Cross-Platform Mobile Services using the SMILE Middleware and the MOVE Framework
Stefano Salsano, Giovanni Bartolomeo
University of Rome “Tor Vergata” Dpt. of Electronic Engineering, Rome, Italy

Abstract
This paper describes a communication middleware tailored to the needs of mobile devices called SMILE and its application in a framework for the design, implementation and deployment of cross-platform mobile services called MOVE. Both the SMILE middleware and the MOVE framework are open source. The SMILE middleware is based on the principles of Service Oriented Architecture (SOA) and borrows from SOA the WSDL language for interface definition. SMILE provides support for asynchronous messages, either using a transport based on SIP protocol or on HTTP protocol and it is complemented by automated code generation tools. The MOVE framework offer a large set of feature to implement Location Based and Community Based Services for mobile devices. Currently supported mobile platforms for SMILE and MOVE are Symbian, Android, Blackberry and Windows Mobile.

1 Introduction
In the latest two years we have witnessed the explosion of mobile computing. Internet and e-mail access from a (smart) phone have moved from a task for specialists or for business men to a service offered to a larger public and they are going to become a commodity for the mass market. Fancy smartphones and a plethora of “mobile applications” that exploits mobile data access are now available to the users, gaining them to mobile computing. The cost of mobile data access has been falling down with the introduction of flat contracts (typically with usage caps in a given period). The forecast of overall mobile data traffic growth in the next few years are impressive, so as to seriously challenge the capacity of current and planned network infrastructure of mobile operators.

The mobile computing arena is characterized by a large fragmentation with several competing platforms (e.g. Symbian, Android, Apple iOS, Blackberry OS, Windows Mobile/Windows Phone…). The development process of a mobile application is typically platform specific, tied to the specific language and to the services and APIs offered by the platform. This complicates the interoperability of applications over the different platform as the communication layers needs to be independently developed. On the other hand, if we analyze the communication needs of mobile applications, we can identify a common base of requirements that can be fulfilled by a common and “cross-platform” middleware solution. Such middleware can ease the development of mobile application both in the single platform case and in a “multi-platform” environment.

In section 2 we analyze the requirements and we describe the design and the implementation of a communication middleware for mobile applications named SMILE (Simple Middleware Independent LayEr). The SMILE middleware is based on concepts from the Service Oriented Architecture / Web Services paradigms. It supports asynchronous notifications and a lightweight messaging mechanism. The SMILE middleware is complemented by a set of software engineering tools.

In section 3 we make a step further in our attempt to achieve a cross-platform mobile application development and execution environment. We present a full application development and execution
environment called MOVE (Mobile Open and Very Easy). The MOVE framework builds over SMILE, adding a GUI framework tailored to the mobile applications need and a way to access the device capabilities needed for mobile applications (e.g. access to location information, camera and so on). MOVE is based on Java Micro Edition CLDC/MIDP and it allows to run the same application on Symbian, Android, Blackberry OS and Windows Mobile1.

Both the SMILE communication middleware and the MOVE application development and execution framework are open source and can be freely downloaded.

2 SMILE - The communication Middleware

The definition of our middleware started from the set of requirements listed in Table 1. We will refer to this list in order to provide the rationales for our choices and to match the features of the proposed solution with the listed requirements.

| Req. 1 | The mobile devices should have the capability to receive and act on information asynchronously. We refer to this feature as “notifications” services, or “push services”. |
| Req. 2 | Mobile application may need to communicate in a peer-to-peer fashion, i.e. from a mobile device to one or more mobile devices. |
| Req. 3 | The processing and bandwidth requirements should be as limited as possible in order to adapt to the capabilities of mobile devices and mobile access networks. |
| Req. 4 | Support of mobile devices accessing behind NAT2 and firewalls should be provided as much as possible. |
| Req. 5 | A standard way to specify the interactions among the distributed elements is highly desirable to facilitate the design of services and application and to foster the interoperability. |
| Req. 6 | Interoperability with canonical Web Services using SOAP: a mobile client should be able to access a SOAP Web Service through the middleware |
| Req. 7 | Mobile-to-web interactions needs to be taken into account: it should be easy to develop applications and services that include mobile users and “desktop” web users. |

As for requirement 1, proprietary technologies for push services have proven very market-successful in the last years, the first and most known being the Blackberry Push Service (built on WAP push specifications but using a gateway and a proprietary technology to connect to Blackberry devices). Following Blackberry example, other platform have introduced equivalent services (e.g. the Apple Push Notification Service for iPhone, exploiting a constantly-open IP connection with iPhone devices). The interoperability among these proprietary platforms obviously has not been considered as a design goal, therefore it is rather troublesome to build cross-platform services and applications based on push services.

Ideally, the Service Oriented Architecture (SOA) principles could provide simple and elegant

---

1 If a Java Micro Edition Virtual Machine is built in or installed in the Windows Mobile device.

2 In operator or corporate networks, mobile devices often are assigned private IP addresses behind a NAT and are normally not reachable from the outside world.
guidelines for the realization of interoperable distributed mobile applications, meeting requirement 5. In particular they provide a standard interface definition language (WSDL [1]) together with a consolidated set of interoperable tools and software libraries to ease the application development. Unfortunately, in practice SOA implementations do not fit at all the mobile environment, mainly due to the choice of the SOAP/XML format for message encoding and to the usage of HTTP request-response pattern as the underlying transport mechanism. The former reveals to be too heavy and bandwidth wasting for resource constraints devices (requirement 3), whereas the latter does not provide, by its own, suitable client addressing mechanisms (requirement 2) nor push notification capability (requirement 1).

We believe in the value of a solution that is able to meet all the above requirements, combining, from one hand, the advantages of a standard interface definition language, compatible with existing SOA standards and programming paradigms (requirements 5 and 6), and from the other hand optimized techniques fitting the specific needs of distributed mobile applications (requirements 1 to 4).

2.1 SMILE overall picture

The Simple Middleware Independent LayEr (SMILE) was originally intended as a lightweight framework allowing to decouple the functional model of a distributed application from the underlying middleware platform used to run the application itself. In this light, SMILE framework can be assimilated to an “abstract platform” [2]. As shown in Figure 1, the SMILE framework provides interfaces to the applications which in turns exploit services from a “real” underlying middleware platform (through the so called “bindings”). This approach allows to easily port the same distributed application over different middleware platforms.

![Figure 1. The two facets of SMILE: the APIs and the bindings](image)

2.2 SMILE APIs, SMILE processes

SMILE definition of interfaces is inspired by SOA principles and uses WSDL as interface description language. The core of SMILE consists in communication primitives based on WSDL specifications. In WSDL a “portType” represent a collection of “operations” offered by an endpoint. There are four types of operations, classified according to the point of view of the endpoint whose service is specified in the WSDL document. A “OneWay” operation consists in an asynchronous message originated by a remote entity (i.e. the “client”) and received by the endpoint
A “RequestResponse” operation consists in a request message from a remote entity (client) received by the endpoint (server), followed by a response message sent by the server to the client. Alternatively, a “fault message” may come to the client if something has gone wrong at the server side. There is a logical correlation between the request and the response: the execution environment creates a “session” and provides the corresponding response or fault in an unambiguous way as reply to a given request. A “Notification” operation consists in an asynchronous message sent by the (server) endpoint towards a remote entity (that has implicitly or explicitly registered for such notifications). A “SolicitResponse” operation consist in a request message sent by the server endpoint, followed by a message sent by the remote entity and finally received by the server endpoint.

For “OneWay” and “Notifications” operations SMILE defines the send method and the onReceived callback, (triggered in the receiver side). The send method is used to send asynchronous messages, it is not blocking and therefore it realizes an “asynchronous” interaction pattern. For “RequestResponse” and “SolicitResponse” operations, SMILE uses the blocking doRequest method on the calling side and the onRequestReceived callback on the receiver side, which returns the corresponding response (or raises an exception if a fault is received). SMILE takes care of correlation between request and response. Thus, “request messages” are synchronous, and implicitly confirmed as soon as a response message is received. Remote Procedure Call (RPC) is implemented using the request/response pattern.

SMILE entities that receive the asynchronous and request messages defined in a WSDL portType are called “SMILE process”. Other than responding to events originated whenever messages are received, each single SMILE process, may execute proactive business logic. In particular, a SMILE process implements three callback methods which are called in sequence. These callbacks are: setup, called as soon as the peer starts, and usually implemented to perform start up operations; doBusiness, invoked as soon as setup returns, to perform core activities and takedown called after the completion of doBusiness (allowing to host release of resources, memory clean-up and finalization code). The SMILE processes are executed in a “run-time container” which in turn is executed on the hosting platform (e.g. a mobile device with a given operating system). More than one SMILE process can coexist (and interoperate) within the same run-time container. As shown in Figure 2, a SMILE process can invoke operations on a remote processes of a given type (the interface being defined in a WSDL document), and an addressing mechanism is needed to identify the target run-time container and the specific process running in the container.
Functional features provided by each process are called “services” and can be published in a directory service called “Yellow Pages” using service publication and discovery primitives (publish, search, delete). More details on this optional service discovery component of the SMILE architecture can be found in [3].

2.3 SMILE-JS and -JSH bindings

In middle 2006, we started the porting of SMILE API to mobile devices, having a Java Micro Edition CLDC 1.1 MIDP 2.0 platform as minimal target and taking into account requirements 1 to 4 listed above. After experimenting with different existing middleware platforms for mobile device we found that none met all these goals; as a consequence, we started developing our own SMILE bindings for mobile: SMILE-JS and SMILE-JSH respectively exploiting SIP and HTTP as transport protocols.

Let us now come back to requirements 1 (asynchronous notifications) and 2 (“peer-to-peer” communication pattern). They call for an addressing mechanism that can identify a SMILE process wherever it is located (considering also requirement 4 about NAT and firewalls!), so that it is always possible to send a message to it, without having a communication session established beforehand. In canonical Web Services, addresses are in form of HTTP URLs. This solution works fine if the entity to be contacted is a Web Server, but does not fit very well to the dynamic nature of mobile computing. Therefore, we preferred to investigate a different approach, based on SIP addressing. SIP is a signalling protocol to establish calls and multimedia sessions between user agents on end user equipments, even behind NATs and firewalls. SIP is widely used in mobile operator networks as a result of 3GPP standardization activities. SIP user agents are identified through their Address of Record (AoR), which is in form of a URI (an example SIP address is: sip:alice@iptel.uroma2.it). Like email addresses, AoRs are assigned by a provider to SIP users to enable them to initiate and receive incoming communication sessions. The SIP infrastructure natively supports the mobility of terminals and users, taking care of the association between a SIP address and the SIP user agent that is intended to handle the communication directed to the address at a given point in time. It includes authentication/security aspects and well established techniques for supporting the communication across NATs.
A first solution we considered for SMILE-JS was to assign a SIP address to each SMILE process. However, a one-to-one mapping between SMILE processes and SIP agents would have resulted in an inefficient use of resources, requiring a new AoR to be assigned to each new process created in a run-time container. To allow a more efficient and flexible mapping of many SMILE processes within each single run-time-container, we defined two additional fields to be used in SIP addresses in order to distinguish among different SMILE processes exploiting the same SIP agent: pType, containing information about the “kind” of implemented services (similar to a class name in an Object Oriented programming language) and pName, unique “address” of a specific process, used to distinguish between many instances of the same “kind” of service. An example of the resulting SIP address for a SMILE process is the following:

<sip:alice@iptel.uroma2.it;pType=org.istsms.mem.MemPeer,pName=0a87f4>

The SIP infrastructure elements and the SIP stack for both mobile devices and server side we used in our implementation of SMILE-JS are based on the open source MjSip project. This infrastructure, shown in Figure 3, is composed of a Registrar and Proxy server, which performs authentication, maintains the mapping between AoRs and IP addresses, and routes SIP messages to recipients, and by a SIP Session Border Controller (SBC) able to route incoming SIP messages to peers behind NATs. The optional “Yellow Pages” server supporting SMILE process discovery is also shown in the figure.

Figure 3. SMILE deployment architecture

Note that the use of SIP protocol is well suited for a deployment within a 3G operator network that implements the IMS (IP Multimedia Subsystem) architecture. In fact, we have demonstrated it porting our SMILE middleware in an IMS environment [4] using the Open IMS Core platform.

Taking into account the requirement 3 (bandwidth/processing limited devices), SMILE-JS adopts JSON as its serialization format. JSON is a text format, based on a subset of the JavaScript Programming Language specifications. It represents a good trade-off between performances (such as compactness and encoding time, in contrast e.g., to XML) and human readability (compared to binary formats).

Sometimes SMILE-JS is not a viable solution, due to the missing of UDP protocol support in the
end user equipment or in the operator or corporate networks. For example, Java Micro Edition
CLDC/MIDP specifications mandate only the support of HTTP connections, being optional the API
to manage TCP and UDP connections. Even when it is possible to use UDP on the mobile device,
UDP could be blocked by network operators or not allowed by firewalls in corporate networks. In
some cases, only web browsing is permitted, maybe passing through a proxy server.

Therefore, in order to better address the requirement 4 (support of NAT/firewall) and to support as
many mobile device platforms as possible, we started considering to use of HTTP as transport
protocol. HTTP was not originally meant for “bidirectional” communications, but the advent of
Web2.0 brought new techniques allowing browser based applications to receive incoming
“notifications” from a web server. This set of techniques, exploiting HTTP requests and responses
as stream channels are often referred under the umbrella term “Comet” [5][6]. Unfortunately, we
noted that existing Comet techniques, though useful to implement server based notification toward
web clients, do not directly represent a complete solution for mobile internetworking, as they lack
of routing and forwarding mechanisms which are instead in SIP.

Thus, in SMILE-JSH (JSON+SIP+HTTP) we used a Comet technique for bidirectional HTTP
communications together with the SMILE-JS infrastructures. This combines the advantage of SIP
addressing and routing with HTTP based bidirectional communications for both mobile and web
based applications. SMILE-JSH messages are tunneled on HTTP channels, towards a HTTP
gateway which takes care of interfacing with the rest of SIP infrastructure (Figure 3).

Our gateway works using two HTTP channels, the first one for downstream and the second for
upstream. It also uses cookies to identify different client sessions. The details on the proposed
Comet solution are described in [3]. Upon completion of HTTP connections establishment, and
after a successfully SIP authentication, the gateway permanently associates the HTTP channels to
the SIP address of the client. As a result, SMILE processes on the mobile device are able to
exchange SMILE messages with any other client, either implemented as a SIP user agent or
reachable through a HTTP gateway.

This approach is also well suited to support rich web clients running into web browsers in desktop
and notebook. Therefore we have realized an implementation of SMILE libraries (for the SMILE-
JSH binding) written in Javascript for a browser platform. This enables a scenario where SMILE
processes running into rich web client developed in Javascript directly interact with SMILE
processes running on mobile devices, thus addressing requirement 7 (mobile to web integration).

2.4 SOAP gateway and PHP Gateway

Following requirements 6, we wanted to provide direct access to SOAP based Web Services from
mobile devices using SMILE. As shown in Figure 3, in order to fulfill this goal, we have added a
SOAP gateway in the SMILE architecture. The SOAP gateway is implemented as a Java servlet
running on a Tomcat servlet container. Using a SMILE-JS binding the servlet is able to receive a
SMILE request and to turn it into a SOAP request towards a SOAP Web Server. The servlet then
translate the response it into a SMILE-JS message and forwards it to the requesting SMILE process.
The messages transit using JSON format from the mobile device to the gateway and using XML
format from the gateway to SOAP Web Server. The choice of using WSDL also to specify SMILE
interfaces gives us the advantage that the WSDL definition of the existing web services can be
directly imported as interface definition for the application running in the mobile device. The
software engineering tools described below will automatically generate interface code for the
mobile device and for the SOAP gateway.

Figure 3 also shows a PHP Gateway. This element is used in order to ease the interaction of servers
written in PHP with SMILE (supporting requirement 7). Thanks to the PHP Gateway a server written in PHP can easily send and receive SMILE-JS messages, acting as it were a SMILE process.

2.5 Software Engineering aspects

A view of the SMILE middleware and of its relations with the automatic code generation tools is shown in Figure 4. The definition of the interfaces can be performed using the standard interface definition language for Web Services, i.e. the WSDL, or using a simplified syntax called IDLight, which represents an equivalent and compatible subset of WSDL constructs. A discussion on the advantages of using IDLight in addition to WSDL is out of the scope of this paper, but all details can be found in [3].

With reference to the numbering of elements reported in Figure 4, a tool (7.1) translates IDLight definitions into a canonical WSDL file. Starting from this latter, source code for the SMILE components is generated (7.2), taking into account peculiarities of the target platform for the SMILE component (Figure 4 describes the generation of Java Micro Edition CLDC/MIDP code). The code generation tools emits: i) data types related to the messages to be exchanged; ii) the skeleton of the server side of the component, with the callback methods that are called when incoming asynchronous messages or synchronous requests are received; iii) the stub methods used to send asynchronous messages and synchronous requests. The generated code is executed within a SMILE process (3), together with the SMILE process logic code added by the developer. The process interacts with the SMILE APIs libraries (4) to send and receive asynchronous messages and request/responses and for process life cycle management. The SMILE binding libraries (5) are then used for serialization and de-serialization of messages, therefore the data type part of the generated code interacts with these libraries. Finally the SMILE binding libraries generates and receive the data (2) on the wires according to the serialization format (e.g. JSON) and transport protocol (e.g. SIP or HTTP).

![Figure 4. SMILE middleware and automatic code generation tools](image)

As regards the SMILE binding libraries for the Java Micro Edition platform, it is worth discussing here some aspects related to the object serialization. A serialization mechanism is needed to transform the internal representation of an object into a stream of bytes that can be transported on the wire, interpreted and reconverted to a copy of the original object at destination. A specific
serialization format could be binary or text based. In Java Standard Edition, a binary serialization mechanism is built in and text based serialization (e.g. XML serialization) is provided in form of API. On the contrary, Java Micro Edition CLDC/MIDP does not support any automatic serialization, thus the application developer should implement her own serialization mechanism for each application typically by enumerating class fields and writing corresponding serialization code. This is due to the fact that this particular Java edition lacks the class introspection mechanisms needed to perform an automatic “exploration” of the internal structure of object classes. To relieve the developer from this tedious task, we integrated a general, seamless serialization mechanism into the SMILE binding libraries for Java Micro Edition (both for SMILE-JS and -JSH). This works by let the code generation tool (which runs on Java Standard Edition) to load the classes to be serialized, introspect them and automatically produce Java Micro Edition CLDC/MIDP compatible code using the above described enumeration approach to serialize and deserialize the corresponding objects, as if it would have been written by a developer.

The example depicted in Figure 4 is tailored to a SMILE process running on a Java Micro Edition based platform. One of the advantages of using WSDL as interface definition language is that starting from the same WSDL it is possible to generate interoperable SMILE peers running on different platforms and using different target platforms (an example include the aforementioned SMILE-JS binding written in Javascript targeting browser based applications). Moreover the tool can generate code to support the gateways scenarios illustrated in section 2.4 (SOAP gateway and PHP gateway). For example in the case of the SOAP gateway, the tool is able to generate the code that support the automatic conversion from XML serialization to JSON serialization.

3 MOVE – The example Mobile Application Framework

The smartphone market is characterized by the competition of different platforms (mobile operating systems). According to [7], the top 5 platforms account for 97% of worldwide smartphone sales as of third quarter 2010 and are: Symbian, Android, Apple iOS, Blackberry OS, Windows Mobile. The fragmentation is augmented by the several different versions of each operating system. A sign of the boom of mobile computing is the success of the so called “market places” where it is possible to download applications for a given platform. The development of “mobile applications” on proprietary platforms is in contrast with the web based approach for distributed applications which
is so successful in the “desktop” computing environment. In desktop computing, the web browsers provide a unifying platform and the “web application frameworks” takes care of providing a unified set of APIs to the developers, adapting to the peculiarity of the specific web browser where needed. This web based approach has not been so successful for mobile devices, for reasons that include: i) fragmentation of mobile browsers; ii) inherent difficulty in adapting the web based approach to the mobile interaction peculiarities (form factor, connectivity…); iii) need to access services and GUI features of the mobile devices that are not exposed to the mobile browsers. Whatever the reasons, “native” applications are developed for each platform, there is no portability from one platform to another and it is not simple to develop “cross-platform” services. Of course, this is not only due to technical reasons, as the different platforms are in competition and there is a business-driven trend to create non interoperable applications and services (with some platform owners more inclined to follow a “closed” approach and others pursuing a more “open” environment).

There exist several approaches and frameworks aimed to support cross-platform application development for mobile devices. The critical issue for any cross-platform solutions is how to access the specific features of each mobile devices (e.g. location based features, camera…) and how to adapt to the different set of features that are available. A quite complete list of cross-platform development solutions is listed in [8]. Most of the solutions are commercial and are not open source. The Java Micro Edition platform (Java ME) has been one of the most notable attempts to provide a cross-platform solution, but currently it is not natively supported on some important platforms (iOS, Android). In addition, the different mobile devices and platform that support Java ME offer different feature sets. Therefore, taken in itself Java ME does not represent a full solution.

The purpose of the MOVE (Mobile Open and Very Easy) framework is to provide an open source proof of concept of cross-platform service development using the SMILE communication middleware. In our approach we use Java ME as the base but add device specific adaptations to support the features missing in the Java ME APIs offered by a given platforms. We use some features of the open source J2ME polish Janus toolset for porting over different platforms, in particular to support Java ME code on Android.

The framework was initially developed in the context of the Simple Mobile Services project, where it was used to run a trial of a set of mobile application, involving 100 participants. As described in [3], the MOVE framework is composed by a MOVE client, a set of server side components and a community web portal [9] that allows to register to the system and to download the client for a specific platform. The interactions among MOVE clients and between MOVE clients and server side components are supported by the SMILe middleware. We have tested the MOVE client on Symbian, Android, BlackBerry OS, and Windows Mobile phones with JBed Java Virtual Machine.

The MOVE framework is an open framework to develop mobile services, offering built-in support for Location Based and Community Based services. The GUI framework is based on the Java ME porting of Thinlet [10], a very efficient GUI toolkit which uses an XML representation of the GUI elements. We have extended the original porting in order to adapt it to the features and the interaction models of different devices.

The complete list of features and services offered by the MOVE client is provided in [3]. We will only mention for Location Based services, outdoor and indoor maps and navigation, search a place, search a route, get GPS position, get approximate position, send position to friends, show friends position (both on the mobile client and on the web portal). For Community Based services, we mention view on-line/off-line friends, send messages to friends. An innovative solution is the use of MEMs “Mobile Electronic Memos” [11][3]. MEMs are electronic notes containing information about a location, a person, a service, a shop, a Web site... Users can automatically capture MEMs from the environment or from other services, store them for future use, share them with other users.
and send them as input to other services and applications. MEMs are stored in folders in the MOVE and can be managed both on the mobile client and on the web portal, which provides catalogues of MEMs. A set of screenshots from of the MOVE client over different platforms is reported in Figure 6.

Figure 6. Screenshots of MOVE client over different platforms
4 References


5 Notes to the reviewers

In order to keep the reference count close to the limit of 10 references, we have:

- provided a single link [3] where we store a set of technical reports that provide details of the different technology aspects in our proposal, rather then providing different links. The detailed references are:


- omitted to refer to other papers about SMILE and to the SMILE home page:

MjSIP

JSON

SIP Session Border Controller

Sip protocol

Blackberry push services
[24]“BlackBerry Push Service SDK, Version: 1.0.0.5, Feature and Technical Overview”

J2ME polish

Open IMS Core
[26]Open IMS Core's Homepage http://www.openimscore.org/

Finally for your convenience, we provide the list of current features of MOVE framework (available in [15])

MOVE Client

- Mobile Electronic Memo (MEM)

MEMs are electronic notes containing information about a location, a person, a service, a shop, a Web site... Users can automatically capture MEMs from the environment or from other services, store them for future use, share them with other users and send them as input to other services and applications.
Sending and receiving MEMs, sending a MEM to a group of users, sending MEMs to offline users
MEM categories: Memo, Position, Places to go, Restaurant, Monument, Hotel, Shops, People…
MEMs can be “tagged” so that new “categories” can be dynamically added
MEMs Inbox, MEMs favourite folder
MEM content: GPS coordinates, Address, opening times, web link, telephone number, small images, comments…
Some operations on MEMs: show on map, show multiple MEMs on map, “drive me 2” (evaluates and shows route from user’s current position to MEM position),
Making a phone call and opening a web site from the MEM content
Creating a MEM on the MOVE client
  - to take a picture and insert it in the MEM
  - automatic inserting the current position (e.g. from GPS)
Sending MEM to MEM server for permanent storage and sharing
Finding MEMs on the MEM server by category and/or by geographic position (work in progress)
Signing of MEMs using security keys stored in the SIM
  - Option to accept only signed MEMs
MEM expiration with auto-deletion (work in progress)
Adding comments to the MEM on the MOVE client (work in progress)

- Navigation application / location based features
  - Show maps, zoom in/out (same application for outdoor maps and for indoor maps)
  - Auto follow the current position (e.g. from GPS) on the map (“Follow me mode”)
  - Search a place by address
  - Search a business by category, address
  - Show a search result on the map
  - Show all search results on the map
  - Jump from one result to another (and back)
  - Convert a search result into a MEM
  - Search a route between two positions.
    - Start and ending position can be: current user position (e.g. from GPS), map position, address to search
  - Drive me to (find the route from current user position from a point in the map)
  - Show the route on the map as a set of “turning points” and related instruction
  - Send map center to a friend
  - Create a MEM using the map position as lat, long.
  - GPS position (internal GPS or external Bluetooth GPS receiver)
  - Send position to server (or keep it in the client)
  - Select visibility of position (private, friends, public)
  - Store the track log of your position on the server, select the visibility of the track log (private, friends, public)
  - Getting approximate position from cell ID (uses Open Cell ID)
  - Find my friends (show position of friends who has chosen to share their position)
    - Choose the time interval for the search of friends (last 30 minutes, last 3 hours, last 12 hours, always)
Choose the geographic span for the search (1 km, 10 km, 100 km, everywhere) (work in progress)
- Indoor navigation
  - Download maps of indoor / local areas (airports, fair shows, malls, amusement parks)
  - Get the list of nearest available indoor / local maps
  - Navigate into indoor / local maps
  - Find places and routes (shortest path) in the indoor / local maps
  - Indoor physical localization using zigbee

• Community
  - view on-line / off-line friends
  - send Memo MEMs to friends (like Short Message Services)
  - chat with friend (work in progress)
  - view friends’ contact details (e-mail, phone number)
  - call a friend
  - find friends’ position

• Information services
  - Meteo service
  - Athens Airport flight information (with notification of info for a subscribed flight)
  - Personalized Car Traffic Information

• Track logger
  - Record a position track log on the client
  - Current speed, average speed
  - View the track log on the map
  - Save the position track log as a MEM
  - Send the position track log to a friend

• Service discovery
  - Automatic discovery of new services
  - Services from trusted servers can be “pushed”

• 2D bar codes
  - Reading 2D barcodes (semacodes) on the MOVE client
  - Getting a MEM corresponding to the read semacode

• Interaction with SIM (only on Windows Mobile Phone and with special SIMs)
  - Signing and verification of messages using keys stored in the SIM
  - Storing and retrieving user preferences and data in the SIM

• Appearance
  - Changing font type and size
  - Change layout from list to grid
• **MEM server**
  o Catalogue of MEMs of different categories: Memo, Position, Places to go, restaurant, Monument, Hotel, Shops, People…
  o Handling of private and public MEMs
  o Search MEMs with given criteria
  o Create a MEM on the server
  o Send a MEM to a MOVE client or to a group of friends
  o Send a MEM hyperlink by e-mail, and open it on the receiving PC

• **Find my friends (the portal version)**
  o Find friends position on an interactive map
  o Choose the time interval for the search of friends (last 30 minutes, last 3 hours, last 12 hours, always)

• **Community (groups) portal**
  o Create groups
  o Register to groups
  o Invite people to groups

• **Authoring wizard**
  o Create personalized instances of a set of mobile services
    ▪ Mobile Poll service
    ▪ Mobile Headlines service
    ▪ Mobile Shop service

• **Indoor map wizard**
  o Create an indoor/local map by drawing “rooms” and “paths”
  o Upload map on the server, providing the geographical reference