IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 2, pp. 144-156
ISSN: 1646-3692

MIDDLEWARE FOR DISTRIBUTED APPLICATIONS IN DECENTRALIZED WIRELESS NETWORKS

Wolfgang Golubski *Zwickau University of Applied Sciences, FB Physikalische Technik / Informatik, Dr.-Friedrichs-Ring 2a, 08056 Zwickau, Germany*

ABSTRACT

Mobile computing is one of the fast-growing fields of the current computer era. In this paper we present a middleware appropriated to the development of applications for mobile ad-hoc peer-to-peer networks. The proposed middleware implements a communication service and an host respectively service discovery in fully decentralized networks. The design and the function of the middleware will be discussed. To argue the quality of the middleware we sketch the development of two services, a distributed card game and a chat application.

KEYWORDS

Middleware, Mobile Application, Ad-Hoc Networks, Software Development, Host Discovery, Service Discovery, Collaboration.

1. INTRODUCTION

Mobile computing is omnipresent. The number of devices (like handhelds, personal digital assistants, mobile phones, laptops) equipped with wireless network interfaces and their facilities to interconnect is continuously growing. The systems are characterized by (1) wireless communication, (2) self-organization, (3) spontaneous collaboration and (4) limited hardware resources (memory, cpu, communication bandwidth). Due to these characteristics host and service discovery is one of the most important challenges in developing mobile applications.

In this paper we present a middleware for mobile wireless networks being suitable for application development in the context of collaborations (like information and data sharing services, entertainment and game services, chat and instant messaging, profiling). Some interesting fields of application are:

- Tourist information - Tourists in a city can collect a lot of interesting data about the city like sights or catering. When a tourist comes new to the city, he can participate on the experiences of other tourists by exchanging respectively downloading their data collection (i.e. experiences).
- Conference situation - At a large conference, with participants from all over the world, one would like to meet persons with similar interests.
- Entertainment - Network gaming is a promising increasing business industry.

The middleware called MaJo acts as an intermediary between the hardware network interface and the application layer. MaJo controls the complete communication area including host and service discovery and management. That means that the software developer is unburdened from recurrent tasks and can focus on the real important things (like business logic, data modeling and user interfaces). We investigate and discuss the facilities, advantages and drawbacks of using our middleware approach.

As case studies we realized two applications: a game service and a chat service. Each application is a service of MaJo. The game we have implemented is a German card-game called *Schwimmen* (swimming).

The paper is structured as follows. In the next section we give a short introduction to middleware for mobile computing. The developed middleware MaJo will be described in Section 3. The following two sections present the developed applications based on MaJo. Then we describe related works. In Section 7 some experiences and future works are reported. Finally a conclusion will finish the paper.

2. MIDDLEWARE AND AD-HOC PEER-TO-PEER NETWORKS

In a distributed computing system, middleware can be defined as the software layer that lies between the operating system and the applications. Today, middleware is successfully used in database management systems, web servers, application servers, content management systems, and similar tools that support the application development and delivery process. The reason why using middleware is to provide high-level abstractions and services to applications, to ease application programming, application integration, and system management tasks. Technologies like Corba, J2EE, DOTNET, enterprise service bus are the base of many frameworks and applications. All are working on fixed network (communication) topologies.

A peer-to-peer (or P2P) network is a model where each host acts as client as well as server. They are typically used for connecting hosts via largely ad-hoc connections. Such networks are useful for many purposes like file and data sharing. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. A typical example for a non peer-to-peer file transfer is an FTP server. One user uploads a file to the FTP server, then many others download it, with no need for the uploader and downloader to be connected at the same time.

In networking, ad-hoc describes a connection model for wireless LANs that require no base station. Devices discover others within their environment to form a network for those hosts. Devices may search for other hosts in the whole environment.

Mobility means that network connections are possible but very dynamic, since hosts can moving around and can leave anytime the common environment.

All these aspects should be carefully attended in the middleware MaJo.

3. THE MIDDLEWARE MAJO

The MaJo middleware was implemented in J2ME (Java 2 Micro Edition) and was tested on iPAQ H3630 and H3850 handhelds equipped with WLAN. As Java virtual machine the J9 from IBM [IBM] was used.

The middleware MaJo offers an interface to the developers which they can use to develop mobile distributed applications for networks in a comfortable way.

Applications developed by using MaJo are regarded as services. The concepts service and application are therefore used synonymously. The developed middleware is put on the open source JRRA system for peer-to-peer networks and provides an application interface to the outside. Unfortunately, the JRRA resources and WWW pages are no longer available.

3.1 JRRA

Java RockyRoad API (JRRA) [JRRA] is a scalable API developed for peer-to-peer networks. JRRA is implemented in Java and in a version for J2SE as well as J2ME platform. Hence, the system can be used for standard desktops as well as for mobile devices. Both can therefore exchange data with each other. JRRA works protocol independently. Several transportation mechanisms like TCP and UDP can be used simultaneously. To be able to send data between peers, types of data packets must be defined by the developer. Furthermore a corresponding packet listener which has the task of informing about the arrival of data packets must be implemented for every type of data packets.

The architecture of the JRRA system will be explained now briefly with the help of the schematic representation in Figure 1.

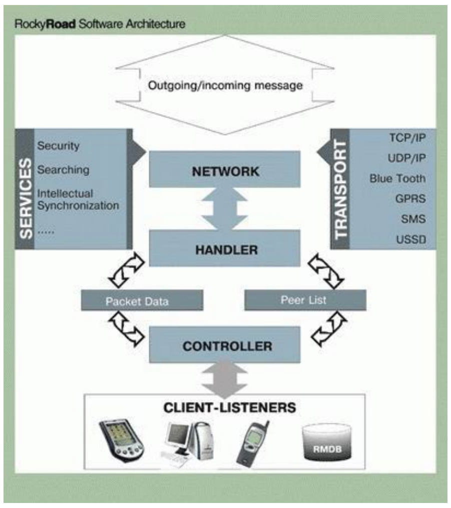


Figure 1. Java Rocky Road API

To send out a data packet, a sender method of the controller is called. The data packet and the list of the receivers will be combined in a message. The controller passes the message to the handler. The handler transforms the message into a byte stream and fragment it. The byte stream now is passed to the network component and sent by the corresponding network protocol.

Also, incoming packets are received by the network component and converted to a byte stream. The byte stream then will be submitted to the handler which sends and defragments the packet. The packets now are passed to the controller. The controller informs the listener objects which are assigned to the corresponding data packets about the arrival of the data packets.

3.2 MaJo

The developed middleware MaJo is put on the JRRA system. The architecture of the complete system has three layers:

Table 1.

Application Layer	Applications
Service Layer	MaJo
Network Layer	JRRA

The interface between the service layer and the network layer is the controller component of JRRA.

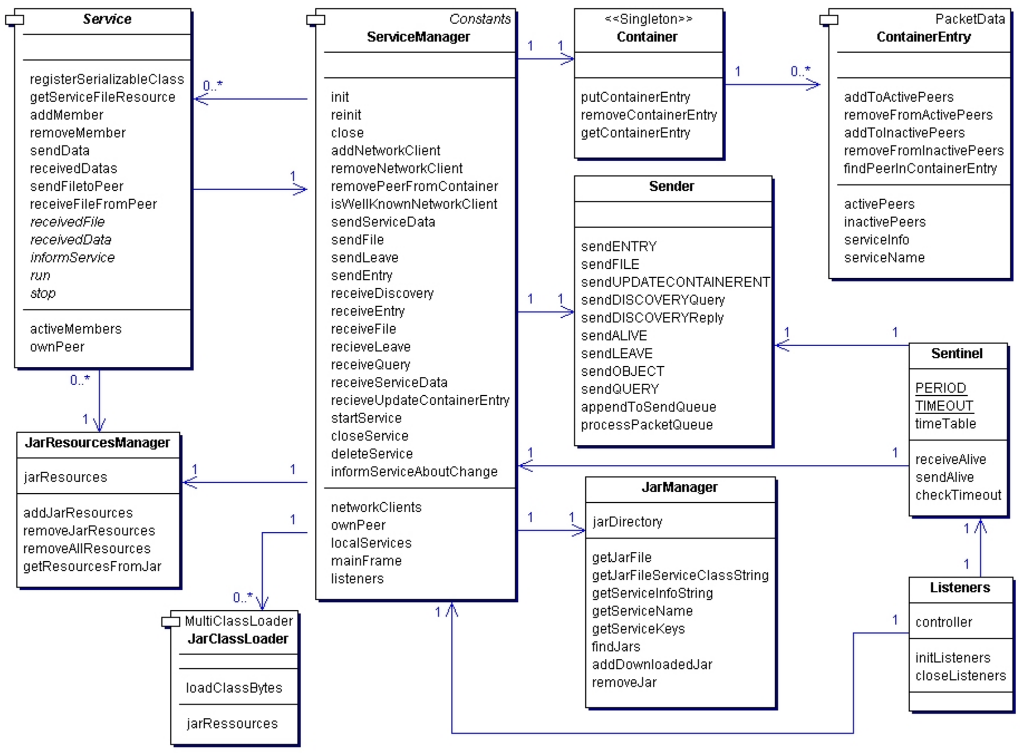


Figure 2. MaJo Model

The interface between the application layer and the service layer is provided by the MaJo service class being the entry point of MaJo. In Figure 2 an overview about the MaJo API is presented.

The individual functional units are specified in the following.

Host and Service Discovery

The middleware is designed for use in mobile ad-hoc networks. Essentially, the middleware must locally organize the complete network infrastructure. The middleware must have self-organized features.

Due to lack of administrative components in a mobile ad-hoc network no information about participants in a central place can be questioned. Therefore the information about the complete network topology must be locally available for each participant.

At the initialization of the middleware it is necessary to find out which mobile devices are active in the environment. Active means here that the middleware is running on the mobile device. Host discovery can exclusively be realized with the help of the UDP broadcast mechanism. The corresponding component of MaJo sends a broadcast packet at the start and informs the other active devices about its own network address. At the receipt of such a packet the already active participants return their own addresses to the sender. If all active devices have received and answered the broadcast packet, the view of the network topology is identical for all participants.

The topology of the network can change fast by the fact that new participants join or leave the network. The ability to take new participants is provided by the broadcast packet mentioned above. If active participants want to leave the ad-hoc network, they can notify the other participants. If an active participant loses, however, the contact to the other participants due to movement from the environment, it cannot cancel. However, leaving must be noticed by all participants. For this a periodical mechanism which can be realized by a proactive or reactive approach is necessary. Proactive means that every active participant checks continuously if all well-known members of the network are accessible via TCP and can therefore notice the leaving. This approach doesn't make sense since it uses a lot of bandwidth. Among ten participants each participant must send a so-called alive packet to all other participants. Altogether, 90 packets have to be transferred. The reactive approach doesn't need less bandwidth under the use of TCP. As alternative we use the broadcast mechanism of UDP which reaches all participants by sending one multicast packet in the case of the reactive model. Now 90 packets can be reduced to 10 packets per period. The disadvantage of using UDP is the fault vulnerability. If no alive packet of a participant was received in a period, this participant hasn't left the network inevitably. The package cannot have arrived because of a UDP fault. The probability that the participant has definitively left the network increases the longer no alive packet arrives. The host discovery is realized so that every participant sends periodically a alive packet whereby the network topology is permanently up-to-date with an acceptable delay.

Service Discovery and Current Service Topology

The middleware offers a permanently current view of the available services in the network. The actual service list is immediately accessible and therefore must be stored locally.

At the start of the middleware the new user must inform the other members of the network about his own services. Furthermore, information about the services which are offered by other participants must be collected. Information about a service of a user consists of the unique service name and the information whether the user just takes part of the service or not. The service discovery concept is similar to the host discovery model stated above and can be combined with it. Each user can start, finish and delete a service. In turn these three activation must be transmitted to the other users of the middleware. Hence, it suffices to send the information consisting of the service name and the way of activation. The alive packets can be extended by some service information.

The Service API

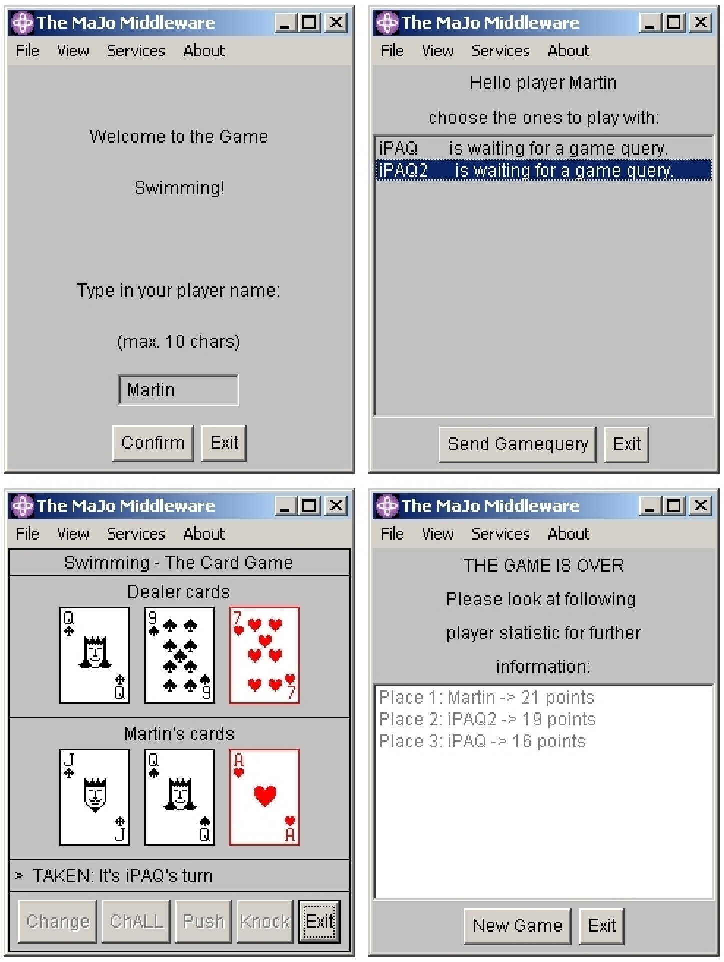
The MaJo middleware provides an abstract programming interface, see Figure 2, and predefines several classes and method implementations suitable for mobile ad-hoc peer-to-peer communication. Service-specific data channels can be implemented in a simple way. Besides this, file transfer will be supported. Services residing on a peer can easily transferred to another peer by means of jar-archives. The only restriction is that a service must be conform to the MaJo API.

4. SWIMMING GAME SERVICE - A CASE STUDY

To represent the strength of the middleware MaJo introduced above exemplarily, a mobile distributed game application was developed based on MaJo. The application is a service of MaJo, the terms *application* and *service* are therefore used synonymously in the following.

4.1 The Game

The game we have implemented [MaJ2] is a German card-game called *Schwimmen* (*swimming*), see Figure 3. Swimming is a card-game, where the goal is to receive the highest score as possible. It is played with usual 32 cards. At least 2 players must play. Each player receives three cards. The dealer receives two packs of cards each consist of three cards. He can regard one of the two packs of cards and decide whether he would like to keep the considered pack of cards for himself and opens the other one. In the negative case the dealer takes the other pack of cards for himself and opens the first one. The actual player can exchange either 1 or all 3 cards with the cards in the center. If he does not like it, then he can push, i.e. make nothing or in addition, close the play by passing. That is however only possible if a complete round is played. If one player closes the game, all other players may still try to exchange their cards. Now the card points will be counted and the player with the highest score is the winner.



Welcome to the Game

Swimming!

Type in your player name:

(max. 10 chars)

Martin

Hello player Martin

choose the ones to play with:

```
iPAQ     is waiting for a game query.
iPAQ2    is waiting for a game query.
```

Swimming - The Card Game

Dealer cards

Martin's cards

> TAKEN: It's iPAQ's turn

THE GAME IS OVER

Please look at following

player statistic for further

information:

```
Place 1: Martin -> 21 points
Place 2: iPAQ2 -> 19 points
Place 3: iPAQ -> 16 points
```

Figure 3. The Swimming GUI

4.2 The Implementation

The swimming service shall be capable in a mobile ad-hoc network on PDAs and make use of MaJo. Since it is a network game, the service requires a lot of communication between the players caused by game requests and information exchanges. The service needs an own data channel for the communication provided by MaJo. At every time, the service must be informed about the state of all participants. Several games are possible in the network at a time. By means of MaJo these requirements can be fulfilled and the game can be implemented in a comfortable way. At least two and at most nine players can be involved in one game. The player names must be unique. Players involved in a game are able to finish the game at every time.

In the following the needed functional units, structures and important processes are described regarding the service swimming.

The functionality provided by MaJo enables swimming receiving and sending data over a service specific communication channel. Swimming will be informed by MaJo about the hosts which activate or deactivate the swimming service.

In Figure 4 the class diagram of the core units are specified.

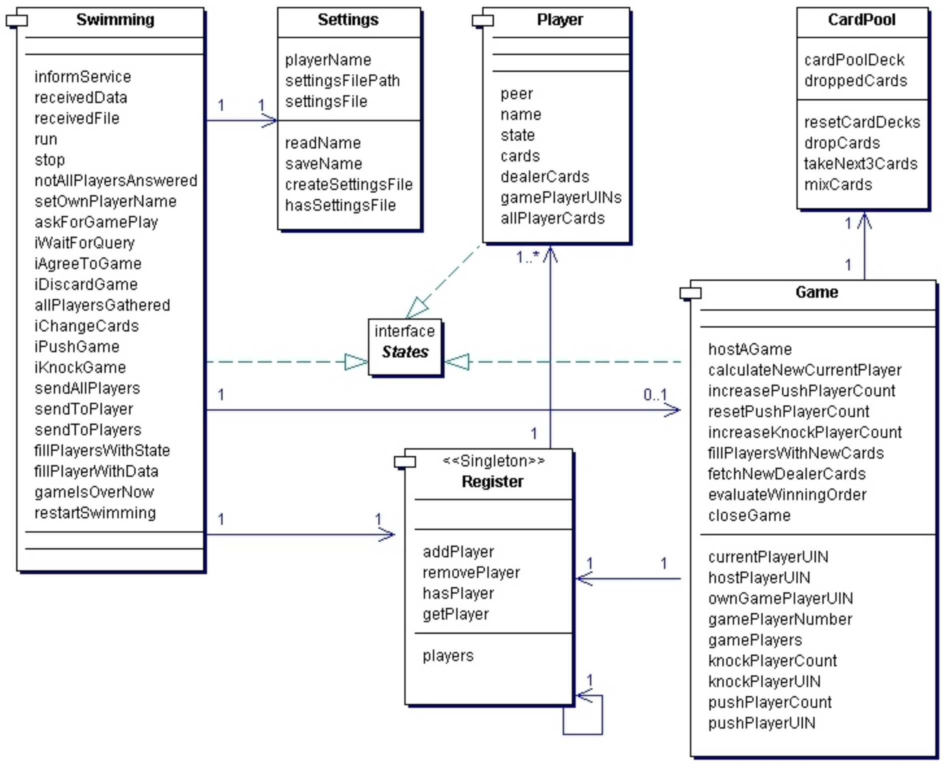


Figure 4. The Swimming Service

- Swimming: The central processing component of the service represents the class Swimming which is inherited from the service class of MaJo. A swimming object has the task of coordinating the application and game logic. It has to manage the incoming and outcoming events (like agreeing game request, stopping a game, exchanging cards). If a game request has been answered by all players in demand positively, it is the task of the component to activate (to start) the game. If a player answers a game request neither positively nor negatively within 60 seconds, the game is rejected. Furthermore, the avoidance of double player names guarantees that a player can play only one game at the same time. A swimming object collects all information about the players from the network so that this information is available on every service host. The information consists of the player name and player state. No central network component are used.

- Game: A game object contains and updates the information important for a given game. There exist two types of game objects. A *hosted-game* object is created on the host of the player who starts a game request. A *not-hosted-game* object is created on the host of each player who receives a game request and doesn't belong to any game yet. If a game takes place, the hosted-game object has the task of servicing the dealer pack of cards and providing the needed cards to all players. The not-hosted-game object does not service the dealer pack of cards. Hence the host-game object adopts temporarily a server functionality. The function of both game types is saving the player of a game and determining the player which will conclude the next move. Furthermore, the game object calculates the current score of each player and therefore can finish the game if necessary.
- Cardpool: There is a class CardPool besides the game class which is used by the hosted-game object. The card pool object represents the dealer pack of cards and the rest pack of cards and realizes the accesses to the cards. The rest pack of cards consists of the cards which were discarded in run of a game. Card triple can respectively be taken by the dealer pack of cards. At the beginning there is an empty rest pack of cards in the card pool and the pack of cards is shuffled.
- Player: A player object represents a player in the swimming game. Every host becomes a player if it activates the swimming service. A player object is comprised of the data which are transferred over the service channel. A player contains the peer information it is assigned to, the player name and the state (*player waits for game request*, *player plays*,...). It also includes the cards of the player and the dealer cards. Every instance of the swimming service saves all players known in the network. This information are steadily updated.

The intention to implement a game service was that such application form seems to be an ideal test candidate for the quality of MaJo. A game service must contain (1) player management, (2) data channels, and (3) high demand on dynamic communication.

5. CHAT SERVICE – A CASE STUDY

A chat service [MaJ1] was also a good candidate to prove the quality of the MaJo middleware. A chat service must contain (1) group administration and management including decentralized subgroups, (2) data channels, (3) file transfer, (4) private chat rooms, and (5) high demand on dynamic workflows. In Figure 5 one can regard the user interfaces of three participants of the chat service.

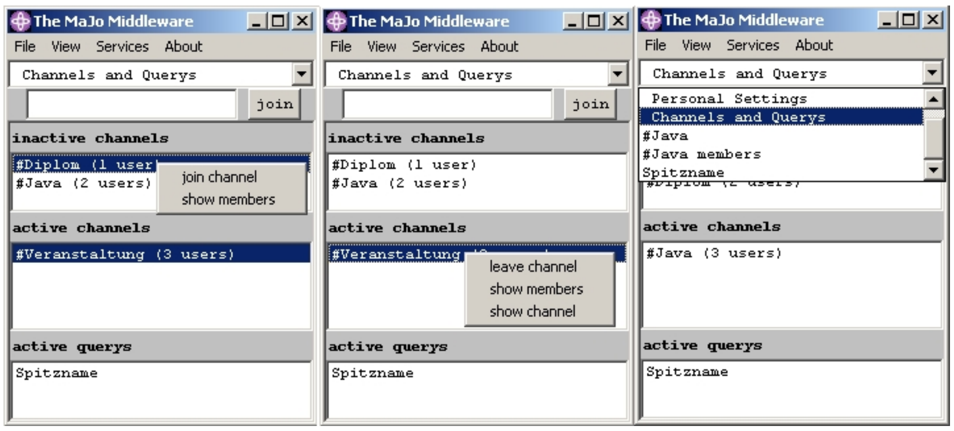


Figure 5. The Chat GUI

The developed chat is equipped with basic functionalities like Internet Relay Chat (IRC) [IRC]. Two short remarks to the realization characterize the challenges. The chat service is conceived so that a complete overview of the channels being available. A decentralized redundant storage of the members of a channel is advisable. The data transport uses the already implemented MaJo methods. This functionality can serve to exchange e-cards or smaller documents between users, e.g. at a conference. The file transfer of the chat services is implemented so that only files of size smaller than 500 KB are transferred. Since the memory equipment of mobile devices are limited we have decided to implement this restriction. But the value of 500KB can be configured by the user. In Figure 6 an overview of the implementation is given.

- ChatService: the class extends the MaJo service class and is the core of the chat implementation. All information are dynamically managed, e.g. the unique nick name, the channel and the query objects. If the chat service object receives any data, it delegates them to the corresponding functional unit. On the other side this object is also responsible for the data transport to one chat member or to all group members of the chat. Hence the chat service implements the complete application logic.
- Channel: an object of this class is mainly designed for group (many-to-many) communication in discussion forums. A channel has an unique name (identifier).
- Query: a query object allows a one-to-one communication via private message.
- PersonalSettings: the personal configuration data are stored.
- InputHandler, InputHistory: all input events and input text are handled by this class.

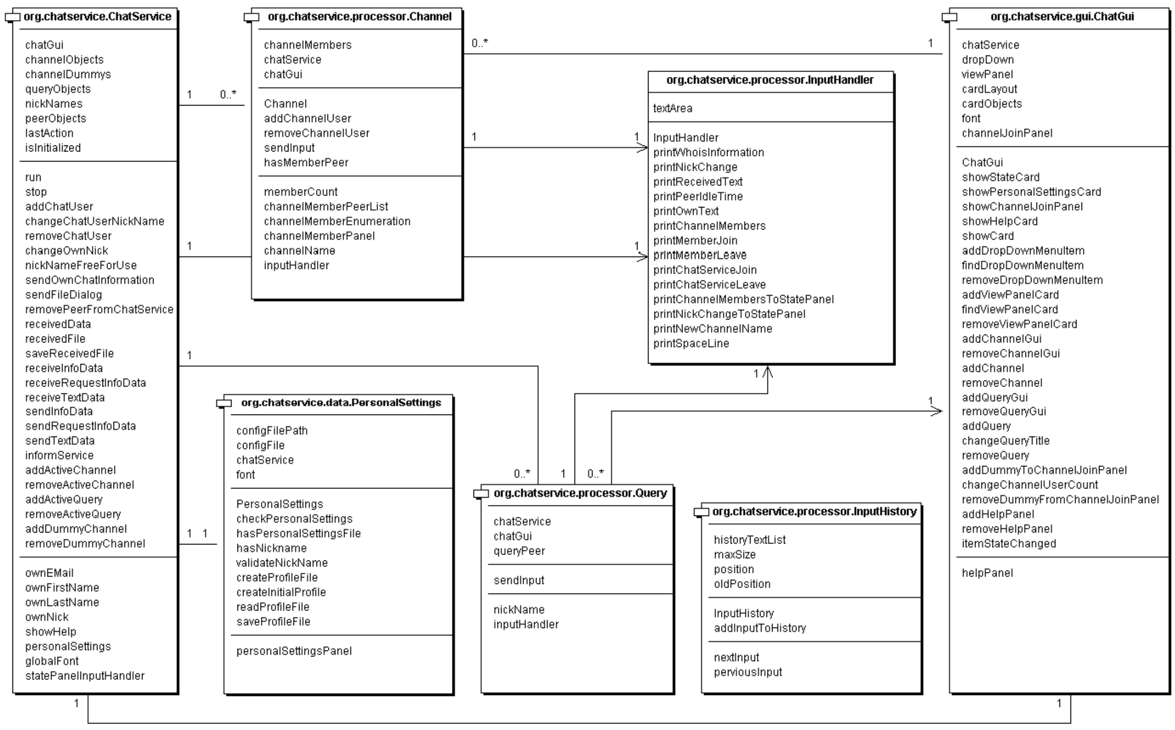


Figure 6. The Chat Model

6. RELATED WORK

JXTA [JXTA] and JXME [JXME] are open source projects initiated by SUN Microsystems. JXTA provides a fully decentralized environment that enables services to work in an ad-hoc fashion. But JXTA cannot be used for mobile devices because of their limited hardware resources. JXME is a project especially focussed to transfer JXTA to mobile devices. But JXME is not fully decentralized so that mobile hosts cannot directly communicate to other hosts. A relay host takes over the role of a dispatcher or a communication manager. A proxiless version exists but the implementation is going on.

Proem [PROEM] is a framework for rapid development of applications for ad-hoc environments. Proem is implemented in Java (J2SE). A mobile version is realized on PersonalJava [JPJ] or JDK1.1.8. The base components are peerlets. Applications are built on top of peerlets. The host discovery and the management of peerlets are executed by the peerlet engine. In some points the Proem approach and the MaJo model are similar. But the main differences are that (1) MaJo provides a sophisticated service discovery and management and (2) Proem needs PersonalJava and J2ME [J2ME] is not supported.

XMiddle [XMID] is a peer-to-peer middleware enabling transparent sharing of XML documents across heterogeneous platforms. The focus is sharing of data in a mobile environment. The support for host and service discovery in the sense of MaJo does not exist.

For more general overviews of mobile middleware and related work we refer to [MID] and [P2P].

7. EXPERIENCES AND FUTURE WORK

The concept of JRRA is deliberate. But, essential parts of the JRRA API must still be implemented new, though, e.g. the protocol implementations. Furthermore it was necessary to remove a number of implementation faults.

Currently we have developed two applications based on MaJo. The game service is described above. The other service is a chat service. In both cases the software development time was relatively short-time. Before the middleware could be used a lot of work has to be done. The realization of the host and service discovery and management was a time-consuming process.

Since the mobile devices work in an ad-hoc manner without any wireless access points, the initial IP-address must be assigned by hand at the start of the middleware. All peers must be a member of the same subnet.

The network performance is moderate for up to 7 simultaneous users. Similar results are reported in [P2P].

All in all, we believe that MaJo is a good choice to significantly reduce the development time for mobile ad-hoc applications.

The next research steps are (1) integration of an intelligent profile model based on user preferences, (2) involving more security features and trust management and (3) using Bluetooth communication. In a similar project we have made the first experiences with Bluetooth and J2ME.

REFERENCES

[IBM] IBM Corporation:. Websphere Studio Device Developer. *http://www-3.ibm.com/software/wireless/wsdd*.

[IRC] IBM Corporation: IRC - Internet Relay Chat. *http://www.irc.org*.

[J2ME] Sun Microsystems. Java 2 Micro Editon. *http://java.sun.com/javame*.

[JPJ] Sun Microsystems. Personal Java. *http://java.sun.com/products/personaljava*.

[JRRA] Offshore Algorithms, 2004. Java Rockyroad API (JRRA). *http://www.jrra.org*.

[JXTA] Sun Microsystems. JXTA Initiative. *http://www.jxta.org*.

[MaJ1] Bredner, J., 2004.: Development of a middleware for mobile ad-hoc networks based on Java 2 Micro Editon – Realization of a chat application, diploma thesis.

[MaJ2] Schmidt, M, 2004.: Development of a middleware for mobile ad-hoc networks based on Java 2 Micro Editon – Realization of a game application, diploma thesis.

[MID] Mascolo, C., Capra, L., and Emmerich, W., 2002. XMIDDLE: Middleware for Mobile Computing (A Survey). *Advanced Lectures on Networking - Networking 2002 Tutorials,* Springer Verlag, Pisa, Italy. volume 2497 of LNCS, pp 20-58.

[P2P] Wang, A.-I., Sørensen, C.-F., and Fossum, T., 2005. Mobile Peer-to-Peer Technology used to Promote Spontaneous Collaboration. *International Symposium on Collaborative Technologies and Systems (CTS 2005),* Saint Louis, Missouri, USA, May 15-19.

[PROEM] Kortuem, G, 2002. A methodology and software platform for building wearable communities. *PhD thesis, University of Oregon*.

[XMID] Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W., 2002. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *International Journal on Wireless Personal Communications,* Kluwer Academic Publisher. Vol. 21, No. 1,pp 77-103.