Journal of Universal Computer Science, vol. 18, no. 5 (2012), 704-727
submitted: 16/2/11, accepted: 28/2/12, appeared: 1/3/12 © J.UCS

Applying a Modular Framework to Develop Mobile Applications and Services

Mabel Vazquez-Briseno
(Faculty of Engineering, Architecture and Design, UABC, Ensenada, Mexico
mabel.vazquez@uabc.edu.mx)

Pierre Vincent
(MobKit, Lille, France
pierrevincent@mobkit.com)

Juan Ivan Nieto-Hipolito
(Faculty of Engineering, Architecture and Design, UABC, Ensenada, Mexico
jnieto@uabc.edu.mx)

Juan de Dios Sanchez-Lopez
(Faculty of Engineering, Architecture and Design, UABC, Ensenada, Mexico
jddios@uabc.edu.mx)

Abstract: The development of mobile applications and services has led to new challenges for software programmers. One of the main differences from standard PC programming is that the mobile environment is still limited in several aspects. Thus, there are many applications for mobile devices that share similarities, and implementing new applications may result in duplicated code within the same project or in similar projects. Moreover, current mobile development is still an error-prone task that may produce difficult-to-maintain applications. As a result, developers need new tools to help them to develop mobile software efficiently. This work focuses on a new framework for generating mobile applications and services, MobAppGen. The framework provides a set of components called modules that can be used to construct new applications using a Web interface. The architecture and functionalities of the framework are delineated, and several tests were conducted to evaluate the performance of the framework. For the tests, we created a number of projects using the available modules. In addition, a survey was conducted to estimate the acceptance and usefulness of the framework.

Keywords: Mobile Computing, Mobile Information Systems, Software, Mobile Devices, Multimedia
Categories: D.2.11, H.3.4, H.4.3, H.5.1

1 Introduction

Today it is clear that the most widely used device in the world is the mobile phone. The number of mobile cellular phone subscribers has surpassed the number of fixed lines and Internet users [ITU 2011] as shown in [Fig. 1]. However, according to [ITU 2010], in developed countries, the mobile market is reaching saturation levels while the developing world is increasing its share of mobile subscriptions from 53% of total mobile subscriptions at the end of 2005 to 73% at the end of 2010.

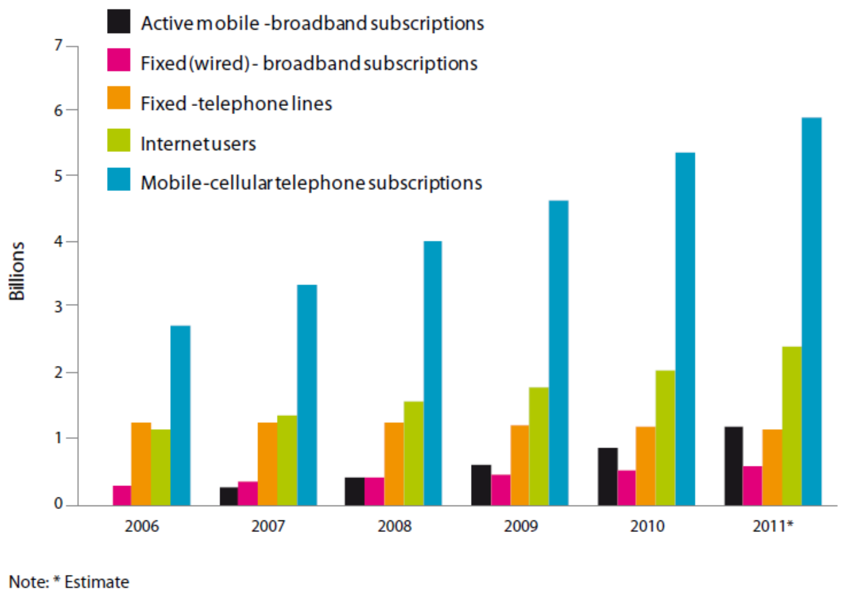


Figure 1: Information and communication technology levels [Source: ITU World Telecommunication/ ICT- Indicators Database.]

Phones are mostly voice-centric devices, but a wide range of mobile devices now exist on the market offering multiple services and functions. The basic capabilities, functions, portability, and cost of each of these devices vary greatly. As a result, they can be classified in many ways. In 2002, Prohm *et al.* [Prohm 2002] categorized mobile terminals according to their application programming interface (API) support as well as memory and processing power as: *basic phones*, *enhanced phones*, *smart phones*, and *wireless information devices*. The term *smartphone* is now used to characterize a mobile phone with special computer-enabled features. These features may include e-mail, Internet, web browsing and personal information management. Typically, the functionality of a smartphone can be further enhanced with add-on applications. The success of smartphones is strongly related to the development of mobile applications and services because mobile phones are now used for more than ordinary phone calls. The demand for mobile applications is also increasing with the growth of ubiquitous computing, and users requiring almost personalized services and applications. For this reason, even people with non-programming skills are beginning to develop their own applications. Mobile application development, however, is not an easy task because it includes several challenges for both experienced and novel programmers [Balagtas 2009]. Implementing such applications requires the use of a programming platform designed to run on mobile devices. Currently, there are several platforms available for mobile application programming; among the most popular are: iOS platform for devices such as iPhone and iPad, Android, which was developed by the Open Handset Alliance and Java Micro Edition (Java ME). The first one is a proprietary platform. Android is a free source platform that allows developing applications for devices running Android OS. On the other hand Java ME offers isolation from specific mobile device hardware and specific operating system software using a virtual machine abstraction. For this reason Java ME can be used in a range variety of devices including feature and low budged mobile phones, which are commonly used in developing countries. Java ME can provide a solution that fits most mobile devices; however, because the number of mobile applications that need to be developed and debugged has increased considerably, the work of developers has gained complexity. They must create almost personalized applications while considering that these applications must work on as many devices as possible. A main difference from standard PCs programming is that the mobile environment is still limited in several aspects. Thus, there are an increasing number of applications for mobile devices that share many similarities, and the implementation of new applications may result in duplicated code within the same project or in similar ones. The current development method is also an error-prone task that may produce difficult-to-maintain applications. As a result, developers need new tools to help them to address users' specific needs efficiently and quickly. Reusing the virtual machine abstractions to hide specific device details is important but no longer sufficient. With this in mind, we have developed MobAppGen, which stands for Mobile Applications Generator. MobAppGen is a framework that helps to create mobile applications relying on existing platforms and virtual machine abstraction on the mobile device but that offer high-level abstractions by hiding the details of a more complex underlying model. MobAppGen uses a Web-based modular approach to develop mobile applications.

The objective of this research is to describe the MobAppGen concept and to present some examples and studies conducted with the framework. The rest of the paper is organized as follows: In Section 2 we present an overview of mobile development. Section 3 presents the description of the designed framework. In section 4 we describe some application examples. Section 5 presents some works related with our approach. Finally, Sections 6 and 7 present the studies conducted with the framework. The first study evaluates the size of the applications created with MobAppGen. In the following study, we present the results related to the usability and acceptance of MobAppGen according to a survey of both experienced mobile programmers and inexperienced users.

2 Mobile Application and Services Development

There are many definitions of what constitutes a mobile application. In this work, we define it as follows: an application is *mobile* if it runs on a mobile device, namely a mobile phone, and is either always or occasionally connected to a network. A mobile application may include data storage, data processing or viewing or transmission to another application or server. In the same way, a mobile service is an electronic service that consists of three main components: a mobile application as a client, wireless networking and server implementation that provides the needed functionality or information (*Content*) to the user. In other words, application is a more technical term referring to the solution itself, whereas service is better associated with some 3rd parties (e.g., content provider or network based server) who provide some valueadded product to the end customer [Verkasalo 2006]. Currently, there are many kinds of services; most services are still only accessible by paying a service provider. We have found that the following are the most typical mobile services [Verkasalo 2006]:

- Real time communication (voice, multimedia)
- Web browsing (HTTP, voice-enabled)
- Location-based services
- Data synchronization (calendars, contacts, files)
- Data services (file transfer, e-mail download)
- Streaming media (music, video, events)
- Peer-to-peer communications
- mCommerce (micro-payments, finance)

In general, mobile services and client-server applications consist of data services that require the Hypertext Transfer Protocol (HTTP) to connect to a Web server and often to a remote database. However, there are also other services, such as multimedia streaming and real-time communications, which require other types of connection to retrieve information. Mobile multimedia streaming services are receiving considerable interest because they are now technically applicable over third-generation wireless networks. However, given that mobile terminals have a wide range capabilities it is not probable that all of them will be able to support all proprietary Internet streaming formats and protocols. A common standardized format is then required to guarantee the creation of compatible solutions. The Third Generation Partnership Project (3GPP) has standardized streaming services and specifies both protocols and codecs [3GPP 2006].The 3GPP defines the Real-time Streaming Protocol (RTSP) [Schulzrinne 2003a] and Session Description Protocol (SDP) [Handley 1998] for session setup and control and the Real-time Transfer Protocol (RTP) [Schulzrinne 2003b] for transporting real-time media such as video, speech, and audio.

Mobile technology is also an opportunity to help preventing health problems worldwide. *Mobile health* or *mHealth* can be defined as using mobile communications and devices, including mobile phones, for health services and information [Vital, 2009]. mHealth was defined in 2004 as “Mobile computing, medical sensors and communication technologies for healthcare” [2]. Since then according to [Vital, 2009] several mHealth applications have been developed to support a variety of services and each of these systems has required the implementation of mobile phone-based applications from scratch.

Mobile applications can be divided into web-based and framework-based applications [Beji 2008].Web-based applications are related to the use of a mobile browser while mobile framework-based applications are applications that run on a mobile programming platform over the mobile operating system. This second type of mobile application can be either network applications implementing a service as interfaces to these services or as stand-alone applications in the handset that require no network/service connectivity. Developing this kind of application is not simple, and it incorporates a new set of design challenges. One of the challenges is that current mobile devices have diverse hardware, operating systems and supported network technologies. Many handsets are resource-tight, while others are equipped with more CPU and modern features. The diversity is a challenge for both design and final porting. In addition, this heterogeneity causes problems to software developers who must fit applications to as many devices as possible and conform their application to multiple older versions of mobile device software. Device manufacturers and other actors have tried to simplify this work through standardization and by providing open platforms for easier service creation and application interoperability. Among these technologies, Java ME is currently one of the most popular, especially in developing countries such as Mexico where devices supporting iOS or Android platform are still too expensive for most users.

Java ME is a smaller version of Java 2 Standard Edition targeted towards small devices. Java ME is standardized through the Java Community Process (JCP), which consists of participants from the industry, and specified in Java Specification Request (JSR).The architecture of Java ME consists of configurations and profiles [Knudsen 2003].Configurations define the minimum set of Java core classes required by the virtual machine to work. Profiles add additional functionalities for specific devices. The configuration that defines small mobile devices is called the Connected Limited Device Configuration (CLDC). The Mobile Information Device Profile (MIDP) is the profile corresponding to mobile phones. The applications developed using the MIDP profile are called MIDlets. A Java ME application suite is stored on a JAR (Java ARchive) file and described using a JAD (Java Application Descriptor) file. Unfortunately, despite its simple programming model, Java ME lacks support for modularizing the implementation of mobile applications, which could help developers address users’ needs for new customized applications and services efficiently.

3 MobAppGen Description

MobAppGen is a web-based framework that was conceived by taking into account that mobile applications share many common non-functional features, such as screen management, data persistence and network communication. In this way, MobAppGen is a generic framework that allows mobile applications to be constructed from reusable software components. These components consist of a set of modules; each module implements a determined basic task and can be reused to construct different applications. Mobile applications generated with MobAppGen can consist of client-only and/or client-server applications. In both cases, the client-side application consists of Java MIDP applications or MIDlets. These applications are targeted to CLDC/MIDP devices like mobile phones and PDAs with networking capabilities.

The framework generates the mobile applications using a set of predefined modules that were designed to target the most common tasks when developing mobile data services and applications. MobAppGen also generates the server side when it is required. In this case, it provides the files that must be installed on the server to interact with the client-side application. Most modules only require the use of the HyperText Transfer Protocol (HTTP) to interact with a remote server to access data stored in a file or database. In addition, the framework includes a streaming module that requires communication with a remote server using the Real Time Streaming Protocol (RTSP) to access a media file. Both servers are included in the framework. The first one is used to manage HTTP connections and to retrieve information using a set of predefined PHP scripts. If the server requires database management, it opens a connection to a MySQL database and sends the appropriate query. The second server is an RTSP server that was developed using Java. However, the streaming client application is able to interact with any other server that complies with the 3GPP streaming standards, like QuickTime Streaming Server and Darwin Streaming Server.

The smallest applications created with MobAppGen consist of a single MIDlet; however, most applications require a set of MIDlets to implement more complex tasks; in this case, the application consists of a MIDlet suite, which share information, classes and resources. [Fig. 2] depicts the types of applications that can be constructed with MobAppGen.

The modules used to construct the applications determine the required elements. The simplest application corresponds to diagram 1 in [Fig. 2]. In this case, the generated application consists of a client-only application that does not require a server side. However, a client-server application may require communication with the HTTP server, with the RTSP server or with both servers.

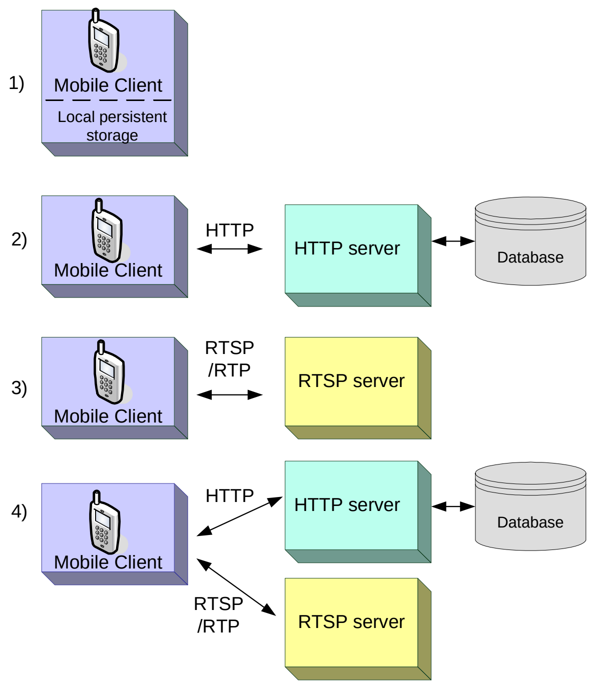


Figure 2: Mobile applications and services generated with MobAppGen

[Fig. 3] depicts the main framework components. *Developer info* corresponds to the information that the programmer provides to the framework through the web-based tool to create an application. The framework has been designed to be used by beginners or advanced programmers. Beginners are able to create complex applications without the need for modification when the application is complete.

More experienced programmers are able to adapt the source classes to their projects if required with the advantage of having some basic tasks already implemented in a functional state.
The main functions realized by MobAppGen are summarized as follows:

- Provides a friendly environment to capture the developer information.
- Creates proprietary code based on the developer information.
- Generates source Java ME classes for clients.
- Supplies the PHP scripts for the server side according to the project requirements.
- Includes a RTSP server for streaming applications.
- Compiles and Pre-verifies source files.
- Generates JAR and JAD files for clients.
- Prepares all files to be downloaded by the user in a light format.

To simplify future software extension and framework portability, open-source and freely available software tools were used to develop each component. As mentioned before, the servers are written in Java or PHP. The web-based tool was built using XHTML and PHP. The MobAppGen Class Creator was also implemented using Java and Velocity templates. The MobAppGen Application Builder uses Apache ANT.

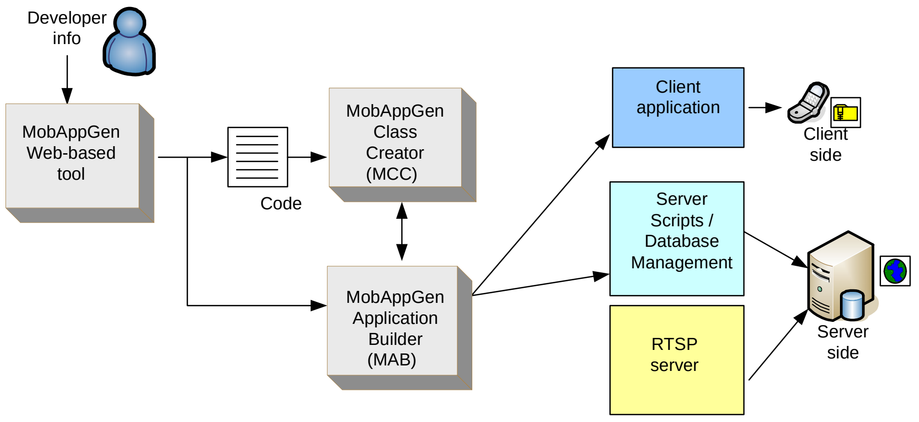
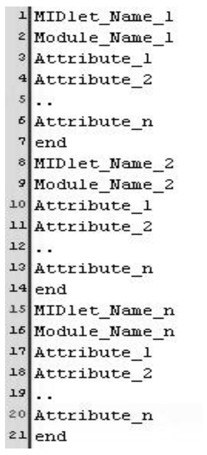

Developer info
MobAppGen Web-based tool
Code
MobAppGen Class Creator (MCC)
MobAppGen Application Builder (MAB)
Client application
Client side
Server Scripts / Database Management
RTSP server
Server side

Figure 3: MobAppGen Framework Architecture

The main MobAppGen framework components are described in the following sections.

3.1 MobAppGen Web-based Tool

The MobAppGen web-based tool is the framework's user interface. It allows the developer to choose the required modules and corresponding attributes to construct a mobile application called a *project* in the framework. With this information, the tool generates the proprietary code that is used as input for the MobAppGen Class Creator. The code syntax is depicted in [Fig. 4].



```
MIDlet_Name_1
Module_Name_1
Attribute_1
Attribute_2
..
Attribute_n
end
MIDlet_Name_2
Module_Name_2
Attribute_1
Attribute_2
..
Attribute_n
end
MIDlet_Name_n
Module_Name_n
Attribute_1
Attribute_2
..
Attribute_n
end
```

Figure 4: MobAppGen code syntax

The code contains information about the modules used to construct the project as well as their attributes. The advantage of the web-based tool is that it provides a friendly environment for the developer who chooses the modules and attributes with graphical interfaces like buttons, multiple choice menus and text field inputs. Each project consists of one or several applications. The developer must add a module for each application in the project. This information is used to create the proprietary code that is used as input in the MobAppGen Class Creator to generate the applications' classes.

In the code each application is identified by using a unique name (*MIDlet-Name*), followed by the module's name (*Module_Name*) that will be used to construct the application, depending on the module a set of attributes are included in the code. The modules and their corresponding attributes are described in section 3.3.

Another important advantage of the web-based tool is that MobAppGen can be used locally on a developer desktop or remotely through the Internet. In both cases, multiple users can access the framework at the same time. In MobAppGen, there is no limit on the number of modules that the developer may add to the project; however, some mobile devices limit the size of applications that can be installed.

3.2 MobAppGen Class Creator - MCC

The MobAppGen Class Creator (MCC) is written in Java. [Fig. 5] depicts the MCC block diagram. The MCC interprets the code previously generated with the Web-based tool and creates all the required client-side source classes using Velocity templates. Velocity is part of the Apache Jakarta project and is a Java-based template engine that provides a simple but powerful template language to reference objects defined in Java code. Each client-side module consists of a class descriptor and an attribute descriptor. There is an attribute descriptor for each type of module that contains a description of the attributes that correspond to that determined module.

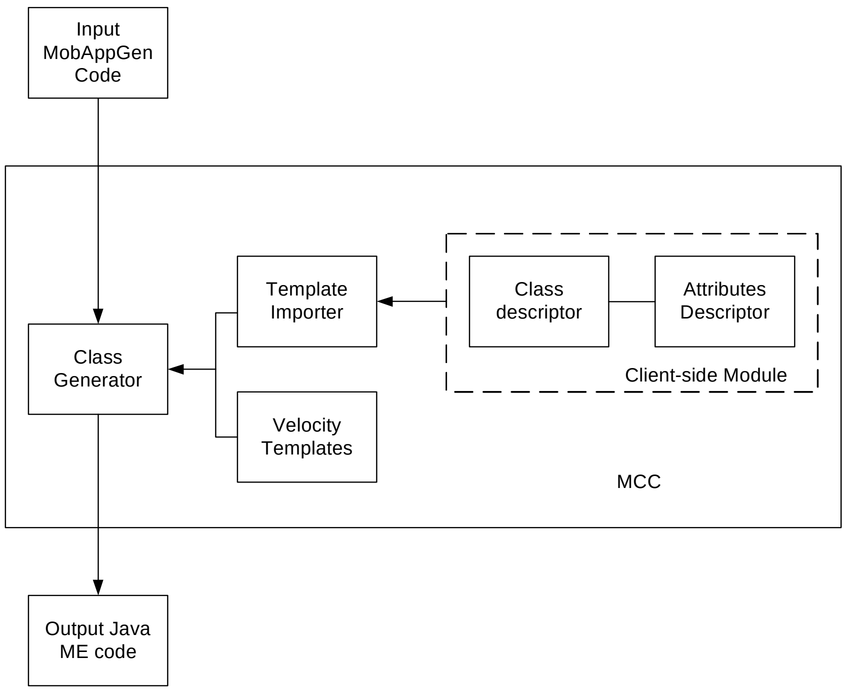


Figure 5: MobAppGen Class Creator (MCC) - block diagram

The MCC merges the modules' descriptions with the predefined Velocity templates included in the framework. This operation creates all the required classes for a specific application. The output of the MCC is used by the MobAppGen Application Builder, which compiles, packages and prepares the files to be used by the final user or developer. [Fig. 6] depicts the MCC general diagram containing its principal classes. ClassGenerator is the main class; it takes the code file generated with the web-based tool as an argument to generate the client-side source classes. The code file is processed by reading each line at a time. The first line corresponds to the MIDlet or application name. The second line corresponds to the module name; it determines the template that will be used to generate the application. The following lines correspond to the module's attributes; an ArrayList is filled with these attributes until the word *end* appears in the code as shown previously in [Section 3.1]. If the project consists of a MIDlet suite, the following MIDlets are described in the next lines with the same syntax as explained before. The code is processed until the end of the file is found. The three application features, *application name, module type* and *attributes array*, are taken by the TemplateImporter as arguments. It determines the classes that have to be included in TemplateControl and fills out the corresponding ClassDescriptor and AttributeDescriptor features for this specific module. This process is repeated for each application described in the code file. There is a single ClassDescriptor defined with different AttributeDescriptor arrays depending on the module type. However, there is an *AttributeDescriptor Set* (ADS) of classes; a different class for each module. The last step is performed by the ClassGenerator, which merges the attributes and class description of each module defined in TemplateControl with the corresponding templates using Velocity context. In this way, all the Java ME classes for this project are created. Some classes are shared by several modules, and in this case, they are added only once into the final project.

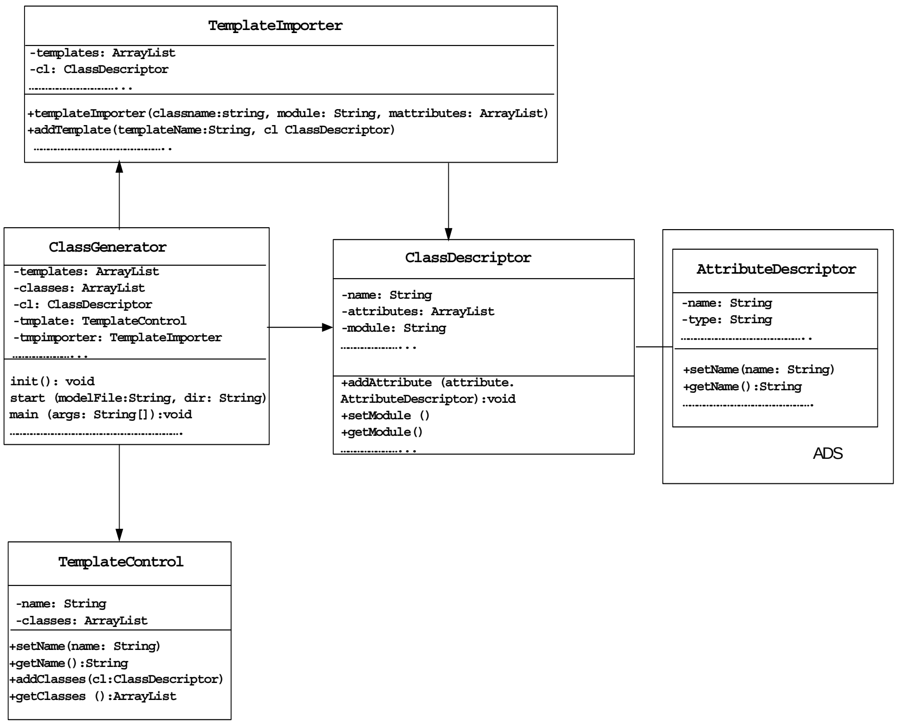


Figure 6: MCC general class diagram

3.3 MobAppGen Modules

MobAppGen aims to represent a modular and extensible framework. It consists of several modules with specific attributes. The available modules in the framework are divided according to the applications that can be placed into two categories: client-only and client-server, as depicted in [Fig. 7]. Each module was conceived to implement a basic task and is adaptable to specific jobs by defining its attributes. Thus, a project generated with our framework could consist of two similar modules doing different jobs based on their attributes but sharing common tasks. Future framework extensions are based on the possibility of adding more modules to implement different tasks that could interact with the existing ones. Because one of the goals of the framework is code reuse, all class templates are predefined to be shared by client-side applications doing similar tasks.

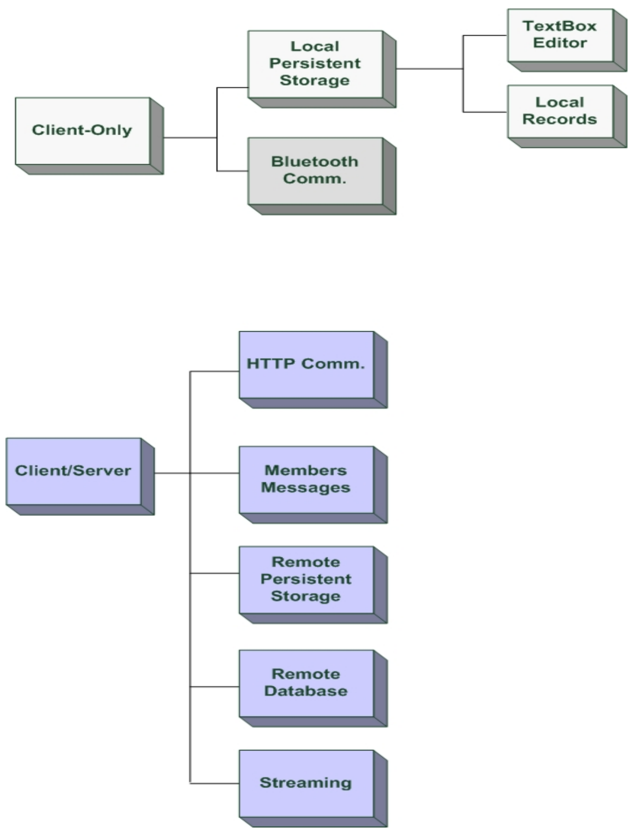


Figure 7: Framework modules

3.3.1 Client-only Modules

Client only applications can be used to store data on the device or to communicate with another device using Bluetooth communication. The main task of the first client-only modules is to store and display data on the mobile device. This data can be retrieved later and can be shared by several client-side applications in the same project. In MIDP, the package Record Management System (RMS) gives the programmer the ability to create groupings of registers called *record stores.* The package javax.microedition.rms is used to handle data storage when little memory is available in the cell phones. MobAppGen includes two client-only modules that provide persistent storage with two types of user interfaces. These modules maintain a series of records as a record store. It can be considered to be a small database on the mobile device.

- **TextBox Editor**: In Java ME, a *screen* is the base for all classes that represent generalized user interfaces. The simplest type of screen is the *TextBox*; it allows the user to enter a multi-line string, and it can be useful for many applications. This module provides the functions of a text editor in a mobile device using a *TextBox* display object. The module adds the main functions required to manage information and provides local data persistent storage. The incorporated functions allow the client to add, edit, cut and copy text like any text editor does. The attributes for this module include the *`records order`* which is used to show de records using the RMS package. The module includes also the attributes to describe the TextBox object wich are: `Name, Lenght` and `Constraints,` corresponding to Java ME characteristics of a TextBox.

- **Local Records:** This module generates applications that offer local data persistent storage on a record store but use different input objects. The developer defines the objects and attributes that are required to capture the information. The objects can be *TextField*, which represents an editable string, and *DateField*, which provides a mechanism by which users can enter dates, times, or both depending on the creation mode. Both objects are part of a collection of user-interface controls provided in Java ME called *items* that are displayed on a screen called *Form.*

In addition a **Bluetooth Communication** module can be used to create applications that retrieve information from devices using Bluetooth technology, including medical sensors. This is done using the Bluetooth API JSR-82. The developer can configure the *read rate* and appropriate parameters depending on the device to be used.

3.3.2 Client-Server Modules

Client-server modules generate a variety of applications with the common need to interact with a server. This communication is established with the HTTP server using HTTP connections and with the RTSP server using socket or datagram connections. As explained before, one single application can be constructed to communicate with only one or with both servers. Some client-server applications are designed to provide service only to authenticated users or *Members*. This measure guarantees some data security. In this case, accessing the HTTP server from a mobile device requires authentication. Member information, such as name, creation date, login, and password, among others fields, is recorded on a MySQL database within the server and can be managed using PHP scripts.

To add a new client-server module in a project the developer must first provide the server's configuration information (e.g., IP address and directories), which is referred to as server's configuration.

- **HTTP Communication**: This module creates applications that allow the mobile device to act as a web client. The web client accesses an HTTP server to retrieve some information and displays it. The client is able to connect to a determined URL that may include a PHP script and a parameter. This service may or may not be authenticated. The attributes in this module include `The HTTP method` that c that will be used to retrieve data from the server as well as server's configuration to access the remote server and corresponding script.

- **Member Messages**: This module creates an application that allows Members to send and receive messages to/from other *Member* using HTTP communication. Because each declared Member has a dedicated safeguarded space to store data, this space is also used to maintain an inbox. All messages from other Members are stored in the inbox, and the Member can access them from any mobile device. After the inbox is recovered, it is maintained locally using the MIDP RMS system to allow the Member to read her/his messages any time without connecting to the server. This module generates the classes required to send and receive messages as well as to perform operations in the inbox.

- **Remote Persistent Storage:** This module creates a client-server application that requires user authentication because it deals with data persistent storage on the remote server. Information is stored as a simple text file without any format. Information is retrieved by the mobile device using a single HTTP connection.

- **Remote Database:** This module is similar to the Local Records module, but in this case, the records are maintained on a remote database in the HTTP server. This setup is useful because the database can be accessed from not only the mobile device but also any desktop PC. Information is stored on a MySQL server, and users can retrieve it using SQL queries. From the mobile device, only authenticated Members have access to the remote database. The module attributes let the developer determine which database field corresponds to each user-interface object. This object is used to capture and display information in the mobile application. The `Server's configuration` attribute includes the necessary information to access the remote database. Each record in this database will be mapped to a `TextField` or `DateField` and managed as a record store once it is displayed on the mobile device. The objects description includes an attribute defined as `DB_Field.` It corresponds to the field in the database that will be related to this object. They also contain the Boolean attribute `inTitle` that indicates if this field will be shown on the record's title list when accessing the records in the application. `Records_order` is used to determine the order in which records will be shown in the list; it can be by modification date or alphabetically by any of the fields. The `Database_name` attribute corresponds to the name of the database on the server that will be used in this application.

- **Streaming:** This module helps the developer create applications that implement media streaming on the mobile phone. These applications interact with the RTSP server provided with the framework. In the simplest case, a mobile client communicates with the RTSP server to get a particular media file; only the mobile client and the RTSP server are needed in this case. However, it is also possible to add communication with an HTTP server that stores and administrates media files received from the clients. HTTP connection is then required to send files from the mobile phone to the server. The HTTP server can also be used to provide security allowing access only to the authorized Members. At the moment only AMR audio files can be read. The file is transmitted using RTP datagrams as defined in [Sjobem 2006]. The player application uses a set of buffers to read and store data while playing them using the Mobile Media API (MMAPI) [MMAPI 2002] provided with the Java Me platform.

4 MobAppGen Application Examples

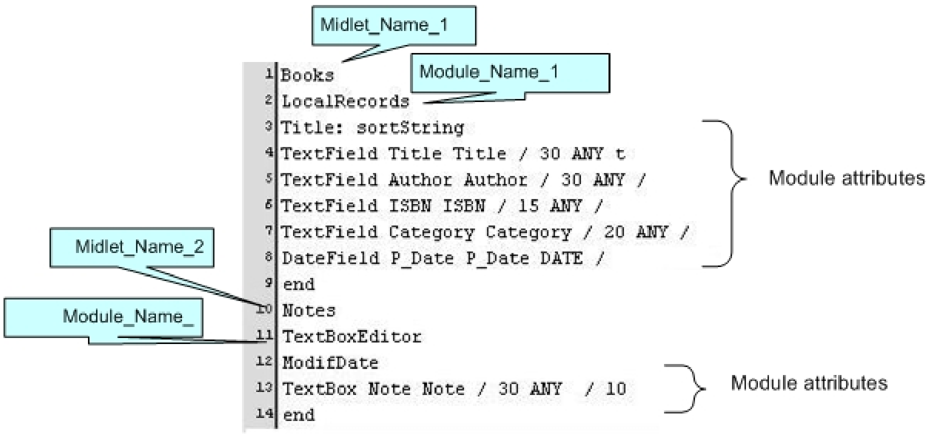
As explained previously, the MobAppGen framework can generate mobile applications based on projects constructed using a set of modules and their attributes. Some examples are presented here.

4.1 The Personal Library Application

In this scenario, the user desires an application to organize her/his personal library information using a mobile device. The application will be used to store and edit a set of records containing information about the books. The user wants to include the following fields for each book in the library: title, author, ISBN, category, and publication date. The application does not require remote storage; thus, the records will be maintained only on the mobile device memory. The user also wishes to be able to add annotations about the books that she/he has read or intends to buy as well as other important notes.

The project created for this case is called “Personal Library”; it consists of two applications. The appropriate module for generating the main application corresponds to the Local Records module. The corresponding module’s attributes match the listed information about the books. The developer needs to add another application generated with the TextBox Editor module to allow the user to maintain and edit the library’s notes. [Fig. 8] shows the code generated using the framework after entering the modules’ information.

The first application referred to as Books is the one generated with the Local Records module. The record store in this application is ordered and listed based on the book’s title of each record. The *t* symbol at the end of line 4 in [Fig. 9] indicates that this field is part of the record’s title. The developer could have chosen another field or several fields. The next lines represent the attributes of the user input objects. The slash symbol (/) indicates that the corresponding attribute has been left blank. The second application in the project is denoted as Notes; it is generated using the TextBox Editor module with its corresponding attributes as shown in the figure. In this application, the records are organized based on the modification date field.



```
Books
LocalRecords
Title: sortString
TextField Title Title / 30 ANY t
TextField Author Author / 30 ANY /
TextField ISBN ISBN / 15 ANY /
TextField Category Category / 20 ANY /
DateField P_Date P_Date DATE /
end
Notes
TextBoxEditor
ModifDate
TextBox Note Note / 30 ANY  / 10
end
```

Figure 8: Code generated with the Web-based tool for the "Personal Library" project

The code illustrated in [Fig. 8] is used as input in the MCC to create the corresponding client-side application. [Fig. 9] depicts the block diagram for the "Personal Library" project.

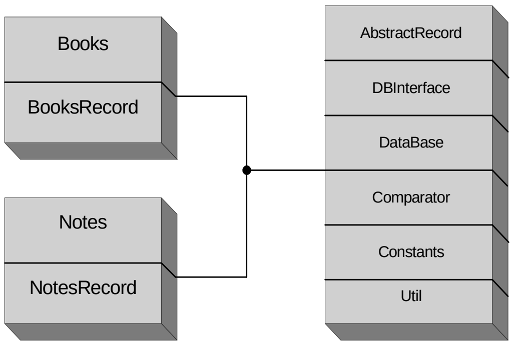


Figure 9: "Personal Library" project block diagram

The source classes shown in [Fig. 9] are generated using the corresponding framework's templates. The project contains a total of 10 source classes. It can be observed that both applications share the classes that implement the record store management, which significantly reduces the size of the project. Otherwise, each application would have to contain the methods required to implement these functions.

4.2 mHealth project

MobAppGen is generic framework intended to develop a wide range of mobile applications and services, after a number of tests it has shown to be particularly useful for developing mHealth applications to be used in developing countries for several purposes including patient monitoring using the Bluetooth module, as well as prescription and medication reminders.

In this scenario the user is a patient with a chronic disease who requires taking medication permanently. The project consists of an m-Health system that is used to facilitate that the users retrieves information about drugs from a remote database installed on the doctor's office as depicted in [Fig. 10].

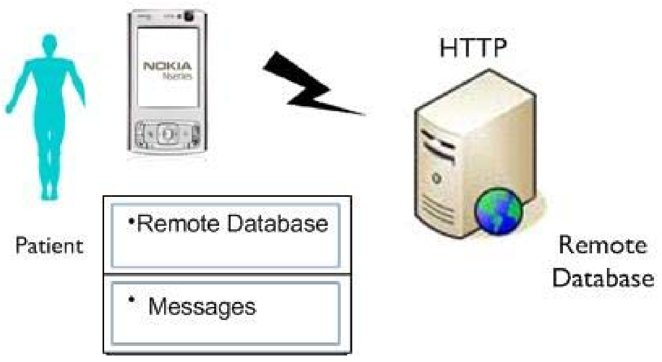


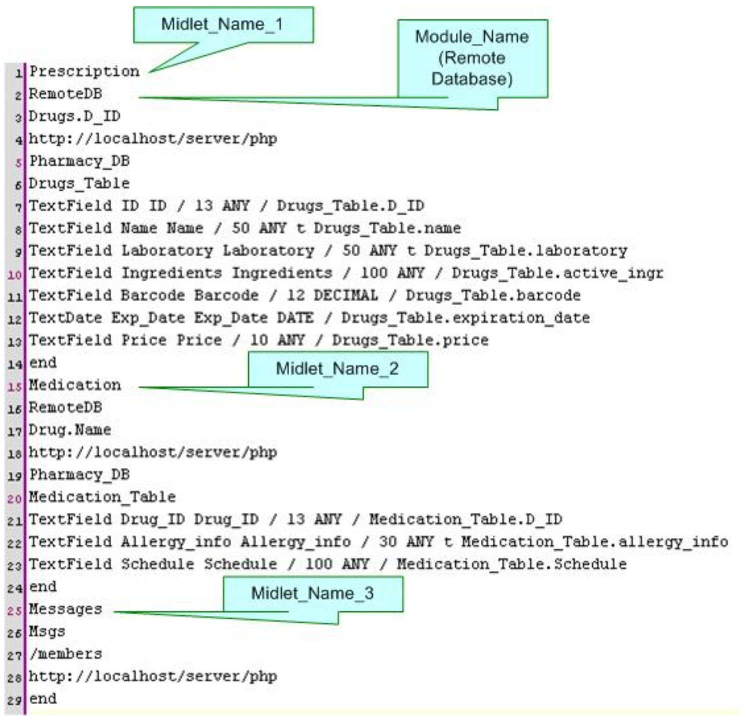
Figure 10: Medication m-Health system architecture

The information retrieved from the remote database includes: medication schedule, allergies warning and active ingredients as well as other drug's information. Data is stored on a remote database that doctor updates each time he or she prescribes a drug to a patient. Doctors can also send notifications to patients about changes in prescriptions or other information concerning medication. The mobile application in this system consists of three MIDlets developed using two MobAppGen modules:

- Remote Database: To retrieve drug's information stored in the remote database. Information is retrieved from two tables; the first MIDlet reads drugs general information from *Drugs_Table*. The second one retrieves prescription information from *Medication_Table*.
- Member Messages: To receive messages from the Doctor using HTTP communication.

[Fig 11] shows the code generated with MobAppGen Web-based tool comprising the modules information. The framework generates the Java Me application to be installed on the mobile device.

The first two applications in the project are generated with the Remote Database module. The database fields corresponding to the tables *Drugs_Table* and *Medication_Table* are matched to the corresponding display objects (TextFields) to capture, edit and present the data. The module Member Messages is also added to send and receive messages to patients that have to be registered on the Member's database. For this example, the server’s URL is referred to as *http://localhost/server/php*.



```
Prescription
RemoteDB
Drugs.D_ID
http://localhost/server/php
Pharmacy_DB
Drugs_Table
TextField ID ID / 13 ANY / Drugs_Table.D_ID
TextField Name Name / 50 ANY t Drugs_Table.name
TextField Laboratory Laboratory / 50 ANY t Drugs_Table.laboratory
TextField Ingredients Ingredients / 100 ANY / Drugs_Table.active_ingr
TextField Barcode Barcode / 12 DECIMAL / Drugs_Table.barcode
TextDate Exp_Date Exp_Date DATE / Drugs_Table.expiration_date
TextField Price Price / 10 ANY / Drugs_Table.price
end
Medication
RemoteDB
Drug.Name
http://localhost/server/php
Pharmacy_DB
Medication_Table
TextField Drug_ID Drug_ID / 13 ANY / Medication_Table.D_ID
TextField Allergy_info Allergy_info / 30 ANY t Medication_Table.allergy_info
TextField Schedule Schedule / 100 ANY / Medication_Table.Schedule
end
Messages
Msgs
/members
http://localhost/server/php
end
```

Figure 11: Code generated with MobAppGen

The applications using the Remote Database module share similar attributes, including database information. In the first application, the records are organized based on the drugs identification number (D_ID), while in the second one the records are organized using the drug’s name field. The third application is referred to as Messages; its attributes correspond only to the server’s information.

[Fig. 12] shows the schematic block diagram for the generated project. It can be observed that the client-side applications corresponding to the Remote Database module share the classes required to implement the database management on the remote server as well as the classes to transform and manage the data into a record store on the mobile device. In a similar way, the three applications share the required classes for Member management.

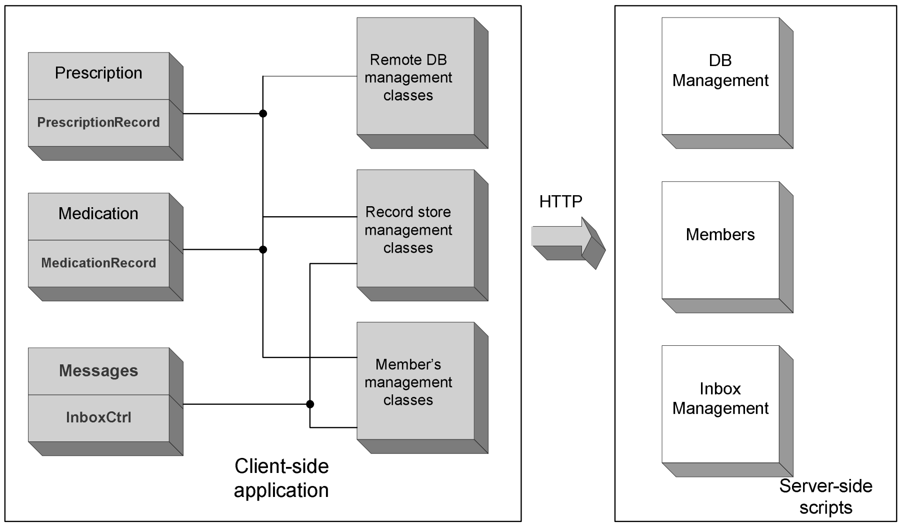


Figure 12: m-Health project block diagram

5 Related Work

There are several projects that may be compared with MobAppGen. The work developed by C.-W. Xu [XU 2006] is a framework that facilitates the modeling, implementation and maintenance of wireless mobile online applications based on Java ME. This work proposes the use of the graphical language MVC (Model-View-Controller) for modeling mobile online applications. The developed applications consist of web applications that access information from a web server and insert data into a database, where JDBC-ODBC (Java Database Connectivity – Open Database Connectivity) is the database bridge driver chosen to access the information. This type of application is particularly useful for mobile devices that do not have a web browser installed. The implementation of the mobile online application consists of the server side and the client side. The server side is based on Servlets and JavaBeans. The client side is based on a MIDlet. The Servlets accept the user's requests, dispatch them to other components for validation and finally generate the corresponding responses. The JavaBeans are components that access the database to query, update or insert data. In this work, every wireless mobile online application must be first modeled using a MVC diagram. A major difference between MobAppGen and this framework is that, as the name implies, this approach is only targeted to web applications. Moreover, the design of the applications and framework structure is more conceptual than practical. There is a set of client and server libraries, but there is no program tool developed to create the design in MVC or to add the libraries to construct a new application. Finally, the client libraries implement only basic functions, such as show screens or request data from the server; the more complex tasks are left to the server side.

Another possibility is QuickFrame (QF) [Hiata 2007], a framework for the development of mobile applications that allows them to run in several types of mobile devices. This framework owns a standard specification language used to define the application interface. QuickFrame's architecture consists of three elements: the QF Designer, which is an Eclipse plug-in used to describe screens and flows in mobile applications; the QF Interpreter, which runs on mobile devices to interpret the applications that were created on the QF Designer; and the QF Server, which is responsible for storing applications described in the QF Designer, exchanging information with mobile applications running in the interpreter and interchanging information with storage systems. The primary goal of QuickFrame is to allow mobile applications to run in several devices without many changes, rather than to facilitate mobile applications creation, as our framework does. Nevertheless, the QuickFrame Designer eases the creation of user interfaces and screen flows using the framework specification language. The created code is interpreted later on the mobile device by the framework's interpreter. A major drawback of the QF Designer is that the developer is forced to use Eclipse IDE to implement an application. Developers may need extra time to familiarize themselves with this Integrated Development Environment (IDE); thus, QF may be more suited for experimented programmers than for beginners.

MobAppGen could also be compared to MobCon [Cepa 2005], which stands for Mobile Container. It employs the container concept adapted to the domain of mobile applications. Logically, a container is a wrapper module that transparently offers services to application components residing inside it. The services provided may vary depending on what the container is built to address. In this framework, services fall into the category of technical concerns that can be reused within various applications. Mobile containers in MobCon deal with the client-side application that resides on the mobile device as a MIDlet. They consist of MIDP transformation plug-ins included in the framework containing functionality to generate client code. Server-side code may also exist to support the MobCon concerns on the server, but it must be manually installed on the server. The basic idea in this work is to use a predefined set of tags (annotations) to build MIDP applications. Tags are declarative constructs used to decorate elements of a program to add semantics to it. End-programmers add these tags to their code to include some predefined technical concerns. The tag-decorated code is then processed, and the required technical concerns are added to it based on the predefined MobCon tags. MobCon is a very complete approach to facilitate mobile application generation. The main disadvantage of the framework is that the developer needs to learn the new language composed by the tags that must be added to the Java code. This addition is done manually, which forces the developer to be familiarized with JavaDoc and the specific MobCon tags before starting an application. Another drawback is that once the code is generated, the mobile application only consists of two classes: the template class with the technical concerns and the abstract class with the code inserted by the programmer. This setup is useful for small applications that do not include many lines but can be a problem for larger applications, which complicates software maintenance.

6 Framework Performance

To test MobAppGen's performance, we evaluated the size of the generated client-side application, which is the JAR file size. The JAR file includes all the files for every class in the MIDP application. It has to be installed on the mobile device to execute the project. The size of the JAR file is important for several reasons. For example, the memory size allocated for MIDP applications on mobile devices is usually very limited. Therefore, users prefer small applications to better employ the available space. Additionally, some mobile devices have a limited size restriction for the JAR file: for example, the Nokia S40 series, among others, has a maximum JAR file size support of 64 kB. In addition, some wireless network operators put a limit on the size of the MIDP applications that can be downloaded from the network to install the application on the device.

In MobAppGen, the client-side application size depends on the type and quantity of modules used to create the project. The framework allows the developer to enter as many modules as she/he desires; thus, the results presented here are intended to provide them with useful design guidelines. Several projects were created to combine the modules available in the framework in order to do the evaluation tests. The projects contain applications created with similar or different modules. The first tests were done with projects that consist of applications generated with client-only modules. [Tab. 2] shows the obtained results. In these tests, only this type of module is used in each project, but several applications are added by changing their attributes. It can be observed that, in this case, the JAR files are less than 25 kB, which is a small file size. The file does not exceed this size even when adding eight different applications in the same project.

No. Applications	**No. Modules**	**Jar (bytes)**
1	1	12371
1	1	12288
2	2	14141
3	2	15872
4	2	18323
5	2	20089
6	2	21923
7	2	23649
8	2	25430

Table 2: Performance results obtained when using client-only modules

[Tab. 3] shows the results obtained when the projects consist of applications generated with the client-server modules. It can be observed that in this case, the JAR file size increases faster than when using the client-only modules because this type of applications requires more source classes. In this case, after we have added seven applications, the project exceeds 64 kB, which is a size limit in some mobile devices.

No. Applications	No. Modules	Jar (bytes)
2	2	21633
3	3	32117
4	4	35779
5	5	53291
6	5	57321
7	5	59243
8	5	65123

Table 3: Performance results obtained when using client-server modules

As it can be observed from the previous results, the projects generated with MobAppGen can be easily maintained below 64 kB if it is required by the mobile device. This result is important because several applications need available memory in the mobile device to store local data.

7 Evaluation Through Qualitative User Study

A qualitative user study was conducted to evaluate the usefulness and acceptance of MobAppGen. There were 23 participants in the user study with an average age of 28. All of the participants are computer engineering students. Of the participants, 13 had no experience in mobile programming, while the other 10 had already had some experience using Java ME. To evaluate MobAppGen, we observed how the participants interacted with the framework and combined it with the participants' answers during the interview. During the study the participants were required to create one project using the framework. They were free to use any modules to complete their project. [Fig. 13] compares the results from experienced and non-experienced users. Most participants agreed that the framework is easy to learn and intuitive to use, but the non-experienced users had more problems using the framework. They stated that this is mostly because they were not familiarized with some of the terms used in the web-based tool, such as *MIDlet*, *JAD* and JAR file, among others. Considering the framework's usefulness, it can be observed that experienced users do not fully agree with this argument. Some of them stated that they could not find an appropriate module to create the project that they wanted to design. Some of the suggestions for new modules included a module to manage MMAPI functionalities and another one to work with files using the *FileConnection* optional package.

In the interview, we also asked experienced users if they foresee using MobAppGen in the future, and if they would use it over other tools that they know of, such as Eclipse or MobCon. Based on the responses, they mostly reported that they would use MobAppGen to develop some of their projects or to start creating a project that they could improve later. Most of them agreed that they prefer using a tool like MobAppGen to using the Java ME SDK provided by Sun. Many (80%) of the experienced users agreed that it is easier to develop applications using MobAppGen than with other tools. The other 20% disagreed because the number of modules included in the framework is still limited.

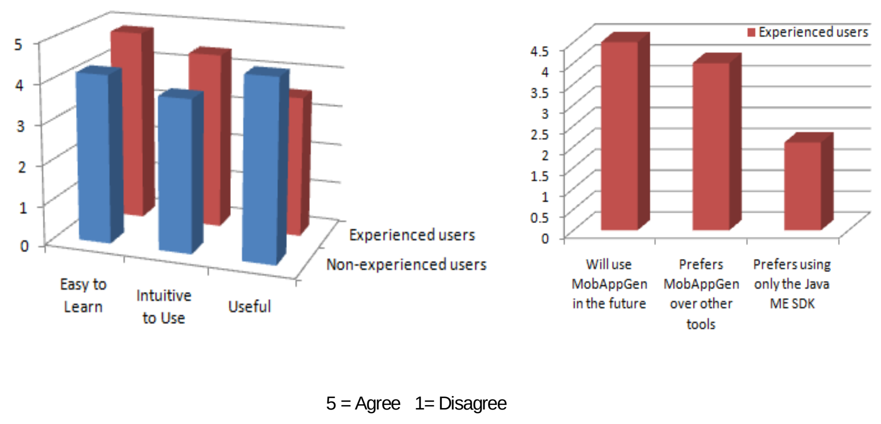


Figure 13: Feedback from experienced and non-experienced mobile programmers when using MobAppGen

8 Conclusions and Future Work

In this work, we have presented the impact of using a web-based modular framework to develop mobile applications. A module has been described in computing as a self-contained component of a system. Our framework, referred to as MobAppGen, has applied modularization to the mobile applications environment. In this work, each module has been designed to perform a particular task commonly used in the development of mobile applications. These modules are characterized by a set of attributes that are used to adapt the particular task to a user's needs. An important aspect of modularization is that the same element can be reused to generate different mobile applications within the same project while sharing common classes and promoting code reuse to reduce the final project's size, which is a very important feature in the mobile applications environment where the available memory and storage space tend to be limited. This modular framework targets mobile computing devices included in the MIDP profile and generates PHP scripts for the server side when required. The framework provides a friendly environment based on a web-based tool that helps to hide the complexity of mobile applications programming and extends framework access to remote locations. It could be observed that the use of a web-based tool provides a friendly environment that was well accepted, even for users that have no experience programming mobile applications. They found the framework easy to learn, intuitive to use and useful for creating the projects that they designed. More experienced users that have already used Java ME to create mobile applications would consider using the framework in the future, even if they find it somewhat limited, to develop some of their projects because they depend on the modules included in the framework. As a result, we are planning to extend MobAppGen by adding more modules or providing a tool that allows experienced programmers to add their own modules to the framework.

Acknowledgements

Thanks to CONACYT for supporting this work by grant 151614 from SEP-CONACYT [SEP (Ministry of Public Education)-Consejo Nacional de Ciencia y Tecnología].

References

[3GPP 2006] 3GPP, "TSGS-SA, Transparent end-to-end Packet Switched Streaming Service (PSS). Protocols and codecs. (Release 7)", TS26.234 v.7.1.0 (2006-12).

[Balagtas 2009] Balagtas-Fernandez, F. and Hussmann, H. "Evaluation of user-interfaces for mobile application development environments". In Human-Computer Interaction. New Trends, 5610 (2009) 204–213

[Beji 2008] Beji, S.; El Kadhi, N.,"An Overview of Mobile Applications Architecture and the Associated Technologies", in: Proceedings of the Fourth IEEE International Conference on Wireless and Mobile Communications, 2008. ICWMC '08, July 27 -Aug. 1 2008 pp.77-83, doi: 10.1109/ICWMC.2008.55

[Cepa 2005] Cepa, V. and Mezini, M., "MobCon: A Generative Middleware Framework for Java Mobile Applications", in: Proceedings of the International Conference on System Sciences (IEEE HICSS-38). Island of Hawaii. January 3-6 2005.

[Handley 1998] Handley, M. and Jacobson, V., "SDP: Session Description Protocol". IETF RFC 2327, 1998

[Hiata 2007] Hiata, A., R. de O. Anido and A. Luiz da Cruz, " An Integrated Approach to Develop Pervasive Mobile Applications", in: Proceedings of the International Conference on Wireless Information Networks and Systems (Winsys 2007), Barcelona, Spain. July 2007.

[Istepanian, 2004] Istepanian, R.S.H., Zhang, Y.T. "Guest Editorial Introduction to the Special Section on M-Health: Beyond Seamless Mobility and Global Wireless Healthcare Connectivity" IEEE Transactions on Information Technology in BioMedicine, 8(4), 2004.

[ITU 2011] ITU ICT World Telecommunication/ICT Indicators Database: "The World in 2011, ICT Facts and Figures" (2011).

[ITU 2010] ITU-D ICT World Telecommunication/ICT: "The World in 2010", (2010).

[Jorstad 2005] Jorstad, I., Dustdar, S. and Do van Thanh., "An analysis of current mobile services and enabling technologies", International Journal of Ad Hoc and Ubiquitous Computing, 1, 1/2 (2005), 92-102.

[Knudsen 2003] Knudsen, J., "Wireless Java: Developing with J2ME", The Author's Press, 2003.

[MMAPI 2002] Mobile Media API JSR-135, 2002 http://jcp.org/en/jsr/detail?id=135

[Prohm 2002] Prohm, B., Dittner, P., Wood, B., "Mobile Terminal Statistics Worldwide", 1996-2005. Gartner Dataquest. 2002

[Reenskaug 2003] Reenskaug, T., "The Model-View-Controller (MVC) Its Past and Present", University of Oslo, 2003, available at http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf

[Schulzrinne 2003a] Schulzrinne H., Rao, A., Lanphier R. and Westerlund, M., "Real Time Streaming Protocol". RFC 2326, March 2003.

[Schulzrinne 2003b] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V., **"**RTP: A Transport Protocol for Real-Time Applications**"**. RFC 3550, July 2003.

[Sjobem 2006] Sjobern J., et al. "RTP Payload and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs", RFC 3267, August 2006.

[Verkasalo 2006] Verkasalo, Hannu., "Empirical Observations on the Emergence of Mobile Multimedia Services and Applications in the U.S. and Europe", in: Proceedings of the 5th International Conference on Mobile and Ubiquitous Multimedia, Stanford, California, December 2006

[Vital 2009] Vital Wave Consulting," mHealth for Development: The Opportunity of Mobile Technology for Healthcare in the Developing World". Washington, D.C. and Berkshire, UK: UN Foundation-Vodafone Foundation Partnership, 2009

[WAP 1998] Wireless Markup Language Specification, The WAP forum, 1998.Available at : http://www.wapforum.org/what/technical.htm, Last visited: June 28, 2010

[Xu 2006] Xu, C.-W. "A Framework for Developing Wireless Mobile Online Applications", in: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International workshop on Component-Based Software Engineering, Software architecture and reuse (ICIS-COMSAR'06). Honolulu, Hawaii. July 2006.