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
golubski@fh-zwickau.de

Mobile Computing

- Devices equipped with wireless network interfaces
  - handhelds, personal digital assistants, mobile phones, laptops
- Main features
  - limited hardware resources
  - wireless communication
  - self-organization
  - spontaneous collaboration

Application fields

- 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 use the experiences of other tourists by exchanging respectively downloading their information (i.e. experiences)
- Conference situation
  - participants would like to meet persons with similar interests
- Entertainment
  - Network gaming is a promising increasing business industry

The Goal

- Communication infrastructure
  - To find the other hosts
  - To find services in the network
  - To manage the connections
  - To manage the services
- As a case study: a game service
  - German card-game called *Schwimmen* (swimming)

The Middleware MaJo

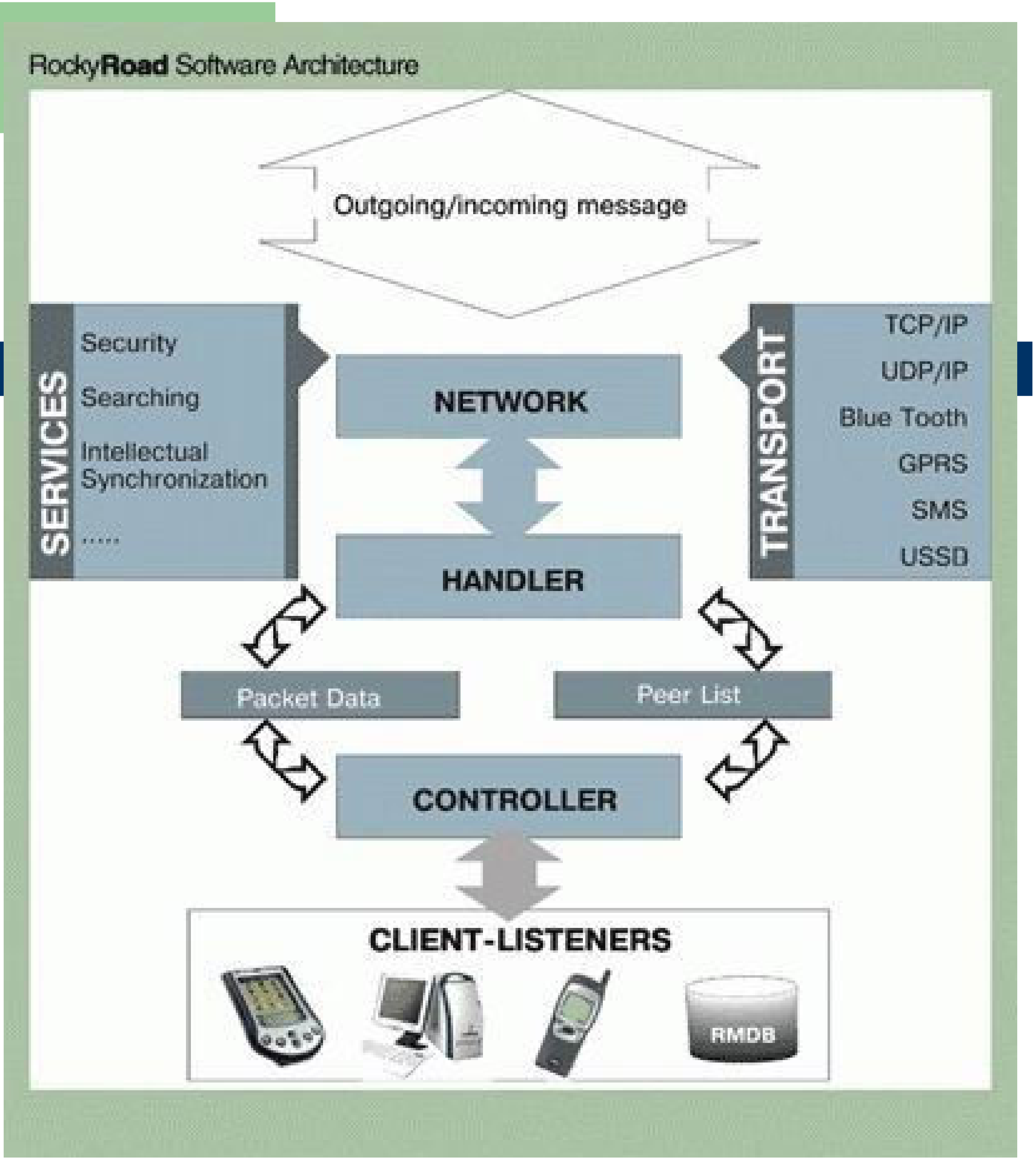
- J2ME (Java 2 Micro Edition)
- iPAQ H3630 and H3850 handhelds equipped with WLAN
- Java virtual machine: J9 from IBM
- Java RockyRoad API (JRRA)

- Support the application development

The Middleware MaJo

- Application Layer
- Host and Service Layer (MaJo)
- Network Layer (JRRA)

- must locally organize the complete network infrastructure
- must have self-organized features


RockyRoad Software Architecture
Outgoing/incoming message
SERVICES
Security
Searching
Intellectual Synchronization
.....
NETWORK
HANDLER
TRANSPORT
TCP/IP
UDP/IP
Blue Tooth
GPRS
SMS
USSD
Packet Data
Peer List
CONTROLLER
CLIENT-LISTENERS
RMDB

Host Discovery

- Problem:
  - no administrative components
  - no information about participants in a central place
- Thus, the information about the complete network topology must be locally available for each participant

Host Discovery

- Initialization: find out which mobile devices are active in the environment
- The new mobile host (MH) sends a broadcast packet and informs the other hosts about its own network address
- The other hosts return their own addresses to the sender (MH)
- If all active hosts have received and answered the broadcast packet, the network topology is updated

Host Management

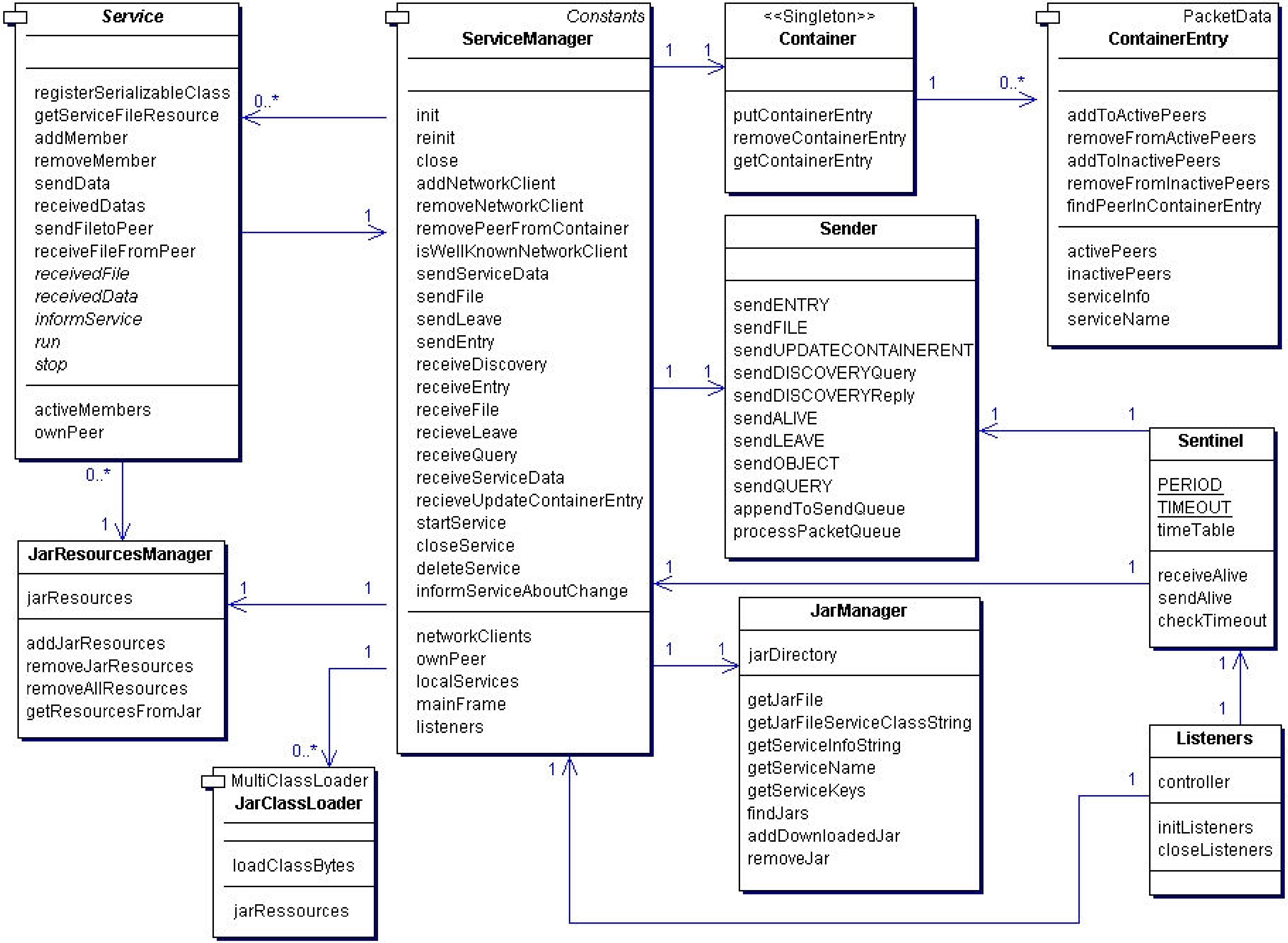
- New mobile host (MH) can leave
- Terminated connection
  - notify the other hosts
- Unexpected disruption
  - leaving must be noticed by hosts
  - sending a multicast packet (alive packet) in periodical intervals

Host Management

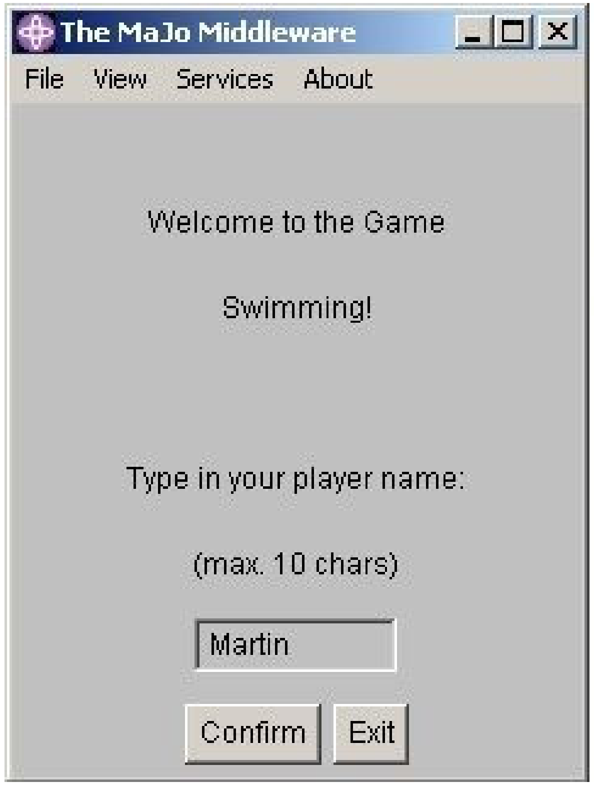
- implemented by UDP
- but UDP is very fragile
- if no alive packet of a MH was received in a specified time => MH has left the network or a UDP error has appeared
- the probability that the MH has definitively left the network increases the longer no alive packet arrives
- hence: alive packets are sent periodically
- after a predefined delay the network topology is up-to-date

Service Management

- permanently current view of all available services
- actual service list is stored locally
- a new user informs the other users about his services
- services of the other users must be collected
- service discovery is similar to the host discovery model
- each user can start, finish and delete a service
- the alive packets are extended by service information


Service
registerSerializableClass
getServiceFileResource
addMember
removeMember
sendData
receivedDatas
sendFiletoPeer
receiveFileFromPeer
receivedFile
receivedData
informService
run
stop
activeMembers
ownPeer
0..*
1
Constants
ServiceManager
init
reinit
close
addNetworkClient
removeNetworkClient
removePeerFromContainer
isWellKnownNetworkClient
sendServiceData
sendFile
sendLeave
sendEntry
receiveDiscovery
receiveEntry
receiveFile
recieveLeave
receiveQuery
receiveServiceData
recieveUpdateContainerEntry
startService
closeService
deleteService
informServiceAboutChange
networkClients
ownPeer
localServices
mainFrame
listeners
1
1
<<Singleton>>
Container
putContainerEntry
removeContainerEntry
getContainerEntry
1
0..*
PacketData
ContainerEntry
addToActivePeers
removeFromActivePeers
addToInactivePeers
removeFromInactivePeers
findPeerInContainerEntry
activePeers
inactivePeers
serviceInfo
serviceName
1
1
Sender
sendENTRY
sendFILE
sendUPDATECONTAINERENT
sendDISCOVERYQuery
sendDISCOVERYReply
sendALIVE
sendLEAVE
sendOBJECT
sendQUERY
appendToSendQueue
processPacketQueue
1
1
Sentinel
PERIOD
TIMEOUT
timeTable
receiveAlive
sendAlive
checkTimeout
1
1
0..*
1
JarResourcesManager
jarResources
addJarResources
removeJarResources
removeAllResources
getResourcesFromJar
1
1
1
0..*
MultiClassLoader
JarClassLoader
loadClassBytes
jarRessources
1
1
JarManager
jarDirectory
getJarFile
getJarFileServiceClassString
getServiceInfoString
getServiceName
getServiceKeys
findJars
addDownloadedJar
removeJar
1
1
Listeners
controller
initListeners
closeListeners
1
1

The Swimming Service



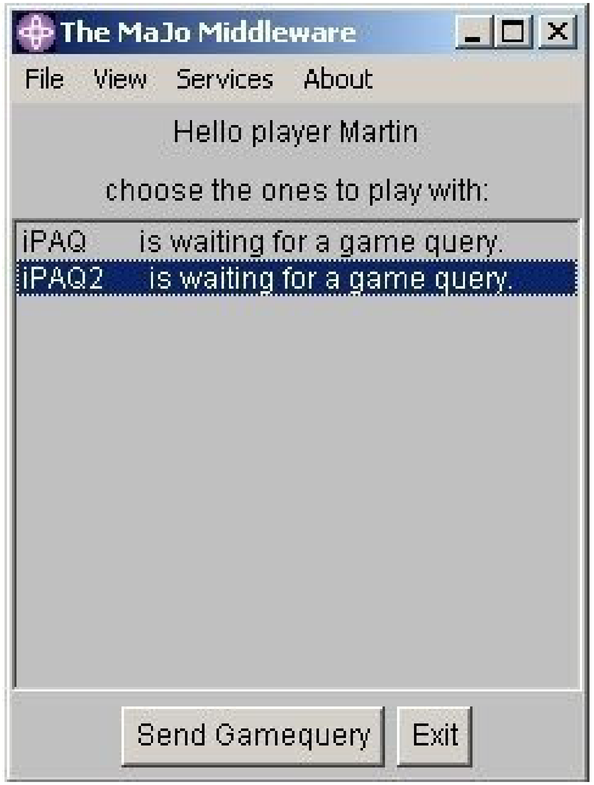
Welcome to the Game

Swimming!

Type in your player name:

(max. 10 chars)

Martin



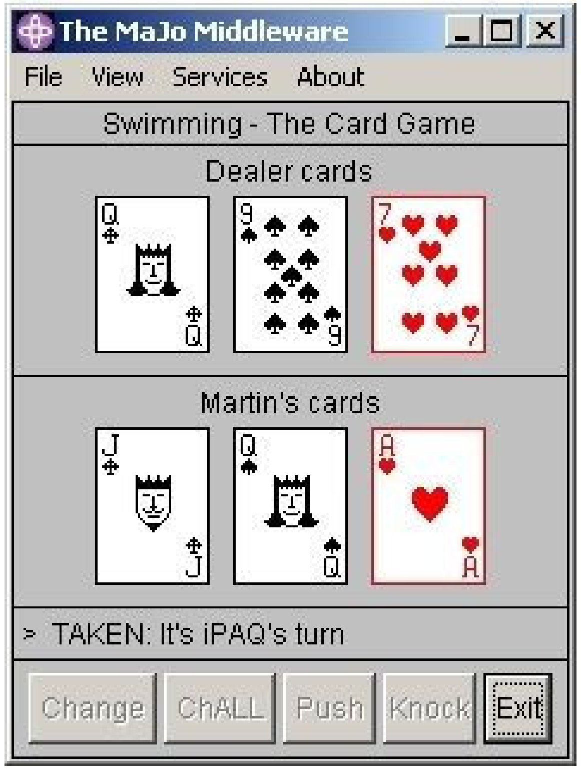
Hello player Martin

choose the ones to play with:

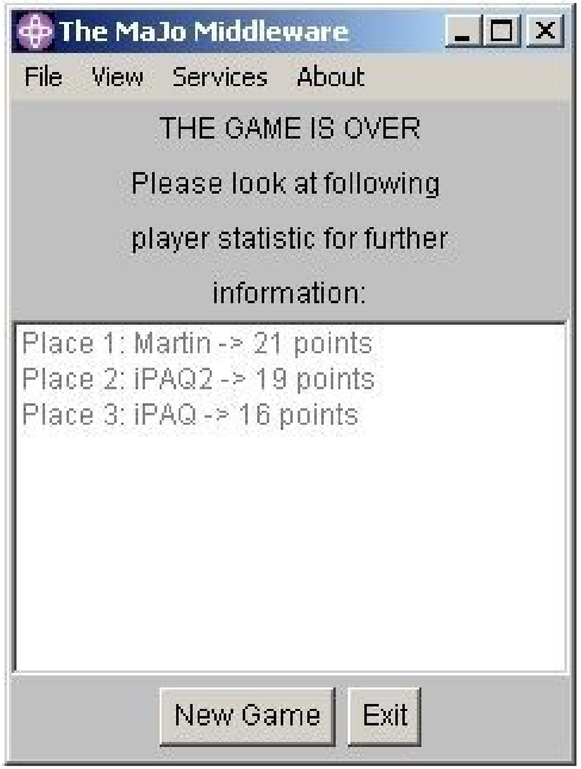
iPAQ is waiting for a game query.

iPAQ2 is waiting for a game query.

The Swimming Service



> TAKEN: It's iPAQ's turn



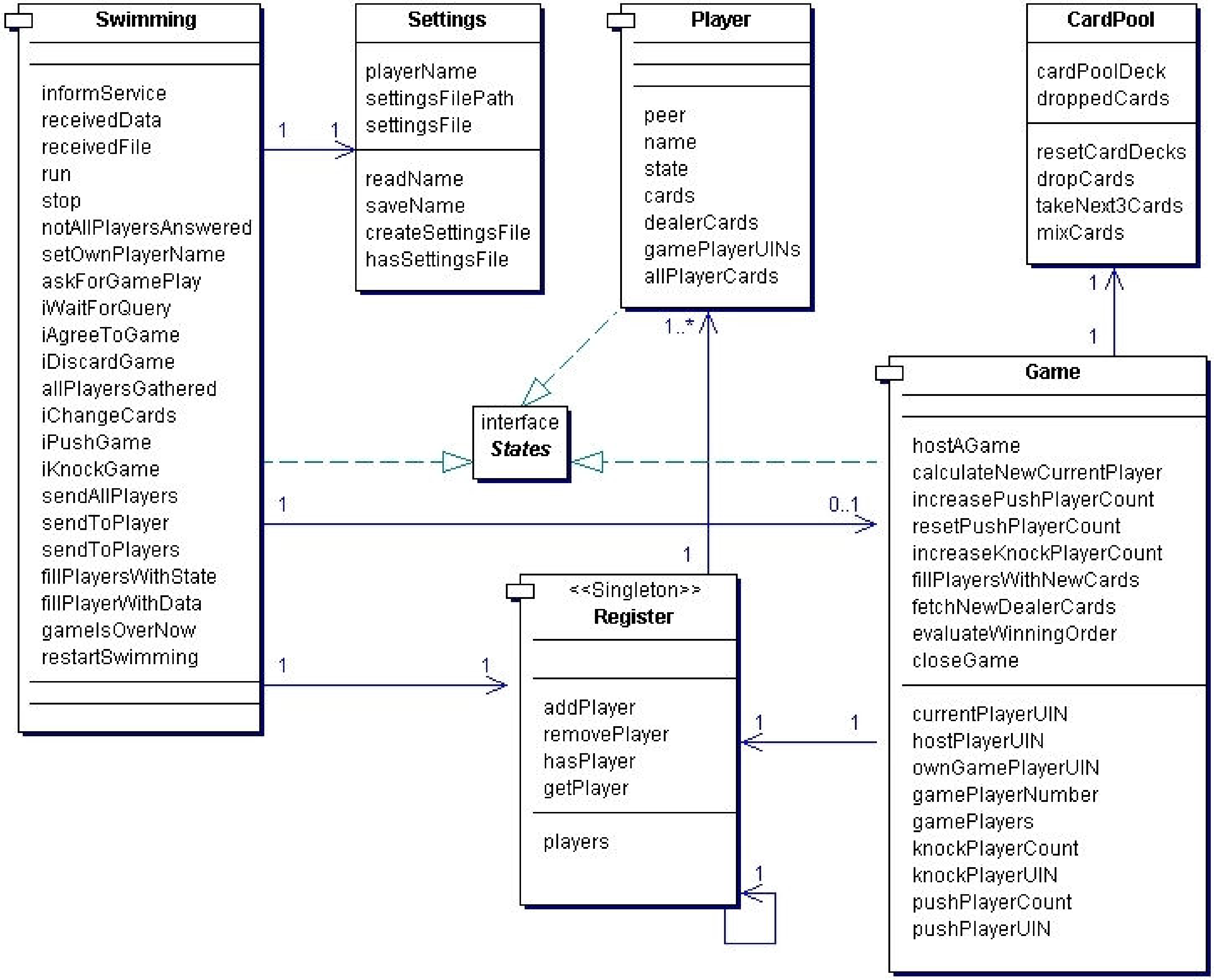
THE GAME IS OVER

Please look at following

player statistic for further

information:

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



Swimming

informService
receivedData
receivedFile
run
stop
notAllPlayersAnswered
setOwnPlayerName
askForGamePlay
iWaitForQuery
iAgreeToGame
iDiscardGame
allPlayersGathered
iChangeCards
iPushGame
iKnockGame
sendAllPlayers
sendToPlayer
sendToPlayers
fillPlayersWithState
fillPlayerWithData
gameIsOverNow
restartSwimming

Settings

playerName
settingsFilePath
settingsFile

readName
saveName
createSettingsFile
hasSettingsFile

Player

peer
name
state
cards
dealerCards
gamePlayerUINs
allPlayerCards

CardPool

cardPoolDeck
droppedCards

resetCardDecks
dropCards
takeNext3Cards
mixCards

interface
States

<<Singleton>>
Register

addPlayer
removePlayer
hasPlayer
getPlayer

players

Game

hostAGame
calculateNewCurrentPlayer
increasePushPlayerCount
resetPushPlayerCount
increaseKnockPlayerCount
fillPlayersWithNewCards
fetchNewDealerCards
evaluateWinningOrder
closeGame

currentPlayerUIN
hostPlayerUIN
ownGamePlayerUIN
gamePlayerNumber
gamePlayers
knockPlayerCount
knockPlayerUIN
pushPlayerCount
pushPlayerUIN

Conclusion

- The development of the middleware was time-consuming
- The development time of the application was moderate
- The network performance up to 7 MHs are acceptable
- Future work: intelligent profile model based on user preferences