName(): String
getQueueSize(): int
pipeMsgEvent(evt)
publishPipe()
removeListener(listener)
sendMessage(destinationName, destinationID, type, info, file, messageID, numberOfPart)
sendMessage(destinationName, destinationID, type, info, file, messageID)
«interface»
net::jxta::pipe::PipeMsgListener
«interface»
net::jxta::peergroup::PeerGroup
net::jxta::protocol::PipeAdvertisement
PipeOUT
listener: PipeOv
peerGroup: PeerGroup
stopThread: boolean
PipeOUT(group: PeerGroup, pipeOv: PipeOv)
addMessage(message: MessageOv)
delMessage(messageID: long, numberOfPart: int): boolean
delMessage(messageID: long): boolean
destroy()
getFilesForSend(): int
getMessage(): MessageOv
getMessagesForSend(): int
getTotalMessagesForSend(): int
run()
PipeIN
peerGroup: PeerGroup
pipeAdvertisement: PipeAdvertisement
pipeMsgListener: PipeMsgListener
serverPipe: JxtaServerPipe
stopThread: boolean
PipeIN(group: PeerGroup, pipeAdv: PipeAdvertisement, listener: PipeMsgListener)
destroy()
run()
- fileQueue
*
*
- messageQueue
net::jxta::util::JxtaServerPipe
java::lang::Thread
MessageOv
net::jxta::endpoint::Message

Fig. 2. UML diagram of the Pipe Service re-implementation

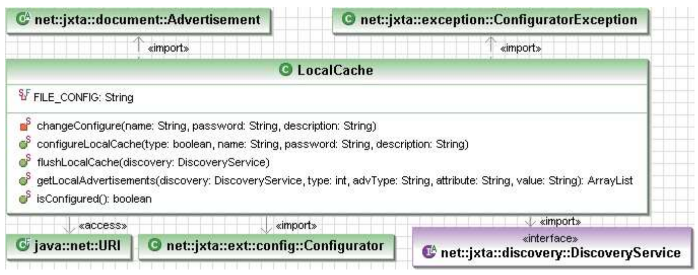
net::jxta::document::Advertisement
net::jxta::exception::ConfiguratorException
«import»
LocalCache
FILE_CONFIG: String
changeConfigure(name: String, password: String, description: String)
configureLocalCache(type: boolean, name: String, password: String, description: String)
flushLocalCache(discovery: DiscoveryService)
getLocalAdvertisements(discovery: DiscoveryService, type: int, advType: String, attribute: String, value: String): ArrayList
isConfigured(): boolean
«access»
java::net::URI
net::jxta::ext::config::Configurator
«interface»
net::jxta::discovery::DiscoveryService

Fig. 3. UML diagram of information management re-implementation

platform[5]. The sample set of PlanetLab's machines forming our slice was about 25 nodes. Moreover we used the cluster nozomi.lsi.upc.edu (a main control node + five computing nodes).

4.2 Reliability tests and computational results

We conducted several experiments to test the reliability and efficiency of the re-implemented protocols. The experimenting was done twice (for the original and re-implemented version) under the same P2P network of PlanetLab nodes.

Testing the presence and service management. When a peer client joins the network for the first time, its local cache is created empty. Next, the local cache must get synchronized, at first place, with the local cache of the rendezvous and then with those of other peers of the same peerGroup. On the other hand, if the peer has already joined the P2P network in previous sessions, the advertisement are stored in its local cache. We observed and experimentally measured that the synchronization for the first connection takes, as expected, much more time than when the peer has already done some prior sessions in the P2P network.

Once the peer has its local cache synchronized, we experimentally studied the peer's services. Peer's services are implemented in a way that each service can have its lifetime and its publishing frequency. We briefly mention here three scenarios that we considered: (a) Statistics services: these services distinguish for their long lifetime (e.g. order of minutes) and a high publishing frequency (e.g. order of seconds, say, 30 seconds). By monitoring the statistics of the P2P nodes, we observed that these services correctly kept the information on all peers. (b) Rendezvous services: these services include criteria for choosing a peer for efficient computation or efficient data transmission, which require that the rendezvous' advertisements must be synchronized because through it are synchronized the local caches of other peers. (c) Peer common services (e.g. information on peer node): these services have usually a short lifetime, otherwise in case the peer is disconnected from the P2P network its services would be *alive* even the node is not present in the network. Experimental values showed that 60 seconds was a good value for the lifetime of such services, meaning that an invalid service could be alive during that time but because the publishing time is inferior to that, a new publishing could take place.

The time connection to a peerGroup. Recall that at least one rendezvous of the general group must be permanently operative so that different peers will try to join the network by trying to connect to the rendezvous. In order to increase the possibilities that a peer will always establish connection to a rendezvous we used a rendezvous list implementation. The experimental data showed that, although it could take more time to establish the connection (now a peer has to examine the list of rendezvous), it considerably increases the possibility of a peer to establish connection with a rendezvous. In fact with our real P2P network, we did not observe failure to establish connection.

[5] At present, composed up of 753 nodes hosted in 363 different sites.

The communication test. One important parameter to measure is the pipe connection times, which depending in the type of peer and the network could vary considerably. Essentially, in order to establish a pipe, JXTA has to previously do some communications based in a communication protocol and each of these connections has its ping time. Due to this variability, considering a fixed timeout (say 2 secs) turned out to not be a good choice; it worked very well for cluster-based P2P network but it was not reliable for a geographically distributed P2P network. This motivated precisely one of the improvements presented in this work, namely, the progressive augmentation of the window timeout, which in our experiments was upper bounded by 40 seconds. Recall that in our re-implementation of pipes, if a message is not delivered to destination peer, it is attempted again to send it. Thus, the number of attempts depends on the time window used. The experiments showed that by progressively increasing the time window to at most 40 seconds, almost all messages were sent to destination. By using this approach less than 2% of messages that could not be delivered (e.g. because the peer is already disconnected) were reported as errors as opposed to 32% of undelivered messages that were reported as errors with a fixed timeout (see Fig. 4).

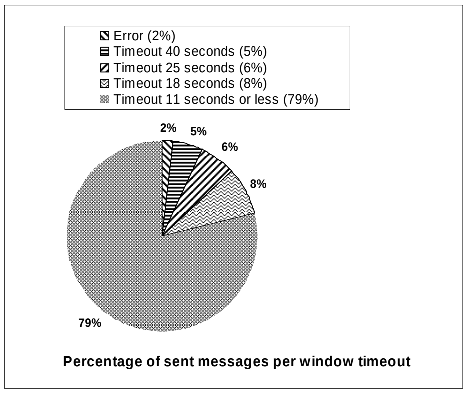

Fig. 4. Time out values and the corresponding percentage of sent messages

Information management test. In our approach, each node publishes a large quantity of information and therefore peer nodes of the P2P network have better knowledge of the peerGroup / the network and can use the information of other peers in decision-taking process. In this context, we considered as a test finding the best peer for efficiently running a task from many candidate peers. In absence of information about the other peer nodes, the task assignment would essentially reduce to a random assignment, which is not efficient. By using the information published by the peer nodes brokers are able to find the best peer among different candidates. Thus, we implemented a data-evaluator model as part of brokerage services to identify the best peer for executing a task.

5 Conclusions and future work

In this work we have analyzed the JXTA protocols/services and have shown several limitations of these protocols regarding the efficiency and reliability of P2P JXTA-based applications. The analyzed protocols include discovery, peer information, peer resolver, and pipe binding protocols/services, among others. We then proposed and re-implemented these protocols in order to overcome the identified limitations. The proposed improvements of the JXTA protocols have been validated in practice using a real P2P network deployed in nodes of PlanetLab platform. The experimental results showed the improvement of both efficiency and reliability of JXTA protocols and services.

In our future work we plan to use the re-implemented protocols in large-scale JXTA-based applications for data intensive computing in highly dynamic environment using a large number of unreliable peer nodes.

Acknowledgements

This work has been partially supported by the Spanish MCYT project TSI2005-08225-C07-05.

References

1. D. Brookshier, D. Govoni, N. Krishnan, and J.C Soto. *JXTA: Java P2P Programming.* Sams Publishing, 2002.
2. I. Foster and C. Kesselman. *The Grid - Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, 1998.
3. Rachid Guerraoui and Luís Rodrigues. *Introduction to Reliable Distributed Programming.* Springer Verlag, 2006.
4. Emir Halepovic and Ralph Deters. The costs of using jxta. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 160. IEEE, 2003.
5. JXTA Projects: https://communications.dev.java.net/ (As of March 2nd).
6. Markus Keidl, Stefan Seltzsam, and Alfons Kemper. *Technologies for E-Services*, vol. 2819, chapter Reliable Web Service Execution and Deployment in Dynamic Environments. Springer, 2003.
7. Elzbieta Krepska, Thilo Kielmann, Raül Sirvent, and Rosa M. Badia. A service for reliable execution of grid applications. In *2nd CoreGRID Integration Workshop, Krakow, Poland, October 2006.*, pages 232–242, 2006.
8. Planet Lab. http://planet-lab.org/.
9. S. Li. *Early Adopter JXTA.* Wrox Press Information Inc., 2003.
10. S. Oaks, B. Traversat, and L. Gong. *JXTA in a Nutshell.* O'Reilly, 2003.
11. Krzysztof Ostrowski and Ken Birman. Extensible web services architecture for notification in large-scale systems. In *Int. Conf. on Web Services.* IEEE, 2006.
12. Joan Esteve Riasol and Fatos Xhafa. Juxta-cat: a jxta-based platform for distributed computing. Proceedings of the 4th Int. Symposium on Principles and Practice of Programming in Java, 72–81, 2006. ACM Press.
13. Ryan Peterson Venugopalan Ramasubramanian and Emin Gun Sirer. Corona: A high performance publish-subscribe system for the world wide web. In *In Proceedings of Networked System Design and Implementation, California*, 2006.