Middleware – An Effort towards Making Mobile Applications Platform Independent

Neeraj Gupta, Vishal Srivastav, M.P.S. Bhatia
Netaji Subhas Institute of Technology,
University of Delhi, Delhi, India
neerajgupta.nsit@gmail.com, vishalsrivasta@gmail.com, mpsbhatia@nsit.ac.in

Abstract

We propose a middleware architecture for GSM mobile phones that makes applications for mobile phones platform independent to a large extent as well as the applications made using the middleware run on a wide range of devices. Currently we have considered Symbian OS and Windows CE OS. The middleware is implemented in C++ and it provides good performance in terms of memory use and execution speed. The middleware requires less than 50 KB of memory and is suitable for resource-constrained mobile phones. The first experiments verify that the middleware operates well and offers a good basis for future development.

1. Introduction

Applications for mobile devices are becoming increasingly sophisticated. Mobile phones are no longer just mobile devices but are used as organizers, data repositories, and the containers for users' applications.

Wireless middleware is still evolving, and it isn't mature enough yet to meet extreme reliability and needs of the application developers. Existing middlewares for mobile phones are not suitable for developing powerful applications due to one or other reason. Some middlewares, such as J2ME, although provide a large set of APIs, but they provide limited APIs for any particular device [1]. Some of them are based on technologies which is not supported by mobile phones such as One.World[9], Homeros[8] and Gaia[5]; some are only for CDMA networks such as Brew[15] while others such as 'Lightweight Middleware for Mobile Phones' [6] do not consider multiple platforms and do not making applications platform-independent.

In this paper, a middleware architecture has been proposed, which eliminates the above said disadvantages of existing middlewares. The proposed middleware targets a broad range of devices, works for GSM phones, does not stress the system with an additional software layer and is based on native code ensuring support of almost all APIs on most phones. Also, it considers multiple platforms and makes applications platform independent to a large extent. Furthermore, since middleware is implemented in C++, it also provides good performance in memory use and execution speed [2]. Symbian and Windows CE OS phones are leading the mobile phone market and will continue to do so in coming years [4]. Consequently, Symbian OS and Windows CE OS are the platforms initially considered for the development of this middleware.

The remainder of the paper is divided as follows: Section 2 talks about related work. Section 3 describes where the middleware stands and what APIs have been developed. Section 4 presents some applications developed as the first experiments and the testing results. Section 5 discusses about the Future Work and Section 6 concludes the paper with a conclusion.

2. Related Work

J2ME is a middleware in use by many application developers now-a-days. J2ME is a device-level middleware which enables development of many applications such as games, appointment books etc. But when it comes to developing powerful applications much beyond games-which harness the whole API set-- such as developing search engine on mobile phones, J2ME is not of much use. This is because for developing a search engine, many APIs are required and although J2ME support many APIs, but not all are distributed evenly across devices. That is, not all packages required for development are supported on the same mobile phone. Reference [1] evinces that J2ME provides a lot of APIs but if we take any particular model, J2ME supports some APIs and not others. That is the reason why even though J2ME makes applications platform-independent, application developers today prefer choosing a

0-7695-2699-3/06/$20.00 (c) IEEE

platform-dependent language such as C++ for their application development rather than using J2ME which provides a means of much rapid application development than C++.

BREW [15] is an application development platform created by Qualcomm. Standing for *Binary Runtime Environment for Wireless*, it is a software that can download and run small programs for playing games, sending messages, sharing photos, etc. But since it is only used for CDMA-based mobile phones, it is not used for application development in GSM-based phones.

Other efforts are also made in developing middleware for mobile phones. Some of them have been discussed below:

Gaia [5] is a distributed meta-operating system designed to facilitate ubiquitous computing. Gaia is based on CORBA [14] technology, which prevents the use of Gaia in resource-limited mobile phones. Several CORBA libraries exist for Java, for example JacORB [10], but they require Java versions that are not supported by commercial mobile phones. Support can only be found in some new communicators, such as Nokia 9500 Communicator. Gaia also uses the high-level scripting language LuaOrb [11], which is not supported by mobile phones and, furthermore, stresses a resource-limited device by an additional interpreter.

One.world [9] provides architecture for building pervasive applications. It targets on applications that automatically adapt to highly dynamic computing environments. But, One.world relies on the Java virtual machine, which provides a uniform execution environment across different devices and hardware architectures. Therefore, the mobile terminal must support Java, which, in turn, stresses the system with an additional software layer. Also, One.world utilizes Java's object serialization, which is not supported by the J2ME version of Java, used in most mobile phones.

HOMEROS [8] is component-based middleware architecture for ubiquitous computing environments. It supports the applications by satisfying all core requirements, but it utilizes CORBA, which makes it impossible to run on current mobile phones. Furthermore, it is targeted on PDA devices and laptop and desktop computers.

Lightweight Middleware Architecture for Mobile Phones [6] is suitable for resource-constrained mobile devices as it uses Symbian C++. But since it is developed in Symbian C++, it can only be deployed on phones supporting Symbian OS. It is not developed for other widely used platforms such as Windows CE OS. In other words, it does not aim at making applications platform-independent.

3. Architecture

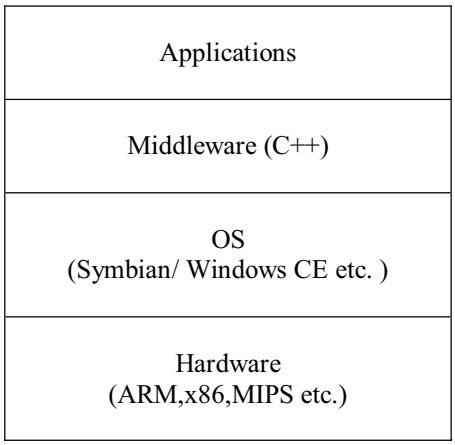

Fig.1. Location of Middleware

The middleware is implemented based upon Fig. 1.

The two leading platforms in the mobile world – Symbian and Windows CE -- which collectively cover more roughly 90% of the market--are considered for the first results. This middleware is composed of platform-specific dynamic linked libraries (DLLs), all of which are implemented in native code i.e. C++. In Symbian, DLLs are implemented in Symbian C++, while, in Windows CE, they are implemented using Win32 APIs. The middleware is developed completely in object-oriented manner, that is, the middleware is implemented using classes and objects are used to model API sets.

The core services provided as of now include messaging, getting HTTP response for GET requests, accessing file system of the phone, retrieval of network information and retrieval of phone information. Messaging currently includes sending and receiving SMS. HTTP response can be obtained both synchronously and asynchronously. Accessing internal file-system of the phone includes opening a file, reading data from a file, writing to a file and closing a file. Network information includes roaming status, operator name and signal strength, city and country name as stored in the handset. Phone information includes hardware information, operating system name and version, device family and model information. Current location of the phone can also be obtained in Symbian while for Windows CE, work is going on to get current location, at the time of writing this paper.

Some of the challenges towards developing a middleware which makes applications platform-independent are:

- Both Symbian C++ and Win32(C++) APIs for Windows CE work with different data types.

For instance, in Symbian, for accessing and manipulating strings, descriptors such as TBuf, TDesC, TPtr etc. provide a safe, consistent and economical mechanism. All access to the data is made through the descriptor objects. While, on the other hand, in Windows CE, datatypes such as LPTSTR and LPCTSTR are used for string handling.

- In Symbian C++, quite many times, for creating an object of a class, two-phase constructor is required so as to avoid memory leakage. But for Win32 APIs, there is no such concept of two-phase constructor.

Some techniques are applied to ensure platform-independence of the applications such as:

- Novel data types are created. *Unicode* characters and strings are used as input arguments and return values for all the function calls to the middleware where character or string is input argument or return type, respectively. The platform-independent datatypes are converted to platform-specific datatypes internally in the middleware provided.
- One static function named "CreateObject" is provided for all APIs for both the platforms. This function creates the object of the class and returns a pointer to the object created. This provides an abstraction over the different constructor call methods in Symbian and Windows CE. To maintain symmetry, a DeleteObject function which takes a pointer to object as input and deletes the object associated with that pointer, is also provided for all APIs.
- Names of APIs for both the platforms are made as same.
- All the classes and structures are made with the same name for both the platforms.
- Number of functions exported from DLLs is same in corresponding APIs of both the platforms with one *GetErrorText* function available for each class which returns the text of the last error occurred while operating on that class in the form of a Unicode String.
- All the function names are made identical with same number of arguments, same argument type and same return values.

All these steps make the interface provided to the application developers simple, intuitive, easily comprehensible and the most importantly-common across both platforms. For e.g. Members of CFileSystemAccess class which handles reading, writing, opening and closing a file are as shown in the Fig. 2.

Public Member Functions

MWINT **Open** (MWLPCSTR fileName, MWLPCSTR mode)

Opens a file in a particular mode.

MWINT **Write** (MWLPCSTR pString)

Writes into the file opened.

MWINT **Read** (MWLPSTR pString, MWINT size)

Reads from the file opened.

MWINT **Close** ()

Closes the file.

MWLPSTR **GetErrorText** ()

This function returns the text of the error last occurred.

Static Public Member Functions

static **CFileSystemAccess** * **CreateObject** ()

Creates an instance of ***CFileSystemAccess*** *Class.*

static MWVOID **DeleteObject** (**CFileSystemAccess** *ptr)

This function deletes the object pointed to by ptr.

Fig.2 CFileSystemAccess Class Reference

As shown in the fig.2. we have created our own data types e.g. MWLPCSTR for handling string constants, MWLPSTR for handling variable strings, MWINT is a type-definition of 'signed int' etc. CreateObject function creates an object while DeleteObject deletes the object pointed to by the argument to the function. An Error Handler function is also provided which returns the text of the error last occurred.

4. Applications

The middleware is tested first on the emulator and then the target device. The middleware is tested on both Motorola A925 and Nokia 6600 supporting Symbian OS and O2's XDA XPhone II smartphone supporting Windows CE OS. It is tested using Unit Testing, by making simple applications to test all the APIs one by one and then integration testing, by making a single application for each platform which tests all the APIs for that platform. These experiments verify that middleware operates well and offers a good basis for future development. Some steps are written in Fig. 3 to demonstrate what application developer needs to do in his application code when he wants to send a SMS.

Note that the middleware provides a very simple and intuitive a solution to such involving a problem. As shown in Fig.3, sending SMS is boiled down to just one function call excluding some necessary operations such as creation or deletion of object.

1. Include Header File of API in the application using
   #include “Messaging.h”

2. Create an object using CMessaging* pMsgObj;
   pMsgObj = CMessaging::CreateObject();

3. Send the SMS using
   pMsgObj->SendSms(messageToSend, phoneNumber);

4. Delete the object created by CreateObject
   CMessaging::DeleteObject(pMsgObj);

5. Link against Messaging.lib provided with Middleware.

6. Compile and run the application.

Fig.3 Method to Send SMS in the application using Messaging API.

Fig.4 shows some of the testing results. Fig. 4 (a) shows one sample application using FileSystemAccess API. It takes a file name (here file.txt) and text to write to that file (here “Data of file.txt”) as input and stores it to the file when selected from menu. Immediately after that it displays the content of that file in the middle-textbox reading that file. Fig. 4(b) shows the output of PhoneInfoRetrieval API tested on Motorola A925 phone. Fig. 4(c) shows an application which displays the location of the phone. It is tested on Nokia 6600. It displays “Loc:Naraina Vhr” which shows the correct location i.e. “Naraina Vhr”. Fig. 4(d) is the output of InternetData API and FileSystemAccess API. It is displaying the response of www.google.com stored onto a file named internet.txt. It is tested on Motorola A925. We can parse it as required by the application to extract important information from the web page source.

For Xphone II supporting Windows CE, Fig. 4(e) illustrates the output of test application of InternetData API and Fig. 4(f) displays the output of testing PhoneInfoRetrieval API on Smartphone 2003 emulator. Fig. 4(g) shows the output of FileSystemAccess API tested on XPhoneII.

Network information is also tested on Nokia 6600 and Xphone II which includes roaming status indicating whether phone is on roaming or not, operator name (for e.g.- Hutch, Idea, Airtel etc.) and signal strength in percentage at any instant. Messaging API (Send/Receive SMS) is also fully tested on Motorola A925, Nokia 6600 and XPhone II.

Fig. 4 (a)
Testing FileSystemAccess API on Motorola A925

Fig. 4 (b)
Testing PhoneInfoRetrieval API on Motorola A925

Fig. 4 (c)
Testing Location Information On Nokia 6600 phone

Fig.4 Results of testing the middleware on Motorola A925 and Nokia 6600 having Symbian OS and XPhoneII having Windows CE OS on device as well as emulator.

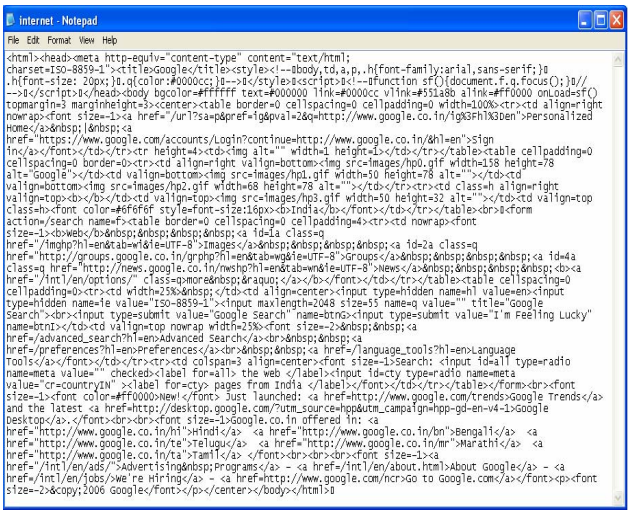

Fig.4 (d) Text file which stores the HTML page of www.google.com using Internet Data API on A925

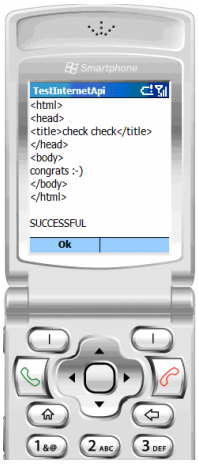

Fig. 4 (e) Testing InternetData API on XPhoneII

Fig. 4 (f) Testing PhoneInfoRetrieval API on Smartphone2003 Emulator

Fig. 4 (g) Testing FileSystemAccess API on XPhoneII

Fig. 4 (h) GETNETW Request by a phone

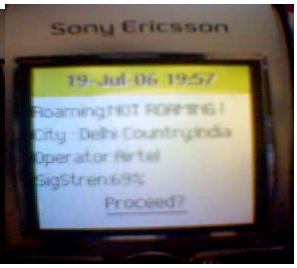

Fig. 4 (i) Reply of GETNETW to the phone

Fig.4 Results of testing the middleware on Motorola A925 and Nokia 6600 having Symbian OS and XPhoneII having Windows CE OS on device as well as emulator. (Cont.)

Many other small applications are also developed to test the APIs, for e.g. sharing your location with friends using NetworkInfoRetrieval API and Messaging API, tracking the information of another user by sending request and automatically getting reply depending upon the message sent. For instance, if the message is GETLOC, location of the requested phone is sent as SMS to requestor; if the message is GETPHONE, phone information of the requested phone is sent as SMS to requestor; and if the message is GETNETW, other network information of the requested phone is sent as SMS to requestor. Fig. 4(h) shows the GETNETW request sent by a phone to a Nokia 6600 phone and in reply it gets the response as shown in Fig. 4(i) which indicates Nokia 6600 phone is not on roaming, it is in Delhi, country is India, operator name is Airtel and signal strength is 69%

The middleware is developed with the aim of developing applications like a search engine for getting information about restaurants, cinemas etc. near some place provided by the user or near the place where phone is. The user can use Messaging API to send SMS about the place whose information he wants or the current location may automatically be appended at the end of the message by using both the Location String information and the Messaging API. He can use ReadSMS function to wait for SMS from a particular sender who sends the result of the query. FileSystemAccess API is then required for reading and writing logs and PhoneInfoRetrieval API for storing information about the phone in the logs. As an alternative, people can use this search service using GPRS also through HTTP requests. A server is available which is ready with a retort in case of SMS or HTTP requests.

5. Future Work

The middleware currently is developed only for phones supporting Symbian or Windows CE Operating Systems. In the same way, APIs can be made for Linux and Palm OS and integrated to the current middleware. The APIs can be made for the Palm OS using the C++ development SDK[7].

At this stage, middleware has support for SMS messaging, getting HTTP response for GET requests synchronously and asynchronously, file system access, retrieval of network information and retrieval of phone information. Many other APIs can be added to the middleware such as multimedia components, user interface components etc. Future Work also includes developing powerful applications using the current middleware like those discussed at the end of Section 4.

At this stage of the project, user interface API is not a priority and is therefore not included in the API set

because, user interface requirements keep changing from time to time. Secondly, since there are a number of user interface platforms in mobile phones, so developers may require to change the user interface according to the screen-size, resolution etc. suiting that platform. Thirdly, user-interface development is made easier by some tools already available. For e.g. for Symbian, C++BuilderX [12] is available which can create User Interface just by Drag and Drop Technique. Another tool called qub[13] creates User Interface for UIQ phones very easily. Moreover, even J2ME program does need a fair amount of adjustment before it could run on different handsets [3]. As well, different devices running on the same platform may use different ways of interacting with the user such as alphanumeric keyboard, touch screen pen input, QWERTY keyboard etc..

6. Conclusion

This middleware is developed mainly to demonstrate an alternative to the current middlewares available (or proposed) which lack one or more features as discussed in the Section 2. But this middleware fulfills all the requirements.

- Doesn't stress the system with any additional software layer.
- Applications developed target a wide range of devices.
- Makes applications platform-independent.
- Based on C++ which is supported by almost all mobile phone platforms.
- Native code (C++) makes the code efficient in both space as well as time.
- Works on GSM phones.

So, the middleware offers a good basis for future development and can definitely act as a substitute for other middlewares if other APIs are also added.

7. Acknowledgements

The authors would like to thank Dr. Huzur Saran (Professor, Indian Institute of Technology (IIT), Delhi) for his continuous support and guidance in making the project a success.

References:

[1] Sun Microsystems, The Java ME Device Table, URL:http://developers.sun.com/techtopics/mobility/device/device, Accessed May 11th, 2006

[2] Symbian Developer Library, Programming Languages, URL:http://www.symbian.com/developer/techlib/v70sdocs/doc_source/devguides/ProgLanguages.html, Accessed May 11th, 2006

[3] Ben King, To J2ME or not to J2ME - that is the question, URL:http://www.benking.co.uk/art/J2me_or_not.php, Accessed May 11th, 2006

[4] Windows CE trails Symbian in converged mobile devices, URL:http://www.windowsfordevices.com/news/NS2122263837.html,Accessed May 11th, 2006

[5] Ellick Chan, Jim Bresler, Jalal Al-Muhtadi, Roy Campbell, "Gaia Microserver: An Extendable Mobile Middleware Platform", Proceedings of the 3rd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom 2005), 309-313

[6] Timo Salminen and Jukka Riekki, "Lightweight Middleware Architecture for Mobile Phones", Proc. 2005 International Conference on Pervasive Systems and Computing, Las Vegas, NE, 147-153.

[7] PalmSource, Developer Documentation, URL:http://www.palmos.com/dev/support/docs/, Accessed May 11th, 2006

[8] Han, S.W., Yoon, Y.B., Youn, H.Y. (2004) "A New Middleware Architecture for Ubiquitous Computing Environment". Proceedings of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS'04)

[9] Grimm, R. (2004) "One.world: Experiences with a Pervasive Computing Architecture", IEEE Pervasive Computing vol. 3, pp. 22-30

[10] JacORB, URL:http://www.jacorb.org, Accessed May 11th, 2006

[11] LuaOrb Online, LuaOrb, URL:http://www.tecgraf.puc-rio.br/~rcerq/luaorb, Accessed May 11th, 2006

[12] CBuilderX for Symbian, URL:http://www.newlc.com/Free-Download-C-BuilderX-1-5.html, Accessed May 11th, 2006

[13] Open-qub – a graphical resource editor for UIQ, URL:http://www.newlc.com/Open-qub-a-graphical-resource.html?var_recherche=qub, Accessed May 11th, 2006

[14] CORBA, URL:http://www.omg.org/corba/, Accessed May 11th, 2006

[15] QUALCOMM, Binary Runtime Environment for Wireless, URL: http://brew.qualcomm.com/brew/en/, Accessed May 11th, 2006