2007

JAMOA

Java Mobile Application

Developing a mobile application using the Pervaho Framework

Boris Fritscher& Ulysse Rosselet

CONTENTS

INTRODUCTION

This application was developed as a full term project for bachelor students implying 6 ECTS Credits under the supervision of the DOP lab, Information Systems Institute, HEC Lausanne.

PROJECT'S GOALS

PRIMARY GOALS

The main objectives of this project are to create a mobile ad hoc application, running on mobile phones, using Java Micro Edition (ME) and the Pervaho Middleware. The requirements are to create an application that allows users to get in touch with each other and meet based on a shared interest and on their physical proximity as well as including elements for managing an urban tribe (trusted interaction) and public (anonymous) interactions.

MOBILE DEVICE

The primary requirement of the application is to run on mobile phones. With the majority of manufacturers integrating Java mobile technology into their devices as well as the growing use of such devices, the power available for more complex applications and their adoption by the public make it a good platform for deploying next generation software solutions.

PEER PROXIMITY

Hence the structure of the Pervaho middleware, the application must have different behaviour related to peer proximity. In our case, Pervaho specializes on providing location based publications/subscriptions that follow the user's position.

PEER TO PEER INTERACTION

The application must also offer support for users to get in touch with each other directly as well in a virtual context as in the real world.

TRUSTED/ANONYMOUS

The application must support trusted interaction, i.e. allowing the user to network with chosen partners; and anonymous interaction, where interactions happen with anybody.

SECONDARY GOALS

By using the Pervaho framework, determine its strengths and weaknesses. Implement and test its actual possibilities and identify missing key features.

IDEAS OF APPLICATION

GETSWEATY

DESCRIPTION

The application provides support for the search of sports partners in a relative short time horizon and in a given proximity area. The application contains a predefined list of sports and it is possible to record friends and organise them in groups. The user is notified of their presence when they are connected and he/she can monitor their position using the map function. He can also define his personal profile, including his name, phone number and the sports that he practices and his corresponding level.

When he publishes an advertisement the user chooses the sport, start and end time, area and anonymous/trusted criterion, i.e. a partner in the public network (anonymous) or in his friends network (trusted). The latter allowing the user to select a group of friends to which the ad will be visible.

When a match between two ads occurs, both partners get notified, receiving each other's contact information. They can get in touch with each other and discuss the terms of the meeting.

PURPOSE/MOTIVATION

The application aims at fulfilling the following needs:

- Meet a partner with an adequate level to do sport quite immediately in a given proximity range.
- Do any sport, but the identity of the partner is the most relevant criterion.

BUDDYRADAR

DESCRIPTION

This application offers a service which notifies the user of his friends' location within a close range (about 100 meters). The friends are displayed as icons on the screen.

PURPOSE/MOTIVATION

The application fulfils the following need:
Keep in touch with your urban tribe and be able to quickly locate your friends.

EXAMPLE

When the application is turned on, the user is visible to the members of his tribe. He can set his mood to indicate his sociability. The service allows users to contact each other directly.

TRUSTED/ANONYMOUS

This application is a pure trusted, group based application.

JAMNOW

DESCRIPTION

This application allows musicians to meet and to form groups during jam-sessions.

The application offers a virtual support to the creation of balanced groups. Therefore it must include rules defining the combinations of instruments which are promising (e.g. bass and drums, but not drums and drums).

The application offers two modes: active, research of partners; and passive: listening for other's requests.

PURPOSE/MOTIVATION

The aims of the application are:

- Satellite (EPFL's bar) organizes monthly jam-sessions which draw a crowd of musicians. Unfortunately people tend to play almost every time with the same known partners. Given the ambient sound level and the overcrowded space, such an application makes sense to simplify the interactions between musicians.

EXAMPLE

On the first run the user defines his instrument(s) and the music genres he mostly plays, i.e. his profile.

The default mode is passive.

To create an ad:

After choosing the genre he wants to play the user posts the message. A match occurs based on the following criteria: correspondence of genre and acceptable/promising combination of instruments .

If an active user has posted an ad corresponding in instrument and genre, both users get notified, receiving each other's contact information.

The passive users whose instrument and genre correspond to the ad are notified of this proposal. They can accept or refuse their contact information to be transmitted to the advertiser.

TRUSTED/ANONYMOUS

This application is interesting at the anonymous level but it is less relevant at group level, the contacts more likely to be established through text messages or phone calls.

EXTENSION

We can imagine posting the ad before the jam-session, to bring other musicians to the event.

REMARKS

This application would be mostly useful to heat-up the jam-session.

The need for this kind of application is not very strong.

EXPRESSYOURSELF

DESCRIPTION

The application allows to organize ad-hoc debates with anonymous conversational partner.

People who join the discussion become "emitters", in the sense that if the initiator leaves, the ad is still attached to the remaining discussion group.

PURPOSE/MOTIVATION

The application aims at fulfilling the following needs:

Debate and confront one's opinion with people differing from the usual friends/acquaintances, who generally think like us.

EXAMPLE

The initiator publishes a topic proposal. Users with the activated application get notified of the debate proposal as soon as they are in a given proximity. They can accept or deny joining the debate. If they accept, they receive the initiator's contact information and they become active members of the discussion, i.e. their phone relays the initiator's proposal.

TRUSTED/ANONYMOUS

This application is anonymous.

TRAINRIDER

DESCRIPTION

This service aims at permitting two persons to find each other in a train, when getting on board at different locations.

PURPOSE/ MOTIVATION

The application aims at fulfilling the following needs:

Fix the situation where we realize, once we got off the train, that we have been travelling in the same train as one of our friends.

The user can indicate if he wants to chat, or set "do not disturb" mode.

EXAMPLE

At Geneva, Alex gets into the St-Gall train. He launches TrainRider and sets his travel-axis (Geneva – St-Gall).

Bill gets into the train in Lausanne and goes to Zurich and activates his application too. A few minutes before the arrival of the train in Lausanne, Alex and Bill are notified that they travel in the same direction. They can then get in touch with each other through text message or phone call in order to meet, or they can use the application's distance monitor to find each other. One can also signal to the other that one is busy and therefore not available for talking.

EXTENSIONS

It would be possible to include a non-exhaustive train database. In this case the user selects the train in which he travels (start time and destination define the train). This avoids any possible misunderstanding concerning the hour, the platform, and so on. Unfortunately, this solution is less flexible in case of train change and implies more manipulation of the phone, making its use less intuitive.

TRUSTED/ANONYMOUS

This application seems to fit well in a trusted/group situation where we know the people with whom we want to use this service.

APPROACH

BRAINSTORMING

In order to bring out all the ideas about this project, we first did quite a lot of brainstorming, imagining all the craziest and weirdest applications possible, drawing mind-maps and freely discussing every idea.

COLLABORATIVE TOOLS

To ensure a good organisation of the project and an easy access to all the ideas and resources, we used two internet based tools: a web platform and a repository server.

WEB PLATFORM

The web platform is built around "an enhanced wiki and issue tracking system for software development projects."[1] The wiki offers an overview of the project as well as an easy way to consolidate the entire information related to the project.

The timeline gives instant access to the latest changes that were made to the repository, wiki, tasks, and questions. This way, we keep trace of any modification.

The ticket system is used to assign tasks, questions or extensions. The accomplishment of these tickets is visible on the roadmap view, where we can also define milestones.

As it is directly connected to the repository server, the trac platform allows us to browse the application's sources directly from any web browser. Thank to this possibility, we can always consult the latest version of the files.

REPOSITORY

The repository server is used for revision control and centralization of the source code. Furthermore, its access and control functions can be seamlessly integrated into the IDE.

SUPERVISION

WEEKLY MEETINGS

During the whole project, we scheduled meetings with Professor Garbinato and his assistant, Adrian Holzer on a weekly basis.

These meetings were useful not only to resolve the implementation problems we encountered during the development, but also to help define more precisely the specifications and the goals of this project.

[1] The Trac Project : http://trac.edgewall.org/

APPLICATION

CHOICE OF THE APPLICATION

We decided to develop GetSweaty because it is the application which fulfills in a most complete way all of the requirements. Also, in a universitary environment, it seemed to be a usable application.

USE CASES

MANAGING USER PROFILE

After launching the application, the user can access his personal settings through the *Profile* icon in the main menu. On the first run of the application, or if no profile data is found, he is prompted to enter his name, phone number and configure the sports he plays with their corresponding levels.

Profile editing:

- The name and number field are directly editable.
- The sports menu is accessible via the *View sports* icon and allows adding, deleting or editing a sport.

Managing sports:

- Adding a sport is done via the *Add sport* command from the menu.
- A drop-down list of predefined sports allows the user to choose the sport. The level is set using a gauge, which responds to the left and right arrows of the mobile phone.
- In *Edit sport* mode, the level is the only adjustable thing.
- If the command *Cancel* is pushed: no changes are stored and the application returns to the *View sports* menu.
- If the command *Save* is pushed: the application controls that a sport is selected and in this case, the changes are saved to persistent storage and the *View sports* menu is displayed, the new sport being listed.

POSTING ADS

Once the user sets his/her profile, the search for partners can begin. The advertisement menu is accessible from the corresponding icon in the main menu. If there is no existing ad, the user can immediately create a new one, otherwise, a list of the existing ads is displayed and he must press/activate the *New ad* command.

Creating an ad:

- All the sports configured by the user are displayed and can be selected thank to a checkbox.
- The visibility (*Public* or *Friends network*) is set using the left and right arrows. By default, the visibility is set as public.
- If the *Friends network* is selected, the user must choose a group.
- Start and stop time are set using special date fields.
- The publication range is set using the following predefined values:
    - City : 5 km
    - Campus range : 2 km
    - Block : 1 km (default)
    - Building: 250 m
- The ad is accepted if it contains at least one sport and if stop time is after start time. If the user chooses *Friends network*, he must also select a group.

- The *Preview* command displays the complete ad for the user to check it before publishing. He can *Cancel* and come back to the ad creation screen or *Publish* his ad.

Managing the ads:

- An ad can be created or deleted using the menu which is accessible by pushing the right screen key of the handset.

GETTING MATCHES

A match occurs when the following conditions are verified:

- one sport in common
- level within a range of plus one / minus one
- friends network or public network matching
- the time overlapping by at least twenty minutes.

When a matching pair of ads is detected, it is dated and kept in memory until the application closes.

- The user receives a message containing the sport, the name of the partner, his number, the overlapping duration of their start and stop times as well as each other's begin and end times. He can choose to contact the partner or he can dismiss the match if he prefers waiting.
- The *Matches* menu is accessible through the corresponding icon in the main menu. This icon appears only if one or more matches occurred in the application.
- From this menu, the user can contact the partner or if he has found a partner, he can delete the ad which corresponds to a match.

ORGANISING FRIENDS

The *Friends* menu is accessible through the corresponding icon in the main menu. The existing groups are displayed.

- By selecting a group, one can access the friends it contains. They are listed with an icon indicating their connection status (green: online, red: offline) and their distance in meters; which is based on the coordinates updated through the received friend's status notification messages.
- The user can edit, contact or delete a friend using the commands from the menu.
- In edit mode, the user can rename the friend and add him to a new group.
- The user can use the *Map* command to display a radar showing the position of his on-line friends using the friend's coordinates.

Adding a friend:

- After pushing the *Add friend* command, the user is prompted to enter the friend's phone number. A request is sent to the friend, who can accept or deny this proposition.
- If the friend accepts, they are added to each other's friend list.

TECHNICAL DETAILS

RMS

In order to store data persistently on the mobile phone, we use the *Record Management Store* of J2ME.

We use three different stores:

PROFILE STORAGE

2 fields:

- "ID1" Name
- "ID2" Phone number

FRIENDS' LIST STORAGE

- N-Tuple which data are separated with a semi-colon :
- Friend.phonenumber ; Friend.name ; Friend.group1;…; Friend.groupN;

When the friends list is retrieved from the RMS, the groups are created and all friends are listed in the default group *All*.

USER'S SPORTS STORAGE

- N-Tuple where data are identified by their length: identifier of the sport on the three first bytes and level on the fourth.
- The sport's name is hard coded inside the application and can be retrieved through the corresponding identifier. Throughout the application the sport data are always evaluated through their identifier. The benefits of this implementation are that it allows a match between applications of different languages.

LPSS

LPSS communication service is initialized at application start-up. This service is used for advertisement, friends' connection status and adding new friends. All the following treatments are implemented in *LPSScom.java* and *Ad.java*.

AD CREATION

When an ad is created, the following pub/subs are generated:

FRIENDS NETWORK

For each sport that is selected:

SUBSCRIPTION

For each friend in the selected group:

s(type: ad, level-1, sport, visibility, "pn"+userPhonenumber, "pn"+friendPhonenumber , range)
s(type: ad, level, sport, visibility, "pn"+userPhonenumber, "pn"+friendPhonenumber , range)
s(type: ad, level+1, sport, visibility, "pn"+userPhonenumber, "pn"+friendPhonenumber , range)

PUBLICATION

P(type: ad, level, sport, visibility, userNumber, userName, start time, end time, “pn”+friend1, …, “pn”+friendN, range, TimeToLive[2])
With friend1 to friendN, the phone number of the friends in the selected group.

PUBLIC NETWORK

For each sport that is selected:

SUBSCRIPTION

s(type: ad, level-1, sport, visibility, range)
s(type: ad, level, sport, visibility, range)
s(type: ad, level+1, sport, visibility, range)

PUBLICATION

P(type: ad, level, sport, visibility, userNumber, userName, start time, end time, range, TimeToLive)

ADDING FRIEND

On application start-up, a subscrition for all friend request is created:

S(type: friendRequest, to_phone: userPhoneNumber)

When the user adds a new friend, the following pubs/subs are generated:

PUBLICATION

P(type friendRequest, to_phone: friendPhoneNumber, from_phone: userPhoneNumber, from_name: userName, range: earth)

SUBSCRIPTION

S(type: friendRequestAnswer, to_phone: userPhoneNumber, from_phone friendPhoneNumber)

Then the requested friend’s application generates the following

PUBLICATION (2)

P(type: friendRequestAnswer, to_phone: userPhoneNumber, from_phone: friendPhoneNumber, message: 1 or 0, range)

Due to privacy concerns the friends name is only added if he accepted the request (message is set to 1)

When the user’s application receives the friend request answer, it unpublishes the publications and subscriptions related to this request.

FRIENDS’ CONNECTION STATUS

[2] Not implemented in LPSS yet!

On application start-up, a new thread is created which publishes user's position every n milliseconds. The publication contains:

PUBLICATION

P(type: friendStatus, from_phone: userPhoneNumber, longitude, latitude, altitude, range: earth)

Also on application start-up or when a new friend is created, a subscription for each friend is generated:

SUBSCRIPTION

S(type: friendStatus, from_phone: friendPhoneNumber, range: earth)

In the same thread as the publication, the friends are set offline every m milliseconds to manage the friends' timeout.

FIRST VERSION LIMITATIONS

The location API (JSR-179) is not implemented in the majority of today's phones. Also in Switzerland, the providers block the communication ports. Therefore, testing was only possible in an emulated environment.

We decided to use a self created address-book because the access to the phone's contact list is supported yet by another API (JSR-75) which is presently scarcely available on today's phones. Also the fields which can be accessed vary on each phone.

The text message contact function could have been implemented using J2ME i/o functions. This would have implied recreating a sms application. Unfortunately, direct access to the phone's communication features is not available because no generic API exists for such purpose.

As the intended usage of the application is short time search, the start and stop times do not include a date but only hours. This makes it impossible to create an ad for the following day.

As it is a prototype, error handling is limited to the most necessary cases. And also we did not test all the combinations of possible errors.

The sport being hard coded in the application, it is not possible for the user define his own sports. This is done in order to ensure that matches occur, but it is a limitation for users who want to practice sports which are not predefined.

POSSIBLE EXTENSIONS

Testing the application, we noticed that the number of manipulations involved in creating an ad hinders the user's experience. Therefore a further version could include a *Quick ad* function which recalls the last ad created or uses a template to facilitate the ad creation process.

Concerning scheduling of the sport period, it seems more intuitive to have start time and duration rather than start and stop time.

The matches could be persistently stored in a history.

A new friend could be added directly from a match.

The map function of the friends menu could be extended in order to support the new Google Map for Mobile[3] service. Unfortunately no API is available to access the service from an application to this day.

The user profile could include more settings like choosing a personal picture or defining the user's hometown.

In every list of the application, a search/sort engine could be integrated. This allows to easily find a particular friend or sport.

The application could also include rules to select minimum number of players before displaying a match notification, in case of sports implying more than two players.

Depending on the standardization process of the mobile devices, direct interfacing with the phone's address book and communication functions is possible.

To improve flexibility and enhance the available sports, the sport list could be downloaded from a server instead of being hard coded in the application.

Concerning the contact friend function, we can also imagine creating a sort of chat-room integrated in the application. But this requires unlocking of the communication ports on the provider's side.

Most of these features could be extended even more in collaboration with an on-line web 2.0 application. This could give the possibility to rate users, synchronize and manage friends and sports lists, vote on new sports to be added, or plan sports on a longer horizon.

[3] http://www.google.com/gmm/index.html

PUSHING THE BOUNDARIES OF LPSS

After using LPSS we find these points are to be addressed with high priority:

The methods of LPSS need to be documented, e.g. describing values that can be passed as argument. For example: when setting an attribute on a new event, the key cannot be only numbers and even if the value of the event can be any object, in reality only one string can be passed. Some comments about such limitations would be welcome during the programming process. Javadoc is not an option!

The phone motion simulator consumes a large part of the computer's resources; we were forced to manually put its thread into a low priority mode.

We changed the coordinates' exchange values to the real float values instead of rounded integers everywhere. This enabled us to use the location API's azimuth and distance functions. The rounded integers values were imprecise and lead to erratic behaviour.

Unfortunately, it seems to disturb a bit the LPSS server who is now generating false matches. If Phomo is not activated and all coordinates have a value of zero, the server behaves as expected.

As for features which would be nice to be included in a future version:

Flexibility could be improved by allowing the use of more SQL operators (< > LIKE % %...). The IN operator would also be useful. These improvements would significantly reduce the number of exchanged messages and allow more server side treatments.

Security should be increased to include domains and other features to ensure privacy.

When implementing TTL, it could be interesting to include it into the subscriptions elements too.

On start-up, the client side LPSS communication module should either queue the incoming publications and subscriptions until it is ready to send them to the server, or have a feature to notify the application when it is ready to transmit.

CONCLUSION

The whole projet was quite a challenge and fortunately we managed to fulfill all the specifications decided in the first place. This wouldn't have been possible without a good collaboration and the project management tools we used, especially the Wiki and the repository server. Working with the Java Mobile Edition platform was an enriching experience but we came up against the problem of the different implementations of proprietary technologies and the lack of appropriate APIs. An effort of standardization is still needed on the manufacturer's side in order for third party developpers to build rich portable applications more easily.

The icing on the cake would have been the opportunity to test the application on real phones but, up to now, their features and the network's restrictions make it impossible. This would have been an uncompromising test of the application as well as a "first flight" for the Pervaho framework.

Finally, such applications are certainly significant in today's context. With the growing number of metropolitan people living alone, the use of the latest mobile technology to support real world interactions makes it a powerful vector of social networking. Most important, our application is not confined to virtual interactions like many existing services but is truly dedicated to real world activities. During the past decade, we have seen a lot of technology products which were meant to replace physical interactions with virtual ones, making technology a factor of physical isolation. Now is the time for technology to be considered as a vector of communication and a powerful meeting solution. That is what Jamoa is here for!

APPENDICES

PREDEFINED SPORTS LIST

Baseball
Basketball
Bowling
Boxing
Cycling
Dancing
Fishing
Fitness
Golf
Handball
Football
Rowing
Sailing
Running
Judo
Swimming
Tennis
Volleyball
Climbing
Ping-pong
Pool

DEVELOPMENT ENVIRONMENT

TECHNICAL SPECIFICATIONS

- Netbeans 5.5 with mobility pack
- Java Wireless Toolkit
  - 2.2 Linux
  - 2.3 beta Windows
- J2ME polish 2.0 beta 3
- Windows XP service pack 2
- Ubuntu 6.X
- Java Application Server 9

APPLICATION PACKAGE

Package	**Content**
application	Model and communication classes
application.db	RMS interface classes
application.graphics	MainMenu
application.graphics.ads	Screens for the advertisement management and confirmation alerts
application.graphics.friends	Screens for the friends management and confirmation alerts
application.graphics.profile	Screens for the profile management and confirmation alerts
resource folder	All the pictures and polish.css
lpss packages	clientLPSS, clientLPSS.com, clientLPSS.com.messages, clientServer, locationSimulator, lpss

FAQ

PORTSCONFIG

GETSWEATY

Pervaho server to contact

```
clientLPSS.com/Config.java localhost:4410
```

Phomo server to contact

```
locationSimulator/ConfigLocationSim.java localhost:4430
```

PHOMO

Server port for the game

```
config/Config.java 4420
```

Listening port for mobile connection

```
simulator/ConfigLocationSim.java 4430
```

PERVAHO SERVER

Listening ports of the server

```
serverLpss.com/ServerComTCP.java 4410
```

```
serverLpss.com/ServerComUDP.java 4410
```

TEST PROCEDURE

Here's the procedure you need to follow in order to have correct initialisation of the phones and correct start of the application. After opening Netbeans:

- Run phomo
  - Launch simulation
  - Task manager: set thread priority to low (hence the heavy resources consuming of phomo)
- Start java application server
- Run pervaho server
- First phone:
  - Run (build) GetSweaty,
  - lauch “generate data A”
  - launch GetSweaty.
- Second phone
  - Quick Run With…
  - generate data B
  - launch GetSweaty

BUILD.XML

JAR NAME

Jar name in the info section must be the same as project name in order for quick run to work.

jarName="GetSweaty.jar"

LOCATION JAR

The location.jar must be included in the build section

binaryLibraries="location.jar"

DEBUG (PREPROCESSING)

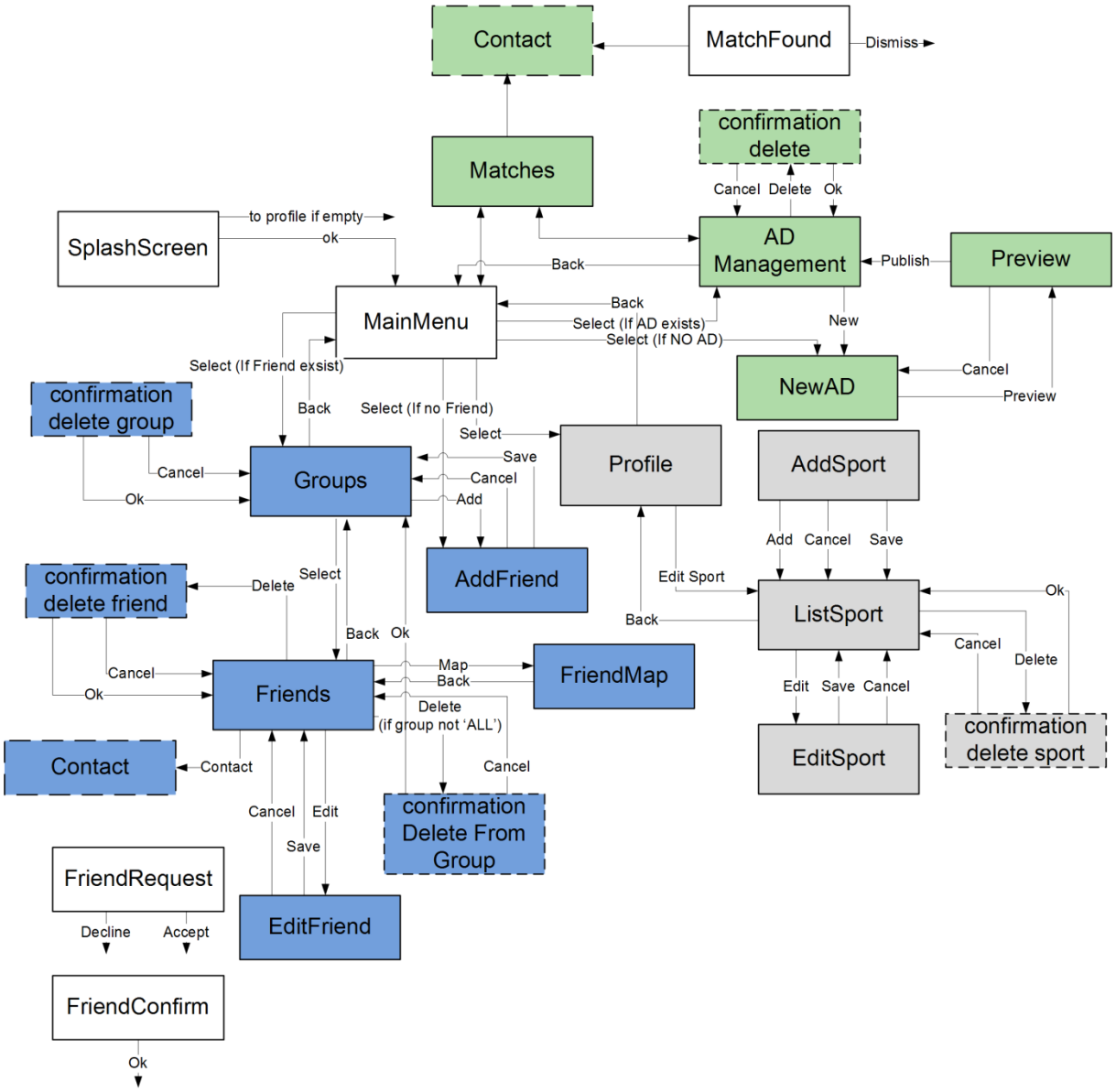
Enables log output of most communications actions.

Activate the commands *Debug make ad* and *Degug make match* in the advertisement menu.

put <property name="VIEWLOG" value="true" /> in your build.xml under <target name="test"> for example to view debug functions

DIAGRAMS

APPLICATION FLOW



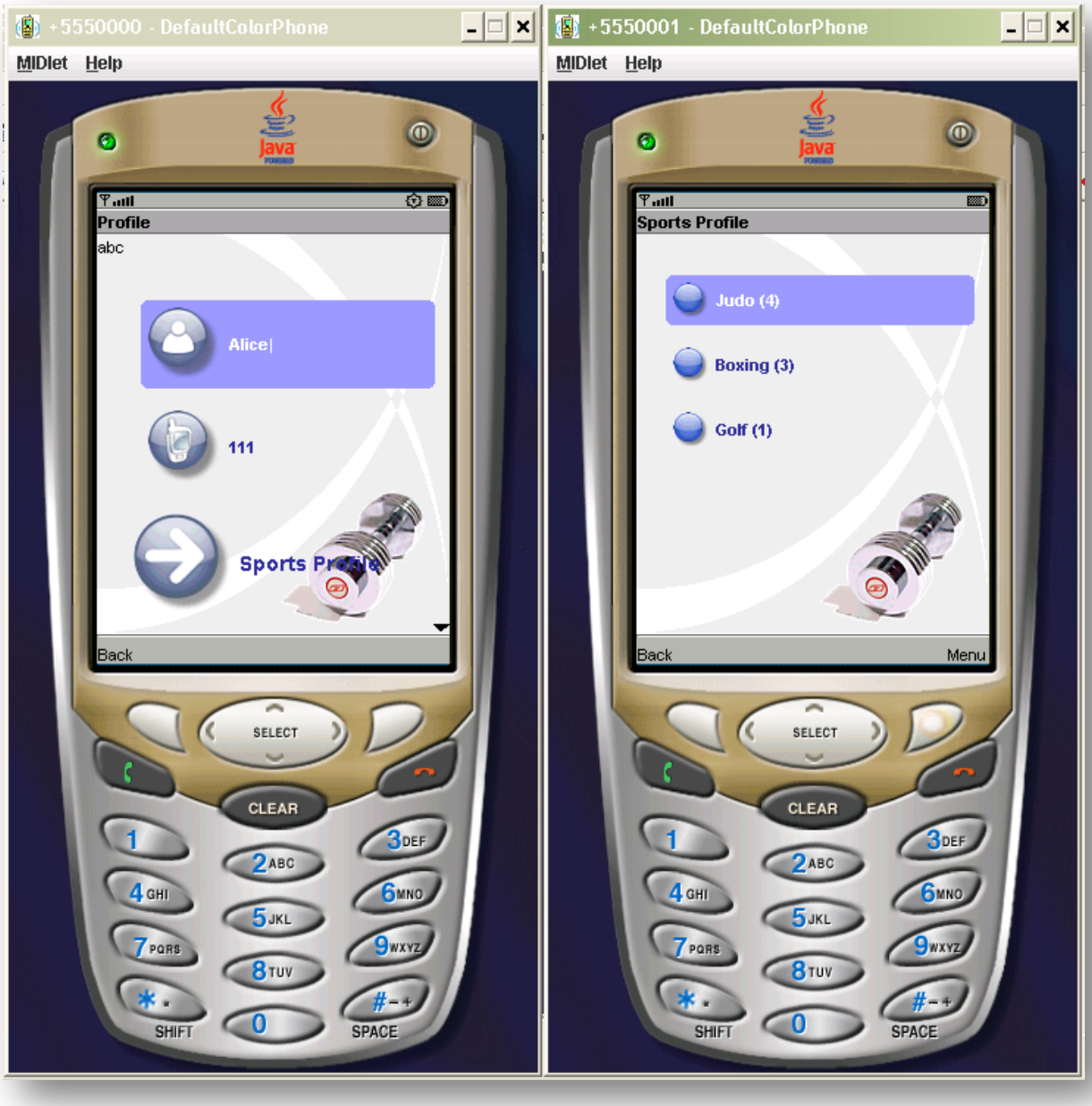
CLASS DIAGRAMS

See enclosed paper

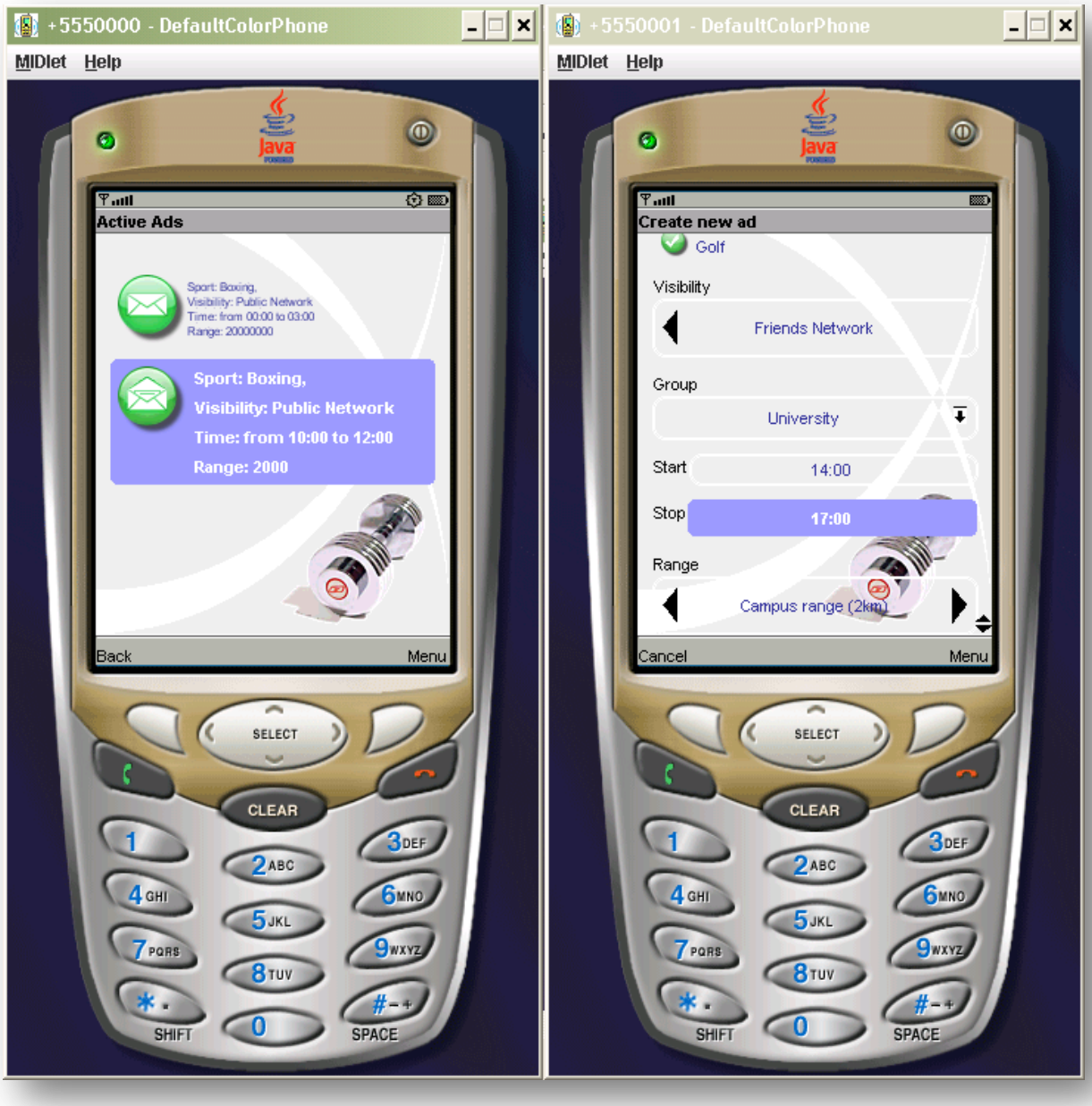
SCREENSHOTS

MAINMENU

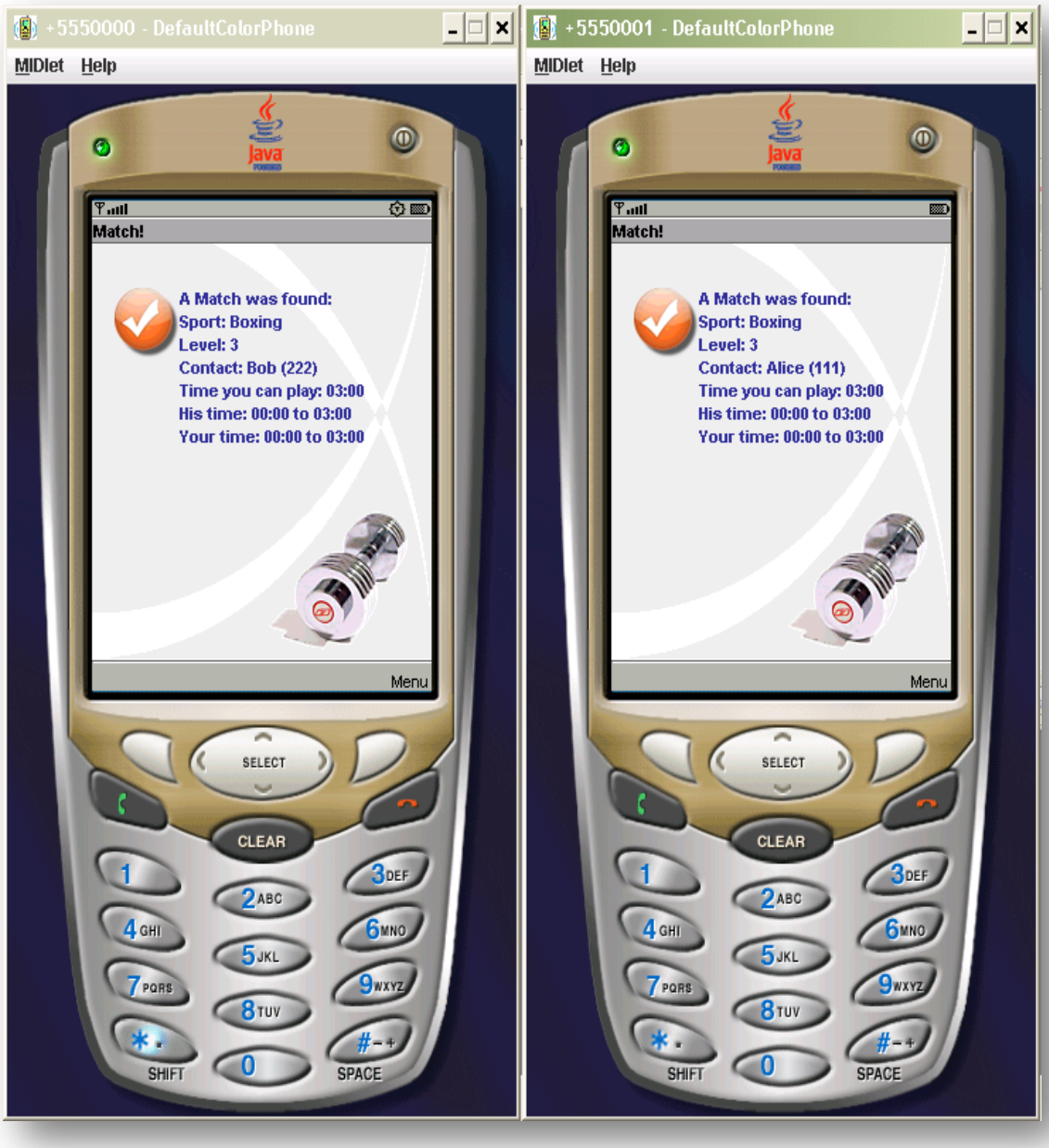
PROFILE SETUP



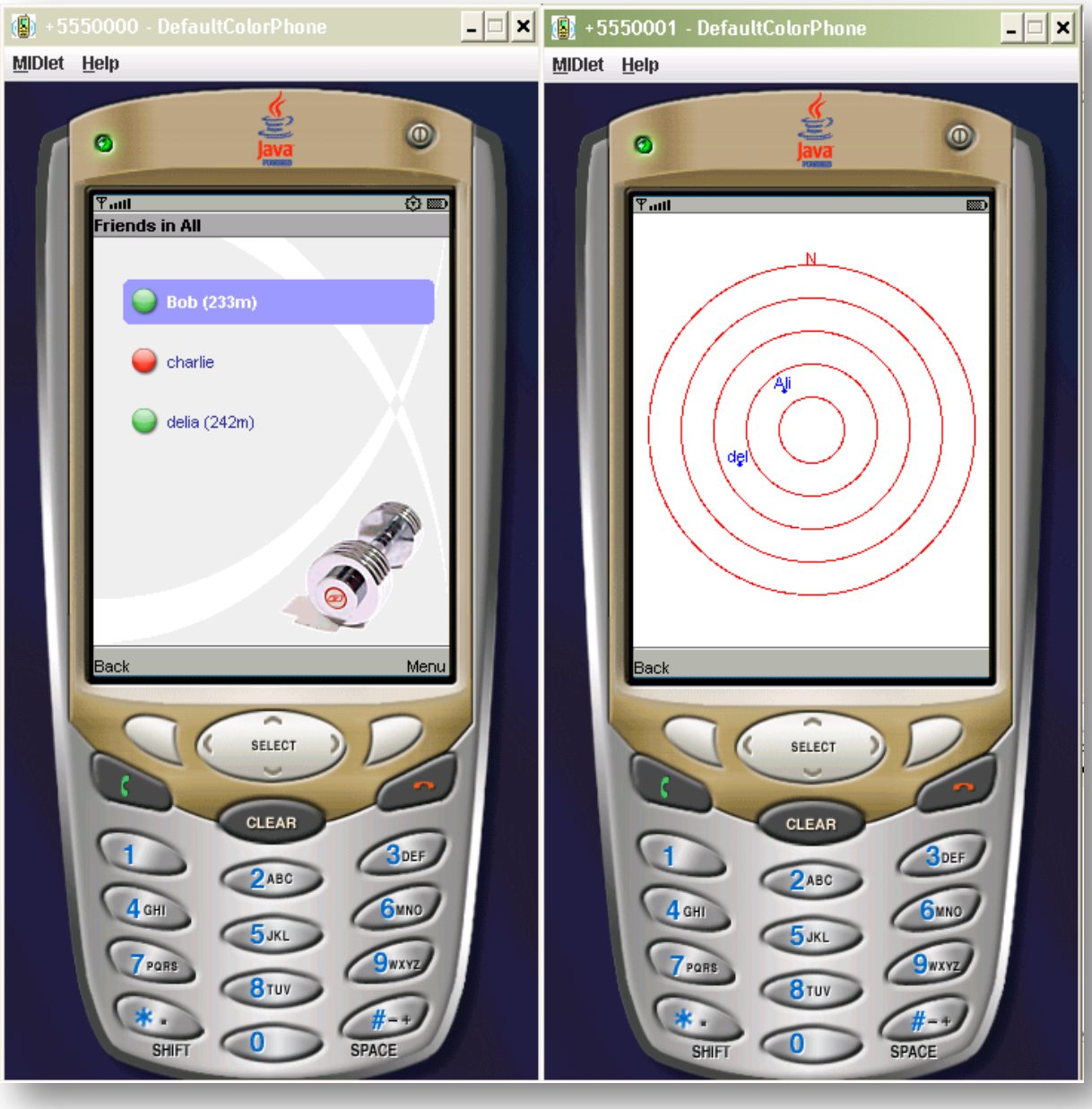
AD MANAGEMENT



MATCHES


+5550000 - DefaultColorPhone
MIDlet Help
Match!
A Match was found:
Sport: Boxing
Level: 3
Contact: Bob (222)
Time you can play: 03:00
His time: 00:00 to 03:00
Your time: 00:00 to 03:00
Menu
SELECT
CLEAR
SHIFT
SPACE
+5550001 - DefaultColorPhone
MIDlet Help
Match!
A Match was found:
Sport: Boxing
Level: 3
Contact: Alice (111)
Time you can play: 03:00
His time: 00:00 to 03:00
Your time: 00:00 to 03:00
Menu
SELECT
CLEAR
SHIFT
SPACE

FRIENDS


+5550000 - DefaultColorPhone
MIDlet Help
Friends in All
Bob (233m)
charlie
delia (242m)
Back
Menu
SELECT
CLEAR
+5550001 - DefaultColorPhone
MIDlet Help
N
Ali
del
Back
SELECT
CLEAR
SHIFT
SPACE