open mobile IS

technology in motion

Optimize the development of your mobile applications with the Open Mobile IS framework

1 Introduction

This document is an introduction / tutorial on how to develop mobile applications with Open Mobile IS framework.
The presentation is divided into two main parts :

- The presentation of Open Mobile IS framework and its components
- A tutorial that shows you how to develop a Customer Relationship Management (CRM) application call MyCRM.

2 Open Mobile IS Framework

2.1 Presentation

Open mobile IS is an open source project (GNU LGPL license) that aims at providing all the necessary tools, API and documents enabling the development of effective nomad applications. Heart of the project, the java framework is divided into components providing all the needed functionalities. The Open mobile IS project is hosted by the ObjectWeb consortium.

Mobile applications are any type of applications used by mobile workers outside the company area. Those kinds of applications concern mostly sales forces, technicians, drivers and deliverymen...
Open mobile IS allows you to build complex and efficient applications like SFA (sales force automation), CRM (Customer relationship management) applications, running on any kind of devices, even the lowest CPU ones.

The key point is that nomad users need access to the company data anywhere, at anytime. To do so, they use a terminal (PDA, smartphone, laptop, Tablet PC...) offering an access to the company applications. Terminals are connected to the company from time to time to update their data. One of the main frameworks added value is to provide an efficient and secured synchronization mechanism between the terminal and the company information system.

Why is Open Mobile IS the best solution for the development of mobile application for your company? Beside the open source benefits, the main reasons are :

- Open Mobile IS is a complete and fully dedicated solution to nomad applications development. One can find many products, but each solves only one aspect of this type of application development. Creating a complete application that way means that you will have to integrate many of these products, regardless of compatibility or architecture consideration. From this point of view Open Mobile 's goal is to provide a framework offering all the needed components, , guarantee your application to be perfectly functional and ensure end users an amazing mobile experience.

- The framework contains codes and components, but not only. Our purpose is to propose patterns and best use cases reflecting our experience in enterprise mobile applications. Developing mobile applications is not only a matter of code, it is a business. Open Mobile IS helps you to avoid main traps or errors in this process.

- Open Mobile IS is not a new project. It's the integration of the Ubikis knowledge and technologies inside an open source project. Several customer applications have already been in production for years.

- Open Mobile IS is open source so you can participate in its development to ensure that it will fit in your needs. Only open source can guarantee that you'll keep the benefits of your investment.

- Open Mobile IS is developped in Java and is compatible with J2SE JVM above jdk 1.3.1 specification. It has been tested on several platforms like windows, Windows Mobile / Pocket PC with IBM J9, linux, embbeded Linux terminal like Archos PMA 400, Sharp Zaurus CL1000 or Nokia N800 PDA.

2.2 Main concepts

Four main concepts have led the development of Open Mobile IS. Those concepts have been deducted from our experience in mobile application. The main fact is that if you want to develop and deploy a mobile application you must answer the following constraints : security, accessibility, availability, evolution capabilities. If you miss one, your application won't be a success. Thanks to Open Mobile IS you can integrate and answer to all these constraints The idea behind our answer is to offer a way to plug your security constraint and solution inside the framework. So we provide security interface for all security needs in mobile applications like :

- Data protection inside the terminal.
- Data protection during transfer between the terminal and the information system.
- The information system access protection.

2.2.1 Accessibility

A mobile application has to be user friendly ..Final users are not always computer aware. In order to maximize the chance to see nomad application accepted, these applications must have an ergonomic adapted to users' needs.

2.2.2 Availability

Applications, and therefore the data, must be available everywhere, at any time. If the data are available only from time to time, users won't adopt the application.

2.2.3 Evolution capabilities

Terminals, software technologies evolve rapidly. Applications must be able to follow this evolution or they become useless within a few months.

2.3 Open Mobile IS constrains Answer

We answered to these constrains by developing a java framework providing the following components:

- An embedded object database optimized for low CPU or memory terminal called FODB.
- An embedded application server providing the servlet API, or a specific services API.
- A synchronization API dedicated to data and applications synchronization.
- A high level module providing basic functionalities needed to manage nomad applications.

2.3.1 Accessibility

In order to make application usability easier, the framework provides some ergonomic solutions to classic data management problems. These solutions have been tested on real applications for years and have been fully approved by nomad users.
As a base of all these solutions the framework provides a web interface for all applications. Internet applications are now used by billions of people and their GUI (graphic User Interface) paradigms have proven to be the most accessible. Beside this web integration, the framework provides some functionalities making the development of web application for mobile devices easier.

2.3.2 Availability
Availability is provided as follows :

- data synchronization, so users can have access to their data anywhere at anytime.

- high stability of the framework. The framework has been in production for several years with many different types of applications :
  - CRM (Customer relationship managment)
  - SFA (Sales force automation)
  - Maintenance apps...

2.3.3 Application evolution
Application evolutions are facilitated by :

- The Java developed framework compatible with Java JDK 1.1.
  Native functionalities like RAS use or PIM integration are mapped with a Java API and we provide a version for all supported terminals and OS (windows, Pocket PC and Linux).
  The framework provides an abstraction layer from the system that minimizes application development when changing the terminal. Some of our customers have developed applications which have been deployed on heterogeneous OS and terminals environment : Pocket PC 2000, 2002, 2003, 2003 SE... with no change in the application code. Only the version of the framework changed.

- Web interfaces dissociating the GUI from the business code.
  When the terminal changes (and so the screen), only the presentation layer has to be modified. Some of our customers' applications have been deployed on both windows PC and Pocket PC for PDA, with an identical business logic code.

3.1 Embedded Database

Open Mobile IS Embedded database is a simple object database optimized for low CPU and Low memory terminal. It provides a simple API for storing and retrieving objects from the database. The Open Mobile IS embedded database or Fast Object DB or FODB is perfectly suited for mobile and embedded applications. The object storage capability facilitates its use inside Java (avoiding mapping tool use and so on).
Simple query API (see from SODA API for more details) provides an easy way to manage objects inside the application.
The synchronisation engine is fully integrated with FODB for easer synchronisation development

3.2 Web server and Servlet API

All OpenMobile IS developments are based on the Internet paradigms. Applications are developed on a web server and all user interactions are made through a browser. In the case of embedded application, both server and browser lean on the same terminal.
Open Mobile IS embbeded server provides an implementation of the javax.servlet API. Thow, OpenMobile IS development can be deployed on any HTTP server supporting servlet API (Tomcat for example).

In order to facilitate embedded applications development, the project provides an embedded java WebServer (based on Acme.Serve.Serve single-class WebServer : http://www.acme.com/java/software/Acme.Serve.Serve.html) optimised for low cpu/memory devices.

3.3 Service API and template engine

With an aim at making OpenMobile IS development portability accross different achitectures easier, we added to the servlet API a new abstract layer named Service API. Service API offers a better integration to all functionalities provided by the OpenMobile IS platform (template service, module and profile managment, ...).

Service API is very similar to the servlet API, using HttpServletRequest and HttpServletResponse (from the javax.servlet API), for the treatment of the request and the response.

3.4 Synchronisation engine

To facilitate data synchronisation between information systems and terminals, we provide a multi purpose synchronisation engine. This engine uses the SyncML protocol for the communication layer. On the server side it can be integrated in any web server compliant with the servlet API (one servlet that handles all communications synchronisation).

We provide several types of synchronisation services to facilitate the development of your own.

The FODB database is fully integrated in it so that no development is required on the terminal side to synchronise a collection.

3.5 Module management

A module management API has been integrated to facilitate the management of applications. Each application can be divided into modules. Modules are associated to user's profile so each user can have his own application with his own module definition. Modules are synchronised on the terminal and updated during the synchronisation. We use OSGI standard as the base module management. The terminal module manager manages the module updates and its data too. If the data model has changed, the database is updated.

3.6 Architecture

The most common architecture is based on web architecture. GUI are developed in HTML. The Customer is a browser and a local web server generates the pages. A web synchronisation server is accessible on the network. It manages all data synchronisation using connections to the information system.

Based on this architecture, several other schemes can be realized. For example, the web server can be put on the network and depending on the functionalities, the user can get pages from the local web server or from a distant server. The synchronisation server, can be integrated inside the terminal. The synchronisation is made by the terminal that connects to the information system.

4 MyCRM development

4.1 Presentation

This chapter presents how to develop a mobile application with Open Mobile IS framework. The application will be a simple CRM (Customer Relationship management) named MyCRM. The presentation of a complete application allows to describe most of the Open Mobile IS functionalities.

The CRM functionalities developed will be :

- Customer account management. The user can create, modify, delete, search customer account.
- Each customer account can contain customer information, none or several contacts, one or several leads.
  - Customer can be created, modified, deleted.
  - Each customer can contain none or several reports.
  - Open leads are shown on the main page.

4.2 Development method

For all applications developed with Open Mobile Is, the same method can be used. The main steps are :

- Define the object model and the model life cycle.
- Develop a HTML static version of the application and validate it with the final users.
- Develop the terminal application.
- Develop the synchronisation.

It's very important to define an object model adapted to application needs and not to the existing data storage schemas. Most of the time data are not stored inside the information system the way it is used in the terminal. Keeping the information system schema will lead to complex developments and poor performance. Open Mobile IS with its Object Model synchronisation algorithms allows to dissociate the terminal object model and the main database data schema.

The main interests of the HTML method are :

- Validation of the application ergonomic
- Validation of the application functionalities

The two validations must be done with all the application actors.

Terminal application is developed before the synchronisation. Most of the time developers prefer to start with the synchronisation because they think its the most complex part. They think they have to validate it first and must know all the constraints it implies before developing the application. With Open Mobile IS there are less constraints made to the object model by the synchronisation. That is why it is preferable to develop the application before the synchronisation.

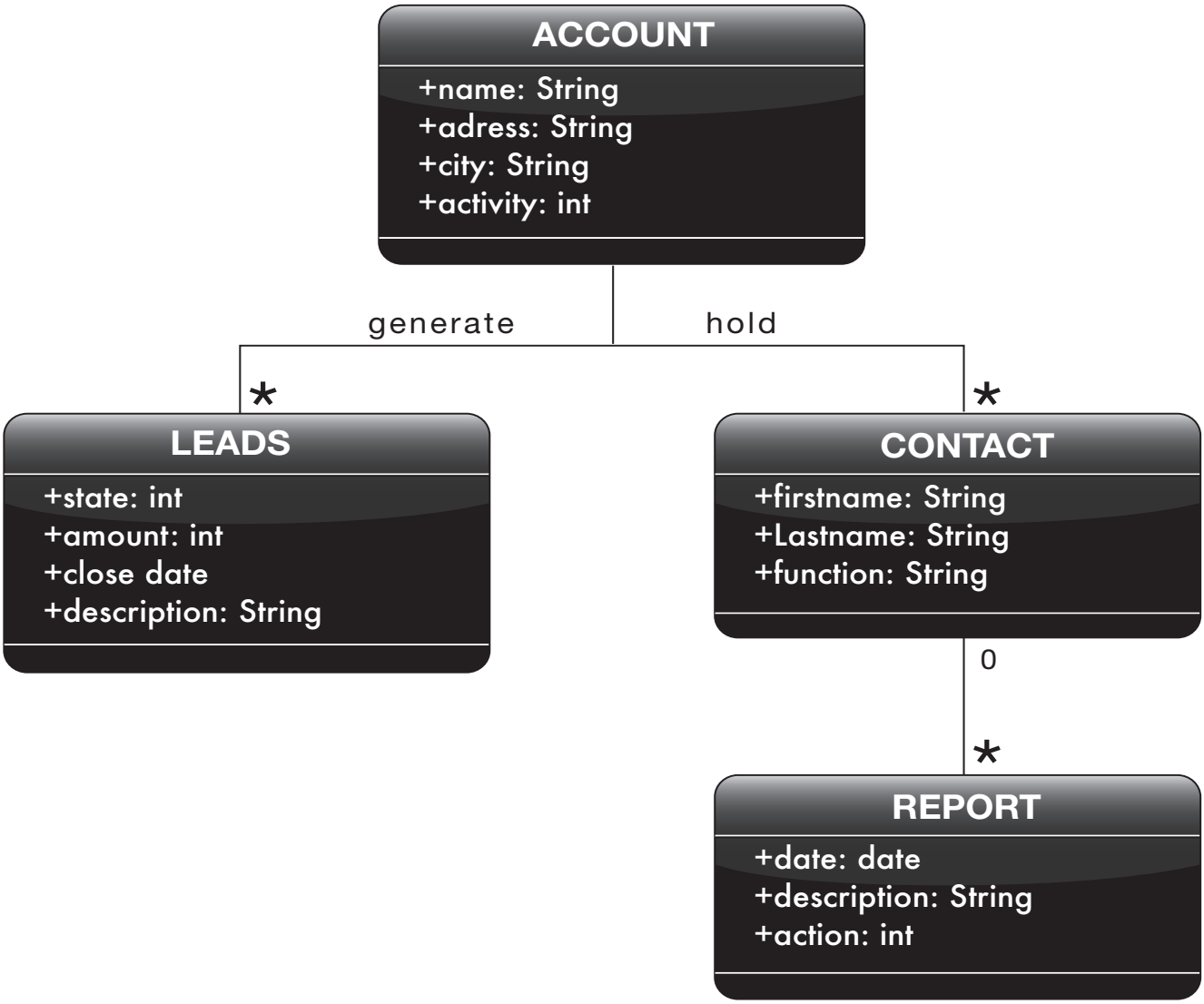
During the development of the application the object model can change. If you have already developed the synchronisation you will have to change it. When the application is finished, the object model is known and won't change. So the synchronisation can be done.

4.3 MyCRM object model

The goal of the object model definition is to organize objects into what I call "Atomic objects". "Atomic objects" are objects which are coherent when they are written to the database. In our example Atomic Objects can be Customer, Contacts, Leads, Reports. But it can be a Customer and its Contacts too. The size of the atomic object will depend on the application functionalities and the life cycle of each object. Objects that have the same functionalities can be treated as a whole.
So to define your object model and the atomic object you must analyse these criteria :

- Study the terminal needs to define your objects. In our example we have Customer, Contacts, Leads, Reports.
- Study the life cycle of each object defined. When/Where objects are created, modified, deleted.
- Define the need in object query. Only Customers are queried. One index will be created for the customer's name.
- Determine where and when objects are modified to identify possible synchronisation conflicts.
- Define objects volumes. Find the average, min and max number for each object stored in the terminal.

Base Object Model

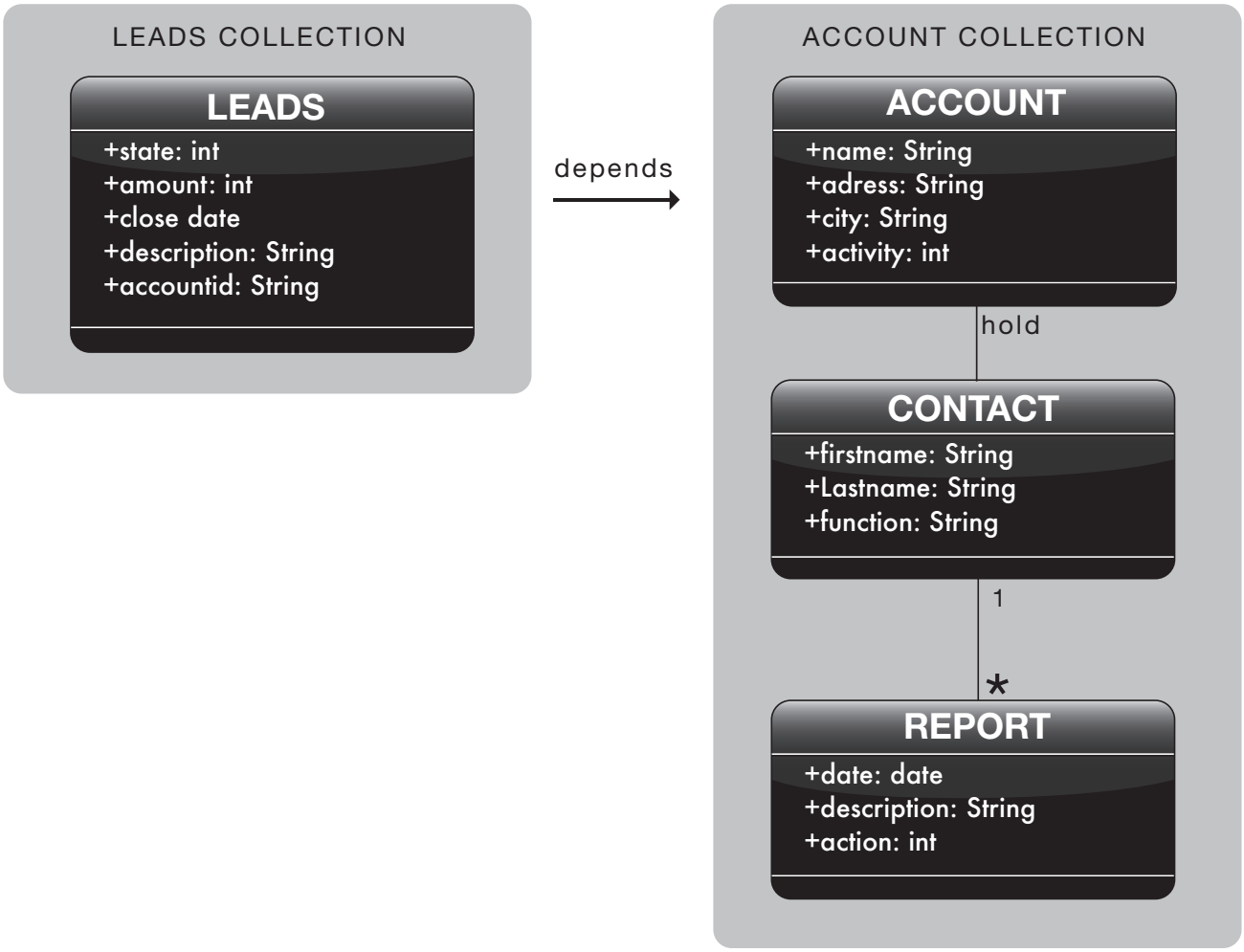


With these information, you can group and arrange objects together. For example in our case, Customer, contacts, reports have the same life cycle, same type of conflicts. There is no volume problems for any object. We suppose a commercial can't manage more than 1000 customers. Each customer has about 4 contacts and a contact is visited once a year. So they will have around 1000 customers, 4000 contacts and 12 000 reports (a commercial has the last three reports) . This is a normal volume for a mobile application and won't lead to synchronisation problems. Open Mobile IS is well suited to manage these types of data volume.
So I decide to create two atomic objects :

- one that contains Customer, Contacts and Reports
- one that contains leads.

There is no link between this object model and the database schema.
The synchronisation database connector will make the link.
On the terminal side, each atomic object is organized in collection for FODB storage.

■ Final Object Model



4.4 Object Model code creation

The only constraint on the object model development is that all data stored in atomic objects must be serializable. Atomic objects must be serializable too.
Example of code for the Account object :

```
public final class Account  implements Serializable{
  //add serial version to avoid serialization error if the version change
  // without class change.
  static final long serialVersionUID = 5521257935120563452L;

  //define the category for all activity label in LabelManager
  public static final String ACCOUNT_ACTIVITY_LABEL_CATEGORY = "1";
  private String id;
  private String name, address, city;
  private String activity;
  private Array contactList;

  public Account(String id) {
    super();
    this.id = id;
    contactList = new Array();
  }

  public Array getAllContacts() {
    return contactList;
  }

  public void addContact(Contact contact)   {
    if (contact == null) return;
    for (int i=0; i<contactList.size(); i++)    {
      Contact c = (Contact)contactList.get(i);
      if (c.getId().equals(contact.getId()))  {
        contactList.replace(i, contact);
        return;
      }
    }
    contactList.add(contact);
  } .....
```

In FODB database objects are stored in collections. To retrieve objects, indexes are used by the Query API. So we will create two collections, Account and Leads. In the Account collection two indexes are created to index the getId() method (Synchroniized collection must have an unique Id index like a primary key in database table for object management during synchronisation) and the getName() method index. The Leads collection will have two indexes getID() and getAccountID to get leads for an account (sort of foreign key).

In FODB there are two types of indexes , UniqueIndex and MultipleIndex. UniqueIndex index methods or attribute that has one object for each return value (sort of primary key or unique key). MultipleIndex index method or attribute that can have several objects for the return value. For example an account can have several leads. So the getAccountId() is indexed with a multiple index because in the collection there can be several Leads that have the same account.
To create a collection it's always the same algorithm :

- Test if the collection exists.
- If not, create it.
- Create collection index.
- Declare non persistent information like JournalLogRendered or SYnchroReturn Listener.

Collection is created using a collection descriptor that describes the characteristics of the collection.

■ Example of code for Leads collection.

```
private void initDB() {
  try {
        FastObjectDB db = FastObjectDBManager.getManager().getCurrent-
FODB();
    // test if the collection has already been created.
    //if not create it. Created collection are automatiquely
    // loaded when FODB is initialized.
    if (!db.isCollectionExist("leads")) {
      //SimpleOpenMISInit intialize the synchronized version of FODB.
      // all created collection are synchronized by default.
      // Synchro listener are automatiquely registered when the collection is
      created or loaded.
      // use the FODBCollectionDescriptor to define a collection that is not
      synchronized.
      FODBCollectionDescriptor coldes = new FODBCollectionDescriptor("leads",
Leads.class);
      db.createCollection(coldes);
      //add index to query the collection by id
      FODBStringIndexDescriptor IDDescriptor = new FODBStringIndexDescriptor("
ID", FODBIndexDescriptor.UNIQUE, "getId()", 12, 26);
      db.addIndex("leads", IDDescriptor);
      FODBStringIndexDescriptor AccountrDescriptor = new FODBStringIndexD
escriptor("ACCOUNT", FODBIndexDescriptor.MULTIPLE, "getIdaccount()", 12,
26);
      db.addIndex("leads", AccountrDescriptor);
      //only not closed leads are synchronized.
    }
    //register the log renderer.
    //logrenderer are use to show synchronization result for this collection.
      ((SynchroFastObjectDB) db).registerJournalLogRenderer("leads", "Leads
synchronization :");

  } catch (Throwable ex) {
    LogManager.traceError(0, "account Factory FODB ERROR : Unknown error
during creation");
    LogManager.traceError(0, ex);
  }
 }
```

- To query the collection, FODB provides an implementation of the Soda API. Example of Query code :

```
public Leads getLeads(String id) throws DatabaseException {
  Query q = FastObjectDBManager.getManager().getCurrentFODB().query();
  q.constrain(Leads.class);
  Query subq = q.descend("getId()");
  subq.constrain(id).equal();
  ObjectSet set = q.execute();
  if (set.hasNext()) {
    return (Leads) set.next();
  } else {
    return null;
  }
}

public Array getAllLeads() throws DatabaseException {
  Query q = FastObjectDBManager.getManager().getCurrentFODB().query();
  q.constrain(Leads.class);
  q.descend("getId()");
  ObjectSet set = q.execute();
  return ((FODBSodaObjectSet)set).getArrayFromSet();
}

 public Array getLeadsForAccount(String accountid) throws DatabaseExcep-
tion {
  Query q = FastObjectDBManager.getManager().getCurrentFODB().query();
  q.constrain(Leads.class);
  Query subq = q.descend("getIdaccount()");
  subq.constrain(accountid).equal();
  ObjectSet set = q.execute();
  return ((FODBSodaObjectSet)set).getArrayFromSet();
}
```

5 Services definition

5.1 Presentation

For the development of application functionnalities, Open Mobile IS provides a Service API that inherits from the Servlet API. This service API adds template integration and some GUI widgets useful for mobile application development. There are two types of services :
- the Service class for base service
- the TemplateService for Service that uses template.

To develop a service, you inherit from the base service you need (normal or template) and you implement the run method (like the servlet way). The getServiceURI method returns the URL of the service. The URI consists of the base service URL (commonly /services) and a specific part that identifies it in an unique way. All services must have a distinct URI.

For the template service, the run method returns the path to the template. The path is relative to the org.openmobileis.services.templatesDir property defined in the openmis.properties file. The template is found by adding the property and the return path. Template can be put inside the application jar. The path is indicated the same way by using the jar resource URI syntax (ex /mytemplate/template.htm, and the org.openmobileis.services.templatesDir is /templates.). All templates must end in .htm.

5.2 Navigation bar

The navigation bar is a GUI component that generates a navigation inside the HTML page. The navigation bar is an ergonomic component that facilitates application navigation.
To use the navigation bar component, a service must implement the NavigationBarService. This interface allows to define the text shown for the service inside the navigation bar (getNavigationBarLabel method), if the navigation is recursive (each time the service is called a new entry is added in the navigation bar or not), if it's a form a warning message can be shown when the user exits the form (displayFormExitMessage()).
Inside the template the tag ${navbarNOSY} must be added where the navigation bar must be shown. If displayFormExitMessage is activated, this script must be added :

```
function displayNavFormMessageExit(URL) {
        if(confirm('Do you really want to cancel the edition\nAll modified data will be lost.')) location.href = URL;
}
```

To define the look of the navigation bar, a class implementing NavigationBarDescription must be registered to the NavigationBarManager.

5.3 Edit service

Special services have been defined to simplify form development and user typed data management. The first one is the SimpleEdit service. It can be used when you have a form that doesn't call another form inside during the edition.

5.4 Service registration

There are two ways to register a service. The first one is to add its class to the listeservices.properties file. At start up this file is read and all services defined in it are created and registered. The second and the best way is to create a file that implements RubricLoader interface that will load all your application service. Each service is defined in the getServiceClassList method. The preLoadingInit is called before service loading and the postLoadingInit is called after. The rubric loader is registered as a service inside the listservices.properties file. When the bundle management is activated, the rubric loaded is declared inside the user's profile and loaded when the profile is loaded.

5.5 Synchronization

5.5.1Presentation

Open Mobile IS proposes a multi purpose synchronisation engine based on SyncML. It can be used to synchronize any type of data in any way. In this presentation I will focus on FODF database synchronisation.

5.5.2 FODB Synchronization

5.5.2.1 Presentation

The FODB synchronization engine, allows to synchronise objects between the information system and the terminal. Synchronization is done by synchronizing an object model. An object model consists of a collection of "Atomic Objects".
Each collection has a Synchro target on the server side that handles information system access. Any type of data can be synchronized : relational database, EJB, Web-Service ...
FODB synchronisation provides several synchronization modes :

- **Incremental synchronization** : Only modifications made on both sides are synchronized. Conflicts are solved on the server side.

- **Complete synchronization** : Terminal synchronization is incremental like in the previous mode. On the server side all server data are synchronized at a time . The terminal server collection file is generated on the server side and downloaded on the terminal. This synchronization allows to synchronize a lot of objects (several thousands) in a few seconds.

- **Complete pre-generated** : In this last mode the file is generated during the synchronization. In this mode the file is pre-generated. This allows to synchronize very huge or complex data in a few seconds.

By mixing all this types of modes in the synchronization, FODB engine allows to handle most of the synchronization problems from object conflicts to collection size.

5.5.2.2 The terminal side

When a collection is created on the terminal side, the collection descriptor indicates if it is synchronized or not. By default all collections are synchronized. To allow collection synchronisation, the SynchroFastObjectDBManager must be registered at the start of the application (see SimpleOpenMISInit class).
On the terminal side, when a collection is created to be synchronized, everything is done to track all collection modifications and to synchronize them. Nothing more has to be done on the terminal side.

5.5.2.3 The server side

On the server side, you must register a synchronization target for each collection. It is created by implementing the FODBSyncTarget. This interface defines all the methods needed to synchronize a collection :

- GetAllModifiedAtomicObjectSince : most important method and hardest to develop. Return all server side modifications since last synchronisation. Each modification is identified by a SynchroAtomicObject.

- GetUpdateMaxNbRow : returns the maximum number of modifications before forcing a complete synchronisation. If the number of modifications returned by the getAllModifiedAtomicObjectSince method is inferior or equal to this number the incremental synchronization is done. If it is superior, a complete synchroni sation is done. To avoid complete synchronisation set it to -1.

- GetCollectionName : returns the name of the collection synchronized.

- UpdateSynchroDB : called to update the server database with an object. The update can be a created or an updated type modification.

- DeleteCollectionObject : called to delete an object on the server side. The object Id is given.

- GetCollectionObjectWithId : returns the server side object with specified id.

- GetAllCollectionObject : returns all the objects in the collection. Called when a complete synchronization is started. Returns an array of collection objects.

- SetSendSynchroMetaData : On the terminal side it is possible to send context metadata for a synchronisation. The metadata objects added on the terminal side are given here.

For an example of implementation, see the MyCrmAccountSynchroTarget and MyCrmLeadsSynchroTarget in the MyCRM project. These classes are synchronizing a relational database. The object mapping is implemented by hand using SQL query. Other tools can be used such as hybernate.
The synchronisation target must be registered at the server startup. The simplest way is to add registering code in the OpenMISInit starter class (see the MyCrmServerOpenMISInit class). This class is declared in the openmis.properties file with the org.openmis.services.initclass property key.

5.5.2.4 Server side installation
Open Mobile IS server component is a servlet that you can load in any servlet engine. This is an example of Web.xml :

```
<web-app>
  <display-name>MyAppName</display-name>
  <description>My App Description</description>
  <servlet>
    <servlet-name>access</servlet-name>
      <servlet-class>org.openmobileis.services.servlet.OpenMISServlet</servlet-class>
    <init-param>
      <param-name>org.openmis.services.conffile</param-name>
      <param-value>/WEB-INF/conf/properties/openmis.properties</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>access</servlet-name>
    <url-pattern>/access/*</url-pattern>
  </servlet-mapping>
</web-app>
```

That maps the servlet to the access url.

6 Ubikis

Presentation

Ubikis is providing a wide range of products and services enhancing the enterprise information system user's experience, inside and outside the company network. Based on the open-source project Open Mobile IS, Ubikis technologies allow the development of business applications for nomad workers, on any kind of terminals (PDA, Smartphone, Desktop...) and on both connected and disconnected mode.

Ubikis is proposing services and support around Open Mobile IS. Besides this activity, Ubikis is developing modules to enhance the Open Mobile IS functionalities like PIM support and development tools.

For further information, please visit www.ubikis.com

7 Conclusion

Open Mobile IS is an open source Java framework dedicated to the development of enterprise mobile applications. It provides all the necessary components such as an embedded web server, an embedded database and a multipurpose synchronisation engine. All the developments use the web paradigms providing simpler development, better user usability and longer lifespan.

Open Mobile IS is not a new project and several projects have been in production for years. Released under the LGPL licence, you can start developing your own application right now.

One address : www.openmobileis.org