An infrastructure-less peer-to-peer framework for mobile handheld devices

Mario Bisignano, Giuseppe Di Modica, Orazio Tomarchio
Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Università di Catania
Viale A. Doria 6, 95125 Catania - Italy
Email: \{Mario.Bisignano,Giuseppe.DiModica,Orazio.Tomarchio\}@diit.unict.it

Abstract

Mobile handheld devices are becoming more and more widely common among users. Their increasing computing and storage capabilities together with the widespreading of new wireless communication technologies, enable these devices for new and useful applications. The P2P paradigm, which has recently experienced a wide diffusion among Internet users, seems to be very well suited to effectively support mobile applications. However, several questions arise when trying to develop such systems: existing solutions do not address all the requirements of these mobile environments. In this paper we present a framework to develop mobile applications which exploits the P2P paradigm: it is designed to work on J2ME enabled mobile devices, interacting each other in an ad-hoc fashion. One of the key feature is to be interoperable with JXTA, a well-known P2P open platform, by extending the advantages of P2P computing to mobile computing environments. A simple but effective example application is also provided showing the viability of the designed framework.

Keywords: P2P computing, MANET, mobile devices, JXTA, J2ME

1 Introduction

The availability of powerful personal computers and high bandwidth home connectivity are changing the way users cooperate and exchange information on the network. Peer to peer computing [18, 10] is becoming a common paradigm for many distributed applications, besides the classic file sharing one, often adopted as a synonym of P2P [17, 8, 7]. Even if some definitions of what P2P is are available in literature [21], no general consensus to what extent this term can be used exists. As some authors state, it is useful to identify the basic features that characterize P2P computing: actions which are carried out at the "edge" of the network (where users resides with their devices), resource sharing and direct communication among peers [1]. Of course, according to the requirements and peculiarities of the applications, each one of the previous features will be more or less stressed. Current P2P applications and architectures are mainly designed to work in fixed and wired infrastructures, where they realize an overlay network of connected peers [17, 8, 7, 18].

But the growing diffusion of mobile personal computing devices, together with the proliferation of wireless communication technologies [11], also demand for the adoption of P2P paradigms in the mobile computing area [15, 3, 23]. This need gives rise to new and challenging problems to face with [2]: indeed, in these environments, on one hand limited resource devices and specific development and runtime environments are used, on the other one wireless access technologies which are, sometimes, quite different from the wired ones are employed (connection instability, limited bandwidth, etc.)

As far as the application development environments are concerned, when dealing with smartphone, mobile phones and PDAs device categories, one of the most adopted and widespread development systems, which is also open and interoperable, is J2ME (Java micro Edition) [12]. This is a stripped down version of J2SE (Java Standard Edition) that fits into small devices with lim-
ited CPU and memory, such as PDAs and mobile phones. It includes a Java Virtual Machines and runtime classes needed to download and run different applications on mobile devices.

As far as network environment is concerned with, when trying to extend the P2P paradigm to mobile environments, it can be also considered network architectures where mobility, decentralization and self-organization are the rule rather than the exception, as it happens for example in mobile ad-hoc networks (MANETs) [19, 4, 5]. They are a new paradigm of networks composed by a collection of autonomous nodes that do not rely on a predefined and fixed infrastructure. Nodes of such a network are often mobile, communicating with each other through wireless links and maintaining connectivity in a decentralized manner: each node operates as both a host and a router, and network connectivity and control is reached through the cooperation of all the nodes. The network topology is, in general, dynamic because of nodes frequently entering and leaving the radio coverage area of the other nodes.

The design of a P2P computing framework for these environments must take into account the previous constraints, as well as the interoperability with P2P system in traditional networks [14, 2]. To satisfy this requirement, in our work the JXTA system has been used as a reference platform: it is a P2P computing platform originally conceived by Sun, but now carried on by many contributors from open-source community [20]. JXTA technology may be seen as a basic framework on top of which developers can concentrate on functionality of their own applications, without worrying about low-level details of the P2P system, that are indeed provided by the underlying runtime system. The key objectives of the JXTA project are interoperability, platform independence and ubiquity [9]. The current implementation of JXTA technology is based on Java, and provides a basic "shell" application through which both developers and users can experiment with its capabilities, developing prototype applications and controlling the peer’s environment.

Inside the JXTA project, a subproject called JXME [13] (JXTA for Micro Edition) has been established, whose goal is to develop a JXTA compliant system on J2ME enabled devices. However, as it will be later explained (described), this system exhibits some limits and constraints, which are basically related to the need of a permanent connection with a special peer belonging to the JXTA network: JXME peers cannot properly work if only J2ME enabled peers exist on the network.

Our work start just from these considerations; our goal is to design and develop a middleware for mobile P2P computing satisfying the following basic constraints:

- to overcome the JXME architectural limits and constraints
- to be compatible with JXTA protocols
- to be able to work in MANET environments, even when disconnected from traditional JXTA networks.

In this paper the architecture of this systems is presented, by describing the problems encountered, their solution, as well a critical evaluation of the obtained system by using a simple application exploiting the peculiarities of our system.

The rest of the paper is structured in the following way: in Section 2 an overview of JXTA and JXME is given, together with the several limits encountered in the JXME architecture. In Section 3 design issues of the developed framework are presented, while the details of the system architecture are described in Section 4. Section 5 presents an example application in order to validate the proper working of the whole system. Finally we conclude the work in Section 6.

2 JXME constraints

The purpose of the project JXME [13] (JXTA for J2ME) is to provide a JXTA compliant framework, designed for portable devices whose resources are limited. The market area involved ranges from the last generation of mobile phones to the most powerful PDAs. The developers of the project present JXME as an environment that allows the above-mentioned devices to be part of a peer-to-peer network, both for the activities of a network of J2ME devices, and for taking part (with some restrictions) in the peer-to-peer activities of a JXTA network active on a desktop, on a workstation or on a network server. Starting from this definition, the use of portable devices equipped with JXME should be possible for the creation of a pure peer-to-peer network. After describing the main features of JXTA and its architecture, we will also explain why
the JXME system does not fulfill the requirements for our work.

2.1 The JXTA architecture

The JXTA architecture [20] provides a set of protocols with the purpose to create a common platform enabling the development of distributed P2P applications. JXTA creates a virtual network above the real physical structure of the networks, and allows the peers to interact directly and to organize independently from the network protocol they use (see Figure 1). JXTA has a three-level logic architecture: core, service, and application. The core level contains the modules necessary for setting a P2P network. The service level contains some basic services that can be used by any application uniformly. Finally, the application level is used for the implementation of applications. The levels are very modular, and they allow the introduction of new customized services.

![Figure 1: Abstraction model of a JXTA network](image)

The basic element of the JXTA network is the peer. It is univocally identified through a logical peer address, in order to abstract from the node physical address. The peers are grouped into peer groups, in order to share services or for security reasons. The peer groups divide the network into logic regions, in order to define the spread of search into limited domains. All the peers that belong to a specific group have the same services. Each instance of a service can work independently or cooperate with another instance on another peer. Since each peer of the group can provide a specific service independently, the service is always guaranteed, as long as at least one node is active.

Peers, peer groups, and services (to which we will refer with the term “resource” from now on) are identified with a unique ID, in order to guarantee a uniform logic addressing for the whole network, notwithstanding the location. To make a resource visible in the network, a specific XML document (called advertisement) must be published. It contains all the information needed to locate and use the resource. Each peer saves, publishes and exchanges the advertisements, in order to know the resources available in the network. Each advertisement is published with a maximum life time, so that each peer must (after some time) ask for the information again. This way, the validity of the information is verified. The Discovery Service is the service that searches for information within a JXTA. Applications executing in a JXTA network communicate through some virtual communication channels called pipes, which abstract from the physical and logic location of the peer.

Two kind of special peers exist in JXTA: Rendezvous Peers and Relay Peers, also called SuperPeers because they provide some special functionalities [6]. A Rendezvous Peer, whose role has evolved in JXTA 2, maintains an index of advertisements published by its edge peers: in fact, each peer during its initialization phase, has to register itself on a Rendezvous peer. The role of Rendezvous peers is ultimately to provide an effective resource discovery mechanism. Relay Peers are instead used for bridging peers that are hidden by a NAT or divided by a firewall. They also provide the ability to spool messages (acting as proxies) for unreachable or temporarily unavailable edge peers.

2.2 The JXME architecture

The purpose of the JXME project [13] is to transfer the features of JXTA to portable devices equipped with J2ME with MIDP (CLDC) profile [12]. The requirements of JXTA make the implementation difficult on devices with limited resources, since they exceed the resources available with a MIDP profile. In particular, these are the purposes of the JXME architecture:

- to be compatible with JXTA;
- to provide a peer-to-peer infrastructure for systems with limited resources;
- to be easy to use for the developers;
• to be compatible with MIDP devices.

Due to the limited resources of the devices, and to the limits of the MIDP, we outline that the JXME peers have the function of edge peers, because they cannot play complex roles in the network, and they cannot provide any service to the other members of the group. For this reason, the functions that require a wider use of resources have been moved to special peers of the JXTA, called JXTA Relay, as it is shown in Figure 2.

![Figure 2: Structure of a JXME network](image)

At the time of the first activation, each JXME peer requests to connect to a JXTA Relay; if this connection is not available, it does not activate, and waits until the connection is possible. Once the connection has started, the Relay maintains the state of the peer, and makes it visible to the rest of the JXTA network. Should the connection with the Relay fail, the peer remains idle until it can connect to the Relay (either the Relay of the previous connection or another one of the network) again. Of course, any connection of the peer with the other members of the network cannot take place without a connection to the Relay.

Messages are sent and received by using the HTTP protocol. The JXME mobile peer takes the function of client, while the Relay takes the function of server. In this case, the Relay performs two different activities: it creates a queue of messages to be sent to the peer, and converts the XML messages (which are used in JXTA) into binary messages, in order to optimize the limited resources of mobile devices. Each JXME peer regularly polls the Relay, checking whether it contains some messages. In order to send the messages, the sender peer sends the binary message to the Relay, which converts it into a XML message, and sends it to the recipient.

The Relay’s action is also necessary for the creation of resources in the JXME network. For instance, if a peer wants to create a group, it sends its request to the Relay, which creates the new group, publishes and stores the corresponding advertisement, and sends only the name and the ID of the new group to the requesting peer. The same procedure is followed in any operation for the creation or the use of resources in the network.

Finally, the network resources are searched by sending a request to the Relay, and by specifying the type of resource that should be found. The Relay searches for the result, and then optimizes it, by deleting any useless field; then the Relay waits for the requesting peer to perform the polling operation, in order to send the final result. The Relay is also responsible for the pipe management, as well as the creation, binding, sending and receiving the messages.

### 2.3 JXME limits in MANET environments

The important role played by the JXTA Relay in the JXME network is a considerable limit in the creation of a pure peer-to-peer structure in mobile environments, especially when considering MANET-like architectures. A MANET network consists of a collection of mobile autonomous nodes that do not rely on a predefined and fixed infrastructure \[4, 5\] (see Figure 3):

These are the main features of a MANET network:

- no infrastructure is required; for this reason, the network can be used anywhere, even if the geographical conditions do not allow to build wired connection;
- its structure is self-organizing, unlike the wired network, whose structure is fixed and determined by the cabling. Two devices can connect in the wireless network at the same time, provided that they are within their scope. For this reason, the network topology always changes, also considering that mobile devices are used;
- it is fault-tolerant, because each node is independent. Consequently, the net is robust. This means that, if a node falls, the overall condition of the network is not affected.
- dynamic topology: the topology of a MANET changes according to the scope and the mobility of each peer.
The architecture of the framework presented in this paper needs to consider the features and the issues present in a MANET network, as well as being compatible with the JXTA environment. The above-mentioned limits of JXME may even be paradoxical, if we consider one mobile device next to another, which cannot communicate because no JXTA Relay is present in their scope. Anyway, the architecture of the JXME network is far from the theoretical model of the peer-to-peer structure, where all of the members have the same rights and duties. Furthermore, the predominant position of a Relay causes some risks that should not exist in a peer-to-peer network. In fact, the fall of any peer should not affect the overall stability and effectiveness of the network. Conversely, if a Relay falls, and no other Relay is close to it, this causes the eclipse of the portion of JXME network that relied on this Relay, as shown for instance in Figure 4; even if this portion of network is active, it cannot be reached. Furthermore, if the functions of a specific Relay are not restored, the peers that rely on it lose their state.

Taking into account the above mentioned features, in our opinion, the JXME architecture is not appropriate to design a pure peer-to-peer network, above all if MANET-like environments are to be considered. These remarks lead to the purpose of the work presented in this paper: to create a framework for peer-to-peer computing with mobile devices and limited resources; this framework should be compatible with the JXTA environment, so to rely on a wide range of applications.

### 3 Design issues

In this section the design issues that have been faced with to meet the requirements and the restrictions above mentioned will be described. In particular, the main issue will be the compatibility of the newly designed environment with the JXTA architecture. These are the main points that will be described:

- Definition of the ways of connection for the peers, of the carrying level, and of the communication messages.
- Univocal identification of a peer and of the resources contained in the network
- Organization of peers into groups.
- Way of searching for the resources in the network.
- Creation and destruction of the resources in the network.
- Way of routing without a direct visibility of the devices concerned.

Some issues arising from these points are only porting issues of JXTA features to the J2ME environment. However, as will be described throughout this Section, in many cases new solutions have been found in order to comply
with the requirements of the limited resource devices involved and of their mobility.

3.1 Network connection

The first issue was to determine how some peers within the communication scope can find each other, in order to share any resource. Once a peer is activated, it regularly sends a multicast message containing its address. If another peer is present within scope, it receives the message and stores the information contained in the message; then it replies to the sender peer, by using the address received. It indicates its address in the reply, so that it can be known and stored by the peer that sent the multicast message. Once the reply is received, both peers can communicate directly. Now the frequency of the multicast message is prolonged, in order to optimize the use of the bandwidth. If a peer is alone again (due to the disconnection of the other known peers), the operation of initial search is repeated. This choice is slightly different than JXTA, where the issue of the bandwidth optimization does not exist. Due to the limited resources of the devices used, the design of the transport level has been very difficult. The standard protocols defined in JXTA are TCP and HTTP. The resources of the devices we analyzed allowed to use TCP only; furthermore, they were able to send by using HTTP, but not to receive. The first phase of our work has been the development of a light http server that could provide the features needed for the communication with JXTA peers. Then we have analyzed the structure of the messages exchanged among the JXTA peers, in order to outline their characteristics and to reproduce them in our transport levels. Finally, in our environment we have outlined the sequence of messages that form a correct communication between two JXTA peers. With reference to the messages used by the peers for the communication, we could use the structure of the standard JXME messages. However, we had to analyze the structure of the JXTA messages, in order to create messages in our environment, and to make them compatible with the ones used by the JXTA services.

3.2 Peers’ identification

In order to identify a peer univocally in the network, an identifier independent from the transport protocol is needed. In order to keep the compatibility with JXTA, the identification method used in such environment has been adopted. However, since the resources are limited, and since the identifier used in JXTA contains common information for all the addresses of this type, only the variable part of the JXTA identifier is locally stored as an identifier element of the peers. Of course, the identifier is reconstructed during the transmission.

3.3 Peer groups

In order to organize the network, to facilitate the search for resources, and for security reasons, the peers can organize themselves in groups. Since each group is assimilable to a resource, it is identified with an identifier. The creation of the group can be free, so each peer is free to take part in the group with no need for an authorization from the other peers. The creation of the group can also be protected, when the membership in the group is managed with an authorization protocol. Once each peer is created, it is registered as a member of the NetPeerGroup base group, in order to save the compatibility with the JXTA architecture. This is the group all the peers of the network belong to. All the peers that belong to the NetPeerGroup use all the basic services of JXTA. Some classes have been implemented for the creation of new groups and for associating the peers to these groups. Each new group benefits from the base services of our environment, as well as using user-defined services. Furthermore, considering the poor resources of the devices, the new peer is enabled to activate the base services selectively. For instance, if a peer is equipped with more than one transport protocol, it can decide to activate just one of them, in order to optimize the resources. Each service of the core (i.e., the services that must be activated when the new peer is initialized) can be redefined by the user, in order to obtain a more versatile architecture that can be more suitable to the device’s resources.

3.4 Resource Discovery

Due to the lack of a central server (where the resources can be registered), the peers of a MANET need to exchange information dynamically. Each time a peer is created, one of its first actions is the creation of the Advertisements. They are documents whose purpose is to make
a resource visible in the network. In fact, they contain all the information needed for the univocal identification of the resources, and for using them. When a peer wants to find a specific resource in the network, a message is sent, which is received by the other nodes of the network. They reply by sending the advertisements they have, if the search parameters contained in the request are met. Once the advertisements are received, the peer analyzes and stores them for their future usage. JXTA uses XML advertisements. This format is very easy to use in fixed devices; conversely, this is a handicap for portable devices. In fact, XML messages have a high portability, but require too many resources both for the analysis and for the creation and storage. One might not think of storing the XML advertisement on mobile devices (whose resources are poor), so a binary format has been defined and each advertisement is converted in this format before storing it. Indeed, a new service, whose purpose is just the management of advertisement, has been introduced with respect to JXTA. This new service does not only convert the advertisements from XML into the binary format (while they are being received); it also manages a database that stores and searches for the advertisements, as well as converting them from the binary format into XML during the transmission. It has been necessary to define some classes (apart from the above-mentioned service) that deal with the conversion of the different advertisements, so to make the architecture expandable and updatable in case of changes in the structure of the JXTA advertisements.

During the storage phase another field to each advertisement has been associated, which limits its life period inside the local cache. By accurately tuning this parameter in MANET environments, an optimal tradeoff can be reached between messages sent in the network and accuracy of network resources information locally stored. The new service regularly analyzes the advertisements, and deletes the ones whose life time has expired. In fact, when a service wants to use a resource, it first performs a local search (by analyzing the local database of advertisements), and then (if the resource is not found because its TTL has expired) performs a remote search.

3.5 Resource life cycle

When a resource is created, the corresponding advertisement must immediately be created. Otherwise, the resource is not visible in the network. There is no specific procedure to destroy a resource in JXTA. In fact, one simply needs to delete the advertisement locally (so that it is no longer visible), and wait for the expiration of the advertisement lifetime in other peers; this way, a resource disappears and is no longer available in the network. The same procedure has been used for our architecture.

3.6 Routing issues

When we deal with mobile peer to peer networks, we refer to a structure where each node can communicate with the nodes that are inside its radio coverage area. If two peers are not directly visible, multi-hop communication must be managed. The routing algorithms that can be used in a MANET network can be classified in two categories [22]:

- **Proactive routing:** a node using this routing algorithm has a complete vision of the network at any time. This vision is saved in a table called *Table Drive Routing*, which contains any path that can be used to reach all the nodes of the network. Each time the network changes (and the contents of the table consequently change), the nodes exchange the new table, so to update the paths. This routing algorithm is very fast, because we simply need to extract the data of a path directly from the table for each peer. Conversely, this algorithm requires too many resources for the storage of the data, and takes too much bandwidth to exchange the tables.

- **Reactive routing:** this algorithm does not use tables for the storage of the paths, because each node sends a *Route Request* to all the nodes of the network, when it wants to reach a peer that cannot be seen, and indicates the destination to be reached. Each recipient node sends the message to the known nodes, until the destination node is reached. This way, a reply message called *Route Reply* is created, whose path is exactly the opposite as the one of the request from the sender node, and contains the routing information. The advantages of this algorithm are a limited use of resources, as well as a limited
use for the transmission bandwidth; the main disadvantages are the time needed to search for the path, as well as the risk of loops.

The JXTA environment uses an adaptive source-based routing [6], which is a compromise between these two types of algorithms. In order to describe its basic operation, we need to describe the concepts of Relay Superpeers and of Edge Peers. As we have already said, the former are specific peers that provide a connection service to the nodes of the network that cannot see each other, while the latter are the peers with ordinary features and resources. The Relay Superpeers use a proactive routing algorithm in JXTA. They store some tables with a detailed vision of the network, and keep them available to Edge Peer. If an Edge Peer needs the path to reach a peer that cannot be seen directly, it sends its Route Request. This request is directly sent to the Relay, if at least a Relay can be reached by the requesting node, which sends the desired path to the requesting node. Otherwise, the request for the path is spread towards all of the nodes of the network, which send it to the other known nodes, until the recipient is reached. Thus, the Edge peers use the reactive routing algorithm. Furthermore, once the reply is received, the routing information is stored as an advertisement, and is therefore related to a maximum life time to allow to use the information in case of need.

In our architecture a reactive routing algorithm has been used, due to the poor resources and to the need for compatibility with JXTA. In particular, messages of request and of reply of JXTA have been reused. As we have already said before, a loop can occur during the propagation of the Route Request. To avoid this inconvenience, some techniques can be used, which help to solve the problem. According to one technique, each peer does not send the request to its sender. According to a second technique, a Time To Live is included in the path request; this determines the maximum validity period, after which the request is deleted from all the receiving peers. Finally, the maximum number can be determined for the hops that the request can do from a peer to another, before being deleted. We need to outline that the modularity of our environment allows the user to replace the used routing algorithm with another one that can be more suitable to his/her needs. Furthermore, our main purpose was not to design a routing algorithm for MANET, because during the last few years the research has become very active in this area [22].

In the next section we will describe the structure of the new environment. We will also show how it has been created, showing the advantages and the obstacles in the use of the different technologies.

4 System Architecture Implementation

Our architecture has been created in a modular way, in order to make it suitable to any device. For instance, we did not implement the Bluetooth transport protocol, since it is not yet a standard for wireless communications, due to the customizations made by the different manufacturers. If we want to add it, we will simply need to define the new transport level, and register it in the manager of the connections. The modularity makes the structure versatile. This is a very important aspect for the structure, considering the rapid evolution in the market of mobile devices, as well as the need to update the architecture, in order to avoid to make it obsolete. Furthermore, the modularity makes maintenance simpler and less expensive.

The diagram in Figure 5 shows the overall architecture of the new environment. At the lowest level there is the Virtual Messenger, which belongs to the core of the architecture together with the Peer, PeerGroup, Advertisement and ID Management modules. At an higher level we find the Resources Discovery and the Pipes Management services. Several applications can be found at the highest
level. Among which we have indicated - for instance - Instant Messaging and File Sharing.

4.1 The Virtual Messenger

By Virtual Messenger, whose components are shown in Figure 6, we mean the transport and service protocols that manage the communications of the peer with the rest of the network. A central role in the management of communications is played by the Endpoint Service, whose purpose is to abstract the physical addresses of the peer into a logic address, as well as to manage the transmission and the reception of the messages. This way, the services of our environment do not need to know the details about the transport. The transport protocols are at the basis of the Endpoint service, while other services are defined at higher levels. They can provide features aimed to make the communication service more sophisticated.

Figure 6: Virtual Messenger: the core of communications

4.1.1 JxmeTransport Interface

Let us start analyzing the transport protocols, doing a bottom-up analysis of the Virtual Messenger. These protocols have the same Java interface, called JxmeTransport. In case we want to implement a new transport protocol, we simply need to define a class that creates the interface we have mentioned, and to register it in the Endpoint Service, as we will show below.

- **HTTP and TCP protocols** The HTTP and TCP transport protocols implemented in our architecture have been created respecting the compatibility with the Jxta protocol. Thanks to the support provided by the sockets, in the case of the TCP protocol we managed to benefit from the opportunities offered by J2ME. With reference to the HTTP protocol, we experienced the restrictions fixed by the MIDP profile of J2ME, which supports the possibility of creating some connections to an HTTP server, but does not support the possibility of accepting incoming HTTP connections. We therefore needed to create a light http server to manage such connections.

- **Datagram protocol** In order to respect the compatibility with JXTA, we experienced the limits fixed by the J2ME CLDC platform, which does not support multicasting. Since JXTA uses multicast messages both while the protocol is activated (to search for other peers) and each time a new resource is searched for, we needed to search for an alternative solution. To compensate for the lack of the multicast, we have used the broadcast, which is a transmission system that allows to send a message to all the devices connected to a network. This way, we have managed to send a single packet that could be received by all of the nodes belonging to the network. The multicast used in Jxta can receive any broadcast message sent through the network, as well as the specific messages. Using this characteristic, we have simply needed to mask the datagram address with the multicast address used by JXTA. However, the mobile peers of our architecture cannot receive the multicast messages sent by the JXTA peers. To solve this problem, each new resource created in our environment is published remotely; this means that each new resource is published on the peer that owns the resource, and on the peers known to it, including the JXTA ones.

4.1.2 Endpoint Service

Like all the services defined in our architecture, the Endpoint Service implements the Service Interface we have defined in the new environment.

These are the main tasks carried out by the Endpoint Service:

- **Registration of transport protocols:** Once the class that manages the protocol is implemented, it must
be registered in the Endpoint Service, to be used for sending and receiving messages.

• Management of messages transmission: The Endpoint Service provides some mechanisms to spread the messages in the network, and to check whether the destination peer can be reached or not.

• Management of the reception of messages and their sorting to the different services: In order to receive messages, the services need to implement their Endpoint Listener, and register it in the Endpoint Service with the name and the parameters to be used to identify the listener for a specific service. According to the destination parameters contained in the incoming message, the Endpoint Service checks whether the destination service is registered. If so, the message is forwarded, otherwise, it is deleted.

4.1.3 Endpoint Routing

The purpose of the Endpoint Routing Service is to manage communications between two peers that are not directly connected to each other. The service:

• searches for a path to reach a specific peer;
• reply to path requests coming from other peers;
• routes the messages sent by the peer to their destination.

The first case occurs when the Endpoint Service is requested to send a message to a recipient identified by its logic address, instead of its physical address. In this case, the Endpoint Service searches for the Route Advertisement of the recipient peer in the database of advertisements, in order to route the message correctly. If the Route Advertisement is not present, the service leaves the management of this event to the Endpoint Routing. It includes the unsent message in a queue, and then makes a Route Query in the network, trying to obtain the information needed to reach the recipient peer of the message. The recipients of the query send the request to the other known peers, until the recipient peer is reached. If the destination is not reached within a fixed Time To Live or a fixed number of Hops, the request is cancelled. Once the destination is reached, the query receives a response that follows the reverse path of the request. This response, called Route Response, contains a Route Advertisement with a sequence of nodes representing the path from the sender to the final recipient. The queue of pending messages is managed in a way that allows to keep the message in memory for a fixed time; if the response to the query is received within this time, the message is sent, using the information provided by the response. Otherwise, the message is deleted. Each new path acquired through a Route Query is stored for some time, in order to allow to use the route again within a short time.

The second case occurs when the peer is requested for information (through a RouteQueryMessage) to reach a destination. If the peer that must be reached is the peer itself, the Endpoint Routing Service creates the RouteResponseMessage, and sends it to the requesting peer. Conversely, if the peer is not the destination one, the Endpoint Routing Service sends the query to all its known nodes.

The third case occurs when the Endpoint Service receives a message whose destination is denoted with a logical address, instead of a physical address. In such circumstances, an element called EndpointRouteMessage has been added to the original message. This element contains the information needed to reach the final destination. If the peer is itself the recipient one, the Endpoint Routing Service sends the message straight to the service involved; otherwise, it identifies the next node of the forwarding list contained in the Endpoint Route Message, and sends the message again.

While creating our environment, we preferred not to opt for a more complex routing algorithm, in order to avoid to make the architecture too heavy. Thanks to the modularity of the environment, this algorithm can be replaced with any other routing algorithm considered appropriate by the user. The compatibility with the JXTA environment is respected in any case.

4.1.4 JxtaPropagate Service

The JxtaPropagate Service allows to send messages within a group with the technique of flooding. This communication method is widely used to make some queries, or when all the peers belonging to a group should receive a specific message. To respect the compatibility with the JXTA Propagate Service, the format of the RendezVous-PropagateMessage has been adopted. This message is
added to messages that must be propagated, and contains:

- the indication of the manager that must receive the communication;
- the latest peer sending the message;
- the TimeToLive, that is, the maximum number of peers that must be crossed by the message;
- a MessageID that identifies the message univocally.

In order to limit the looping, the following action are taken:

1. the message is no longer propagated once its Time To Live (that is reduced each time the message is sent) is reset; this is the most important mechanism;
2. the message is not sent back to the peer it came from;
3. finally, we have used the MessageID to filter all the messages already propagated by a peer.

4.2 Advertisement Management

Advertisements are the only way to know the resources made available to the peers in a network. All of the advertisement in JXTA are represented in XML, so to allow a good portability. J2ME has no built-in support for XML, so we needed to adopt a parser that does not make our architecture too heavy. We therefore selected the open source kXML parser [16]. Our next step in the designing of advertisements management has been to find an appropriate storage method. In fact, it is unfeasible to store the advertisements in XML, due to the high amount of memory that should be used. A binary format has been defined to store them in a local cache. Finally, we have analyzed the issue of transmissions among the peer. Since the JXTA peers send and receive advertisements in XML, we have needed to adapt to this condition, notwithstanding the storage mode of advertisements. Consequently, we have needed to reconvert the advertisements from a binary format to XML.

4.2.1 Advertisement Service

The Advertisement Service is responsible for the management of the advertisements, and in particular, it deals with all the aspects concerning their creation, storage, and search. It is a service not present in JXTA, but needed in our environment in order to manage advertisements resource limited devices. Each time an advertisement is created, it is registered in the Advertisement Service, which becomes aware of the advertisement, and can therefore manage it. This way, the set of implemented advertisements can easily be expanded, since the run-time registration is always possible. The registration is necessary because the XML format of each advertisement consists of a single inner structure; the conversion from XML to the binary format is therefore different for each advertisement. The registration allows the Advertisement Service to know the class that implements the encoding from XML to the binary format, and vice versa, for each advertisement.

Advertisement Storage

The advertisements are stored in the local cache, organized as a vector consisting of three fields: the first field is the binary format of the advertisement, the second field is the class of membership (this field has been created in order to make the search activity faster), and the third field is the Time To Live of the advertisement. The last field (TTL) indicates the lifetime of the advertisement in the local cache. It is different from the TTL contained inside the advertisement itself, which instead represents its lifetime in the network. The Advertisement Service, after checking that the corresponding class has been registered, inserts the binary representation of the advertisement in the local cache. If the class has not been registered, the Advertisement Service registers it. However, the storage depends on whether the advertisement being stored is present in the cache. If the advertisement is already present, it is replaced with the new advertisement, which is considered an update.

The final step in the management of the database of advertisements consists of searching for them, and taking the corresponding data. Each time one of these operations is done, the archive is analyzed, and all the advertisement with an expired Time To Live are deleted. The choice of using a binary format for the local storage of advertisement relies in the considerable saving of memory resource it can be obtained. In Figure 7, we show that by using a binary format, in some carried out tests, a 70% reduction of storage space was observed. A conversion from the binary to the XML format may seem unnecessary, consid-
ering the context where we operate. However, in order to respect the compatibility with JXTA, each advertisement must be sent in plain XML format.

Figure 7: Comparison of storage requirements of Peer, Pipe and PeerGroup advertisements in XML and binary formats

4.3 Peer and PeerGroup

A peer is mainly identified through its identifier (PID) and its group (GID); the other information are: the name, a short description, and a vector that stores the endpoint addresses that allow to reach the peer. When the peer is created, both the name and the description are set to a standard value (The two fields can be later specified by the user), while a new univocal peer identifier will be automatically created. The new peer will receive the identifier of the JXTA base group (jxta-NetGroup) as a group identifier. The PeerGroup Service is a central point of our environment, because it allows to start the services and the protocols that should be used by our peers. Furthermore, the PeerGroup provides the methods needed for the management of JXTA groups, such as the creation of the group or the join of a peer to a group. The activation of our environment corresponds to the creation of the NetPeerGroup, which is the group each newly created peer belongs to since the beginning. In fact, the Net Peer Group provides the set of minimum functions to allow a peer to take part in the activity of a Jxta network. Then a peer may decide to belong to other groups, where some services with specific features can be present. These features may not be essential for the operation of the environment, and may not be included in the set of base services for this reason. While designing the service that manages all the aspects of a groups of peers, we have also dealt with the following issues:

- activation and deactivation of a service within the group;
- creation of the user peer;
- activation of the transport protocols within the group;
- publication of the advertisements that describe the resources of the NetPeerGroup;
- operativeness for the creation of a new group, and to make possible the connection to and the disconnection from it.

4.4 Resolver Service

An interesting aspect of peer to peer networks is the way of publishing and searching for the information that describe the resources and the services. The function of the Resolver Service is to provide a base tool to search for the information within a group of peers. The service is designed in order to be a starting point to define more sophisticated search services. Like any service proposed in JXTA, the Resolver Service is based on sending and receiving specific messages. Each message includes a "query message", or a "response message", whose syntax is defined by the service that uses it.

To clarify how this service works, let us assume to define a new service to search for a specific file in the network, and to call it DiscoveryFile (see Figure 8). The DiscoveryFile service communicates with the same service of other peers, using a message whose structure is:

```
<jxta:DiscoveryFileQuery xmlns:jxta="http://myDef.org">
  <Type> File </Type>
  <Name> file.txt </Name>
</jxta:DiscoveryFileQuery>
```

This message is passed to the Resolver Service, which adds it in the query message and sends it to the other peers. When the query message reaches its destination, the Resolver Service of the recipient peer extracts the information contained in the specific search message, and sends it to the service responsible for the management of the query.
Each query message is addressed to a "manager" characterized by specific semantics of the query and response messages; however, this manager is not associated to a specific peer. The same query can be received by an unlimited number of peers. Each of them can process it, provided that the peer has defined an appropriate "manager". If we want to define a service that can work as a "manager" in our architecture, the first step is to define the semantics of the query and response messages that we want to use. After that, our "manager" service must be defined, so that it can implement the interface QueryHandler. This interface that we have created allows the services that implement it to receive the messages that reach the Resolver Service, and that belong to them.

The Discovery Service

The Discovery Service is based on the above-mentioned Resolver Service. It is the service that searches for and publishes the advertisements in the group. Due to the poor resources of the considered environment, we have activated only the main features of the original service present in JXTA. The service searches for and publishes the advertisements remotely, as well as publishing the advertisements locally. The remote publication is essential in our environment, because (as we have already said) our peers cannot receive the multicast messages to search for the resources sent by the Jxta peers. Consequently, our peers must make their resource public as soon as it has been created, notifying it to all the known peers. For instance, we have decided that both the Peer Advertisement and the Route Advertisement should be published remotely when the platform is activated, in order to enable all the peers of the network to know about the presence and the physical address of the peer that is being activated. Once a request for a resource is received by a peer, it will directly reply to the requesting peer.

5 Example application

In order to provide an example of how our architecture works, we have created an application that stresses the self-organization and the absence of central structures in our system. For this purpose, we have developed a card game, using both the potential of our architecture and the new graphic libraries provided by J2ME. The main steps of this application involve:

- starting up the platform and the application;
- gathering of the players;
- dealing the cards;
- cards playing;
- ending.

In order to start the framework, a configuration dialog box has been created, which allows to set up the main working parameters. Like any JXTA application, our application is based on sending and receiving asynchronous messages. These are the messages used in our application: GameCardSession (GCS), GameCardLink (GCLK), GameCardOpenGame (GCOG), GameCardPlayerCard (GCPC), GameCardPlay (GCP), GameCardLeave (GCL).

Once the platform has been started, the player who wants to start a game simply needs to enter the name of the session and the name that he/she wants to use during the game. From now on, he/she will be visible in the network as a player ready to start a game. The name of the session is needed, in order to allow to play several games in the network at the same time. If another user of the network wants to join in, the active playing sessions in the network are displayed on its device, and he/she can...
therefore choose the game he/she wants to take part in. However, the user can take part in the game only if this is confirmed by the player who started the game.

In Figure 9 the steps and the exchanged messages involved in a typical game session are depicted. In order to make all the peers of the network visible, the GameCardSession message containing the name of the session and the JxtaID of the sender peer is broadcasted (step 1). The next step consists of sending the request to join the game session (step 2). This message is called GameCardLink, and contains the name of the player who is asking to join, as well as his/her JXTA identifier. After this message is sent, the would-be player waits until the beginning of the new game. When the peer who has created the game session has received this message, it includes the new player in a vector called Players, which is used to determine the turn for the game. We want to outline that the JXTA identifiers are used to send the messages from a peer to another. This opportunity provided by our architecture allows to not bother of the physical addresses of the players, because the EndpointRouterService has the task of sending the messages correctly to their destination. When the game is started, the peer that has created the session sends the message GameCardOpenGame to each player (step 3). This message contains the vector Players, in order to determine which player plays first, and the remaining playing turns.

Then the message GameCardPlayerCard is sent, which contains the cards open on the play-table (Figure 10a). Once the game is started, each card that is played (Figure 10b) is shown to the players of the virtual play-table through the message GameCardPlay. This message contains the ID of the player who has just played, the card that he/she has played, any cards he/she has drawn, as well as the ID of the next player. The player who has the next turn is enabled to play according to this ID.

5.1 Quantitative Evaluation

After implementing the example application, we made some quantitative evaluation about the resource usage by the application itself and by the developed framework. The diagram presented in Figure 11 shows the Java classes and the packages used in our system, together with their size (in KBytes). In particular, we show the framework size and the total size of the application, making a distinction between the classes of the application logic and those related to the management of graphics.

We can easily understand that the graphics takes more
than half of the size of our application. Furthermore, among the instructions for the application logic, the ones concerning the communications among the devices through our framework do not exceed the 6% of the total. This is due to the fact that all the communication and low-level issues (discovering peers and resources, sending messages, etc.) are provided by the underlying framework. In spite of these functionalities, its byte code sizes only 123KB, which is an acceptable dimension for todays J2ME enabled devices. In our opinion, this proves that our architecture is a valid tool for the management of Jxta networks.

Another evaluation can be done about the amount of data that must be stored by the device, in order to make our framework work properly. Thanks to the use of the EndpointRouting Service, the only information that must be saved by our peer is the Route Advertisement of each participating peer. This information takes about 150 bytes of memory; this quantity is compatible with the resources available to our devices. Finally, we remind that since our architecture is fully compatible with Jxta, even a user owning a device equipped with full JXTA may transparently play the game that we have provided as an example.

6 Conclusion and future work

In this paper a middleware for mobile P2P computing on handheld devices has been presented. Its key features include the interoperability with JXTA protocols, the overcoming of JXME architectural constraints, and the ability to work even in ad-hoc configuration, when no connection to a fixed network exists. The implementation work showed the viability of the approach, both for its technical soundness and for its effective management of resources. The quantitative evaluation of the application showed the low requirements in code dimension needed for applications built on top of our framework. Our current work includes more extensive testing of the framework in order to evaluate its performance with respect to different parameters such as network traffic generated (both unicast and broadcast), latency of advertisements propagation, performance of discovery mechanisms and so on.

A parallel line of research includes investigation on some key and challenging issues for mobile P2P computing: routing, discovery algorithms, support for different communication protocols and effective security mechanisms. Experimental work on these issues will be simplified and allowed by the modular architecture of the framework: each component can be indeed easily replaced, while maintaining the same set of services offered to the other components. Efforts can be centered on the new functionality to be added or to be enhanced.

References


