Design and development of a Jxta middleware for mobile ad-hoc networks

Mario Bisignano, Andrea Calvagna, 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,Andrea.Calvagna,Giuseppe.DiModica,Orazio.Tomarchio*}*@diit.unict.it*

ABSTRACT
The combination of personal computing devices and wireless ad-hoc networks allows the concept of mobile ad-hoc information system, consisting of a highly dynamic, decentralized and self-organizing network of autonomous and mobile devices that interacts as peers. Application developers have to deal with a new set of problem peculiar of these systems, due to user and terminal mobility, to low bandwidth, to transient loss of connectivity and to lack of centralized infrastructure. Starting from an existing open software framework for P2P systems, JXTA, in this work we tried to define a new middleware, named Expeerience, to face with these problems. The introduction of a mobile code service is one of the main innovation introduced, allowing the dynamic services deployment at runtime. An example application is finally presented, showing the benefit of the new middleware mechanisms introduced and the low overhead needed for developing an application for ad-hoc environments.

KEY WORDS
MANET, P2P, mobile computing, JXTA, mobile code

1 Introduction

The recent advances in the development of affordable and portable wireless communication and computation devices has fostered a tremendous amount of research on wireless mobile ad-hoc networks (MANET) [15, 9, 4]. The convergence of wireless, cellular, and PDA technology presents many new challenges in order to make the *"anytime, anywhere, anyplace"* computing paradigm [8] real and effective. Thus, in the very next future, one can envision scenarios where the "world" is composed of several islands where all participants are connected by wireless networking in an ad-hoc manner sharing some form of synchronization, while some participants in each of these islands might occasionally be connected by cellular or wired connections to the rest of the Internet/world [8, 10]. New research challenges arise, since many of the classical hypothesis that are possible in traditional distributed systems do not hold anymore in these new contexts [4, 18]. From a practical and market perspective, the relevance of this kind of mobility is not very clear, even if different industrial interests are moving around this kind of issues [9]. Indeed, MANET seems very well suited for all those scenarios where a fixed and wired network infrastructure cannot be installed, both for economical and practical issues. A typical example can be found in military applications, in hostile or unexplored territories, or in a civil scenario, in emergency situation for disaster recovery application [13, 4]. Different scenarios, more user-oriented, include the so-called *"impromptu meetings"* such as mobile conferencing in classroom or business-meeting environments, where a group of people meet in a remote locality and need to collaborate, by exchanging information, with their personal computing devices [5, 11]. Other kinds of scenarios include many entertainment applications, for instance, for exchanging audio files or for multiplayer games sessions. All these new classes of applications pose new challenges to software developers [3]. In particular, the resources are limited in terms of available central memory, mass memory, CPU speed and battery power, and therefore need to be used effectively. The network connectivity can be suddenly and intermittently stopped. Furthermore, the available bandwidth is typically smaller than the one available in fixed networks. All these issues make an adequate layer of middleware software necessary. This layer should hide the complexity to the programmer, making things easier, and allowing the development of applications that can fully use the peculiarities of such environments.

It is evident that, for these classes of systems, applications cannot be designed according to the conventional architectural paradigms, such as for example the client-server one. On the contrary *"peer-to-peer"* architectures are needed [17], where each single host is able to play the role of server and client according the user's needs. The existing approaches are limited to wired infrastructure when considering P2P environments [14], or oriented to specific applicative areas like file sharing[6, 7]. Some of the existing approaches for peer-to-peer (P2P) computing provide a flexible and quite general set of abstractions to operate in wired and fixed infrastructure [16]: however, as we will see in the next section, they do not generally consider the dynamic reconfiguration of the network and/or the dynamic joining/leaving of active hosts.

This work goes towards this direction, trying to define a middleware layer that can answer to these issues: while doing this, we have decided to adopt as a basic software framework, an emerging P2P open technology such as JXTA [16]. The choice of JXTA allowed us to use the existing implementation of common P2P concepts: its modular architecture allowed to operate only on those mechanisms needed in order to address specific MANET requirements, such as the dynamic joining and leaving of peers to a network, and the low reliability of wireless network link. The second issue we address in this work is the integration of some code mobility support in the developed middleware. In fact, one of the aspects that a middleware for mobile computing should have, is the possibility of dynamically deploying services at runtime. This is the capability of distributing and executing services on peers that originally do not own the service code. This has been introduced in our platform by adding a *mobile code* service, which allows to download and install new services dynamically only if necessary. They can be downloaded either by a service provider or by any peer connected, which has such service. This way, all of the mobile code paradigm advantages for mobile computing [2, 12] are also available for MANET environments, greatly enhancing their potentialities. At the same time, the code mobility allows to adapt the behavior of the middleware dynamically to situations that were not considered during the design, and that take place only during the run-time. If such issues can be solved by preloading the middleware with all the possible scenarios, this cannot be done in some environments where the devices (as we have said before) often have limited storage resources.

The rest of this paper is organized in the following way: in Section 2 we describe an overview of P2P and MANET systems, outlining the similarities and the key differences as far as middleware is concerned with. In Section 3 we present the developed platform, named *Expeerience*, outlining the additive modules to the existing JXTA architecture and describing the code mobility module. Then in Section 4 an example application is provided, in order to test the framework and to synthesize the major benefit provided by Expeerience. Finally, we conclude our work in Section 5, giving also a roadmap for our future work on related issues.

2 AdHoc and P2P middleware

Peer-to-peer systems A peer-to-peer (P2P) system can be considered as a distributed system without any centralized control or hierarchical structure,where each node has equivalent functionality and responsibility [17, 14, 1]. Although there is not actually a universally accepted definition of peer-to-peer computing or networking, in general P2P refers to an environment where hosts (peers) connect to each other in a distributed environment that does not use a centralized control point to route or connect data traffic. In this context, each node (peer) acts as a server and as a client, as understood in the common sense, at the same time. This is very different when compared to the traditional client/server computing paradigm, in which one or more computers are dedicated to serving the others. In practice, P2P technologies deployed today are mainly used for file-sharing purposes and adopt a network-based computing style that neither excludes nor inherently depends on centralized control points [6, 7].

Mobile Ad Hoc Networks Ad-hoc networking refers to the spontaneous formation of a network of nodes without the help of any infrastructure, usually through wireless communication channels. Example of MANET nodes are various type of devices with different computation and communication capabilities. In ad-hoc networks, a basic routing infrastructure emerges through the collaboration of every node with its neighbours to forward packets towards chosen destinations. This basic infrastructure is highly dynamic not only because of node mobility but also because of lacking of guaranteed connectivity. In ad-hoc networks, the lack of guaranteed connectivity is caused by the limited-range, potentially unreliable, wireless communication. The absence of a routing infrastructure that would assure connectivity of both fixed and mobile nodes precludes using the traditional internet protocols for routing, name resolution, trust establishment, etc.

Key differences and similarities Currently developed mobile ad hoc systems are not supported by any middleware, but rather rely on each application to handle all the services they need. This constitutes a major complexity/inefficiency in the development of MANET applications. Most of MANET research efforts so far are still focused on the network, link, or physical layer, while application and system level issues have not received sufficient attention, in our opinion [13]. Compared to wired networks, middleware design and development for ad-hoc networks will be strongly influenced by considering intermittent network connectivity as a normal lifecycle and not as an exception. Moreover, hiding network topologies and other deployment details from distributed applications become both harder and undesirable, since applications and middleware should adapt according to changes in location, connectivity, bandwidth, and battery power [3]. Considering the technologies currently used,the most appropriate paradigm for dealing with the mobility of mobile devices is (in our opinion) the Peer-To-Peer one [14, 17, 1]. Both systems, P2P and MANET, share the key characteristics of self-organization and decentralization, and both need to solve the same fundamental problem, that is, how to provide connectivity in a completely decentralized environment. Conversely, they present some important differences, in particular with respect to:

- **Dynamic network topology**: Both have a flat and frequently changing topology, caused by node join and leave in p2p overlays and MANETs. But MANET are

characterized by additionally node mobility, so that the node's access point can change dinamically.

- **Hop-by-hop connection establishment**: Per-hop connections in p2p are typically made via TCP links with physically unlimited range, whereas per-hop connections in MANETs are via wireless links, limited by the radio transmission range.
- **Scalability**: in the case of Peer-to-Peer, it is limited by additional high user data rates. In ad-hoc networks the limit is due to the physical constraints of the mobile terminals, i.e. low bandwidth and low processing power and in both systems by bandwidth consuming signaling traffic.

Apart from the traditional challenges of P2P like decentralization, self-organization and unreliable peer availability leading to topology dynamics, it is imperative that any p2p middleware built for MANETs should account for the peculiarities of these environments, particularly regarding resource constraints like memory of portable devices, bandwidth, power, low computation capability, unpredictable (dis)connectivity and dynamics of topology because of peers mobility patterns.

3 Design of P2P middleware

Our main purpose in the design of Expeerience [19] was to use the services provided by a P2P environment, in order to quickly build a working prototype on top of which the various modules needed for MANET environments could be developed and tested. The initial choice as the basic P2P framework for our work was JXTA [16], due to several reasons. The key objectives of the JXTA project are interoperability, platform independence and ubiquity. Starting from these general sets of objectives, project JXTA is creating a peer-to-peer system by identifying a small set of basic functions necessary to support generic P2P applications and providing them as building blocks for higher-level functions.

Our idea is therefore that of creating another software layer on JXTA, in order to meet all the requirements that characterize the MANET, networks, and that cannot be met by JXTA. On one hand, some changes have been made to the original setup of JXTA core. On the other hand, new JXTA services have been introduced, which make new features available to the developers of services for MANET networks. The new middleware introduces these new features: management of the intermittent connections and of multiple physical interfaces, increase in the potential of resource discovery, and code mobility service. Figure 1 shows the basic ideas of our middleware framework: in particular, the changes to the core layer and the new added services have been highlighted. Although based on JXTA, the design is such that the user's application entry point is provided only by the **Engine** interface, which offers a complete set of APIs. The Engine is currently built over JXTA, as we will describe later, but in the future it could use a different P2P framework, if JXTA constraints should limit us too much. In the following sections, a more detailed description of the new services is provided.

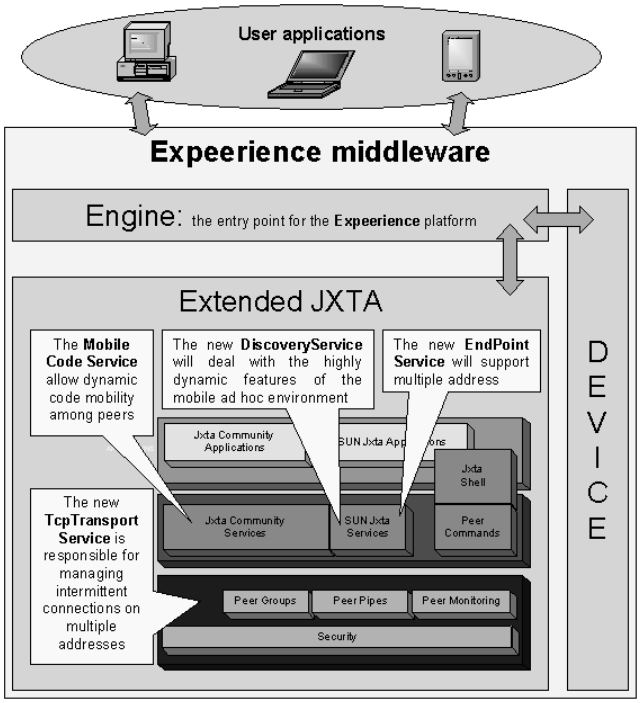

Figure 1. Expeerience architecture

3.1 Disconnection management and Resource discovery

Disconnection management is one of the feature we have introduced in our middleware. As already stressed throughout the paper, MANET environments are characterized by the unpredictability of mobile node, as well as the unreliability of the wireless links. A typical scenario is that of a node that, while moving and communicating through the wireless interface with another peer belonging to its community, looses the radio signal and is forced to rejoin the community through another access point, whether it is a wireless one or a wired one. Furthermore, in an environment that lacks of any fixed infrastructure, and the peers are equipped with different network interfaces (someone owns a wireless one, someone a wired one, someone else both of them), a functionality for "bridging" the wireless peers to the wired ones is needed. Since the described scenarios are quite common in the considered environments, a robust and effective mechanism is needed, that will have to accomplish the following tasks:

- **Intermittent connectivity management**: The new feature will enable the peers to leave the peers' community and rejoin it through another access point,

while keep granting the connectivity. The whole process is transparently handled, in the sense that the peers in the community will not realize that the peer is attached to another access point.

- **Multiple interfaces management**: Thanks to this functionality, the peer will be capable to handle the network interfaces it is equipped with. If needed, a peer equipped with a wired and a wireless interface can act as a bridge between two peers (or two group of peers), each one provided with only one kind of interface respectively.

- **Enhanced Resource discovery**: The Resource discovery service has been accurately enhanced in order to cope with the frequent disconnections and the subsequent reconnections. In the new scenarios, carachterized by uncertainty and unreliability, the new Resource discovery service provides more precise and updated information is needed about the peers and the services that they offers.

The JXTA platform (in the version 1.x currently used) does not support either the use of multiple network interfaces for each peer, or the assignment of more than one address to the same interface. This implies that a peer cannot communicate through more than one channel, nor can it use whatever protocol it chooses. Let us focus on the scenario shown in Figure 2, where three devices are depicted.

The devices A and B are attached to a wired LAN through an ethernet card; the device C is only provided with a wireless card, therefore it can only communicate with the device B, which is provided with a wireless interface as well. Only the device B is able to communicate through the two different interfaces it is equipped with. The JXTA default implementation allows just one interface to be active at the same time. There is no way for the peer B to have multiple communication channels. In the just depicted scenario, a peer residing in the device B can communicate with the devices A and C, but not with both at the same time (i.e., the wired and the wireless interface can not be used by the peer at the same time). Besides, a peer is assigned a network address (namely Endpoint) during the initialization phase of the platform. JXTA does not provide any mean to modify that address at runtime. If the peer leaves for a while for any reason and tries to reconnect to the network through another access point, it will not be able to join the peer community, since there is no way to update its Endpoint and the other peers cannot see it. As a consequence, JXTA (at least, the present implementation) is not recommended in highly dynamic environments where peers frequently log off, move and log on again (typical environments of the ad-hoc networks). To face the exposed limits, a new appropriate mechanism has been designed and implemented. A new functionality has been added to monitor every new peer's network interfaces that turn active. In particular, the new **TCP Transport Service** (see figure 1) is in charge of notifying each new network interface, turning active at any time, to the new **EndPoint Service**, which in turn will make them available to the application as new Endpoints.

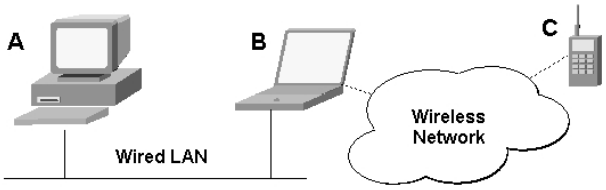

Figure 2. Dealing with multiple interfaces devices

As far as the Resource discovery service is concerned, the one provided by JXTA has been designed bearing in mind that it has to deal with fixed networks and with stable connections, available for most of the task execution time. It is based on the mechanism of publishing/discovery of advertisements containing the information that each peer wants to share (e.g. addresses, services, etc.). The advertisements are saved in a temporary cache and managed by a Cache Manager. Each advertisement will be automatically removed from the cache when its own lifetime has elapsed. Since JXTA relies on fixed and stable connections, in its existing implementation the timeout is set to a high value and there is no way for a user application to configure it. This is not acceptable in ad hoc network environments where frequent connections cause a lot of discovery requests and, consequently, a lot of slow reading operations on disk. Starting from the old resource discovery mechanism, new features have been added in Expeerience, in order to cope with the requirements of mobile ad hoc networks. In particular, the original **JXTA Discovery Service** has been enhanced in order to adapt to the ad hoc environments. The user applications get benefit of faster advertisement storage procedures, relying on the central memory rather than on the disk cache.

3.2 Code Mobility

In Jxta, services can be compared to applications that use the paradigm of the Remote Calls Procedures. When a peer finds a new service offered by a remote peer, it simply uses that service by passing the input parameters and waiting for the results of the processing. In Expeerience, the concept of *Mobile Service* has been introduced. The peer is given the capability of uploading/downloading a service to/from another peer. This way the service code, composed of files and classes, is migrated towards the requesting peer and is locally executed. Furthermore, the service code maintains its execution state during the migration, allowing for more complex and flexible application on top of this architecture. The novelty introduced by the Mobile code Service is that now the user application not only relies on the possibility of acting as a client with respect to a peer implementing the server, but it can also become a server of

itself, thus relieving the neighbor serving peer from consuming power calculation in the cases where it is not available. By using the Mobile Code Service, the peer owning the service ”shares” the application with other peers. The traditional concept of sharing only data, which is a basic concept in peer networks, has been enhanced by the concept of ”code” sharing. From an implementation point of view, the Mobile Service has been implemented as a JXTA service. There are two different ways to exploit the service : 1) a peer requests a service from another peer that will reply by uploading the service code to the requesting peer; 2) a peer autonomously decides to upload the service code to another peer, which in turn will have to choose whether to accept the code or to reject it.

The service for the code mobility is described by a specific interface *MobileCodeService*, which defines the methods for the management of mobility. These are the methods for requesting and/or sending a class and accepting a class that has not been requested.

```
public interface MobileCodeService extends Service {
  public void requestClass(String classname,String destPeerID,
                           MobileCodeListener listener);
  public void sendResponse(PipeAdvertisement adv, String className,
                           String peerID,String type,int queryID);
  public void sendClass(String classname,String peerID, String destID,
                        MobileCodeListener listener);
  public void approveClass(PipeAdvertisement adv, String className,
                           String peerID,int queryID);
}
```

In particular, a request of a class or of an object transfer, and the related replies, have been implemented as XML messages. For instance, the request of the byte code of a class looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
 <ByteCodeRequest>
   <PeerID> . . .</PeerID>
   <ClassName> . . . </ClassName>
 </ByteCodeRequest>
```

where `PeerID` is the tag related to the identifier of the requester, and `ClassName` is the tag related to the searched class.

These methods enable to implement one of the pure concepts of code mobility in the scenario of MANETs with a few code lines: a mobile agent system. In fact, a mobile agent can be created, which keeps its state and resumes the interrupted execution by migrating from a peer to another and using the services provided by the MobileCodeService and the ones of the PipeService (this is another standard service of Jxta). All this can be done with no need to worry about network programming, sockets, protocols, but only with a couple of classes imported from the implemented middleware.

Just to provide an example, let us see what happens when an agent needs to be sent from a peer A to a peer B. The peer A serializes the agent and includes it in a Message in the form of a byte array. Then the peer A asks B for the authorization, and waits for the opening of a pipe: if B replies positively, it opens the input pipe on its side, and is ready to receive a message on it. Then the peer A can enter the message in the open pipe. This message contains the bytes of the mobile agent. Once the agent has reached the peer B, it is regenerated by the state where it was serialized.

4 Event Organizer: an example Application

In this chapter we will see how an application can be written, by using the features made available by Expeerience, for organizing events and personal meetings. The Code Mobility feature will be used both for transferring the application from a node to another and for implementing a mobile agent paradigm widely used by the application. Let us assume we are in a network environment consisting of several users, with their own devices equipped with a wireless interface, working on common projects in a meeting room. Let us suppose that one user only has a specific application that allows to synchronize the meetings for the different users. The user who has the application contacts the peers that should be involved, and tries to send the application code to them. All the accepting peers will receive the application, and will be available for scheduling new meetings. From now on, anyone can start a procedure for a meeting request, by using the application provided. The purpose of the application is to avoid the need for the user to consult his/her organizer, and to find (by attempts, if necessary) a useful time for scheduling a new meeting. Some applications are currently available, which try to solve these issues by synchronizing a palmtop’s content with the desktop’s one, but not with several palmtops belonging to other users. In the best of the cases, a file sharing can be done. However, by using on-the-market applications, we cannot think of communicating while moving from a room to another, by detecting the devices belonging to other colleagues.

The designed application, which will be simply called *Organizer*, manages the user’s meetings day by day, hour by hour. If a user wishes to schedule a meeting with other users for a specific day at a given time, he/she will only need to enter this information and wait for a successful synchronization. The application will:

1. search for the users whom the person wishes to schedule the meeting with;
2. check that all of the contacted users are available for that date and time;
3. schedule this meeting on each user’s organizer;
4. notify each user with the confirmation of this new meeting.

Now let us see how the new services provided by the platform Expeerience are used. The new peer discovery service will be used for searching the peers available in the network. This allows to detect which users (peers) are in the network at any time, and can be involved in the meeting. The synchronization will be performed by a mobile agent that, exploiting the service of Code Mobility, migrates from one peer to another. This agent is called *Resolver Agent*; its task is that of migrating from one peer to another of a to-be-contacted list and checking whether the request for a new meeting is compatible with the schedule of the peers’organizer. If the last peer in the list has

been reached, the agent will migrate back to the just visited peers in order to confirm the new appointment, and finally will migrate to the peer that originated the meeting request. It is worth nothing that the peers's reachabililty is granted by the disconnection management service. Even if a peer moves while the agent is migrating, the latter will always be able to get the new peer's access point and migrate to it, in a totally transparent manner. The scheduling fails if either at least one of the peers' schedule is not compatible with the new meeting time, or if at least one of the peers has left. The application can be configured in such a way that if the next peer in the list is not reachable (since it has left), the agent will keep on migrating to the available peers, so that in the end the new meeting will be scheduled only among the peers available at request time.

Just to give an idea of the (relative) easyness of developing such an application we provide some data about the code dimension: apart from the graphical user interface classes used for user interaction, the logic of the application has been coded using few lines of code for a total amount of about 4 KB of bytecode.

5 Conclusion and Future work

In this paper we have discussed about the similarities and the main differences between two completely decentralized architectures, P2P systems and ad-hoc networks, highlighting the ad-hoc system middleware requirements. The main goal of our work was to define a middleware providing high-level support for MANET application developers exploiting P2P technology over mobile ad hoc networks. The prototype of the runtime environment, named Expeerience, has been developed in Java and is based on JXTA, which significantly simplified the development work and is a well designed P2P system with a rich set of concepts. The middleware designed fully satisfies MANET requirements: it manages the discovery service, multiple interfaces, and intermittent connectivity, and supports the code mobility for writing applications that use the mobile agent programming paradigm. Apart from the implementation of other more complex test applications to prove the benefits and the added features of our system, our future research will focus on the following areas: security, new and optimized resource discovery algorithms, and PDAs support. With regard to security issues, the use of a public key infrastructure (PKI) within a mobile ad-hoc network is more difficult than in a classic P2P system. The creation of an Internet-like PKI in an ad-hoc network is not so straightforward, and poses basic questions to be faced with. A second issue we are working on is related to the optimization of the discovery peer service for our architecture using the Rendezvous mechanism of JXTA. Finally, a parallel line of investigation and development is related to the porting of what has been presented in this paper to heterogeneous devices including PDAs, handheld devices supporting light version of the Java environment (J2ME) and a lighter version of the JXTA middleware (JXME).

References

[1] D. Barkai. *Peer-to-Peer Computing: Technologies for Sharing and Collaborating on the Net*. Intel Press, 2002.

[2] P. Bellavista, A. Corradi, and C. Stefanelli. Mobile Agent Middleware for Mobile Computing. *IEEE Computer*, 34(3):73–81, March 2001.

[3] L. Capra, W. Emmerich, and C. Mascolo. Middleware for mobile computing. In *Tutorial Proc. of the International Conf. on Networking 2002*, Pisa, Italy, May 2001. LNCS 2497, Springer Verlag.

[4] I. Chlamtac, M. Conti, and J.J-N. Liu. Mobile ad-hoc networking: imperatives and challenges. *Ad Hoc Networks*, 1(1):13–64, Jan 2003.

[5] A. Dutta-Roy. Networks for homes. *IEEE Spectrum*, 36(12):26–33, 1999.

[6] FREENET. http://freenetproject.org.

[7] Gnutella. http://www.gnutella.com/.

[8] M. Hannicainen, T.D. Hamalainen, M. Niemi, and J. Saarinen. Trends in personal wireless communications. *Computer Communication*, 25(1):84–99, January 2002.

[9] IETF Mobile Ad hoc Networks (manet) charter. http://www.ietf.org/html.charters/manet-charter.html.

[10] L. Kleinrock. Nomadic Computing and Smart Spaces. *IEEE Internet Computing*, 4(1):52–53, Jan-Feb 2000.

[11] G. Kortuem, J. Schneider, D. Preuitt, T.G. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proc. of the International Conference on Peer-to-Peer Computing (P2P2001)*, Linkopings, Sweden, August 2001.

[12] E. Kovacs, K. Rohrle, and M. Reich. Integrating mobile agents into the mobile middleware. In *Proc. of the 2nd Int. Workshop on Mobile Agents (MA'98)*, Stuttgart (Germany), September 1998. Springer Verlag.

[13] ACM International Symposium on Mobile Ad Hoc Networking and Computing. http://www.sigmobile.org/mobihoc/.

[14] A. Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[15] C. E. Perkins. *Ad Hoc Networking*. Addison-Wesley, Boston, MA, USA, 2001.

[16] The JXTA Project. http://www.jxta.org.

[17] R. Schollmeier. A definition of peer-to-peer networking towards a delimitation against classical client-server concepts. In *Proc. of the 7th EUNICE Open European Summer School (EUNICE'01) and the IFIP Workshop on IP and ATM traffic management*, Paris (France), September 2001.

[18] R. Schollmeier, I. Gruber, and M. Finkenzeller. Routing in mobile ad hoc and peer-to-peer networks: a comparison. In *Int.Workshop on Peer-to-Peer Computing. In Networking 2002*, Pisa (Italy), May 2002.

[19] O. Tomarchio, M. Bisignano, A. Calvagna, and G. Di Modica. ExPeerience: A JXTA Middleware for Mobile Ad-Hoc Networks. In *Third Int. Conference on Peer-to-Peer Computing (P2P2003)*, Linkoping (Sweden), September 2003.