JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 23, 1391-1405 (2007)

An Adaptive and Unified Mobile Application Development Framework for Java*

MING-CHUN CHENG[1] AND SHYAN-MING YUAN[1,2]
[1]*Department of Computer Science*
National Chiao Tung University
Hsinchu, 300 Taiwan
[2]*Department of Computer Science & Information Engineering*
Asia University
Wufeng, 413 Taiwan

Although wireless networks and mobile devices have become prevalent, the diversity of mobile devices and unsteadiness of wireless networks still cause software development much trouble. In addition, the variety of services in Internet or Intranet, such as Web services, UPnP services, Jini services and so on, will also increase the difficulty in using them. Thus, when developing a mobile application to access these services, developers are forced to expose to these problems and therefore it will spend much time writing a mobile application. Although many studies on user interface adaptation and language transformation have attempted to solve the problems, most of them do not consider the computing power and functionalities of end-devices. As a result, some resources are ignored or wasted. To solve above problems, an adaptive framework, named GMA, is proposed to help developers build mobile applications quickly and easily. GMA framework can tailor an application to fit different devices according to not only user interface formats but also the computing power and functionalities of the devices. Besides, a universal service interface is proposed and developers can use a unified API to access different backend-services. As a result, a mobile application developed on GMA can enjoy the "write once, run everywhere, and access anything" benefit.

Keywords: computing model adaptation, user interface adaptation, network protocol adaptation, universal service interface, bytecode manipulation

1. INTRODUCTION

Nowadays, mobile devices, wireless networks and service technologies become very prevalent and widespread. Hence, the requirements for developing mobile applications have increased. However, there are many differences among them. First, these devices may have different runtime environments. For instance, some of them comply with WAP [1], some with J2ME [2], and some with Microsoft .NET CF [3]. Second, the computing power and functionalities of these devices are diverse and they may have different hardware resources. Third, these devices may support different kinds of networks, such as GPRS, UMTS and WiFi. These networks have different bandwidths, latency, and reliability, and they may disconnect during use. Fourth, there are many different service technologies, such as Jini [4], UPnP [5] and Web services [6], and they have different

Received November 15, 2006; accepted February 15, 2007.
Communicated by Sung Shin and Tei-Wei Kuo.
* This paper was partially supported by the National Science Council of Taiwan, R.O.C., under grants No. NSC 95-2752-E-009-PAE and NSC 95-2221-E-009-021.

usages. These differences all increase the complexity of developing a mobile application capable of supporting them all. Developers have to face these issues, and have spent much time solving them.

According to the above discussion, writing an application capable of supporting multiple devices is difficult. Thus, many studies and standards have tried to solve them. For example, Mobile Execution Environment (MExE [7]) defined by a 3GPP working group categorizes these devices into four execution environments, named classmark 1-4, to reduce mobile application development complexity. Different classmarks mean different execution environments. If a mobile application was developed for classmark 1, it can be run on all devices which conform to classmark 1. Consequently, before developing a mobile application, developers have to decide which classmarks the application will support. This approach makes developers focus on specific execution environments, and implies that the application cannot support devices belonging to other classmarks. To overcome this problem, many studies have been made on adaptations and attribute programming [8], including user interface adaptation [9-13] and programming language transformation [14, 15]. They can tailor the application to fit different user interface formats or execution environments. However, most of them do not consider the computing power and functionalities of devices and these resources are ignored or wasted. One of aims in this paper is to design and implement a generic mobile application (GMA) development framework. Every application developed from GMA is capable of tailoring itself to fit different devices or situations according to user interface formats and the computing power and functionalities of the devices. In other words, more powerful devices will do more things in GMA.

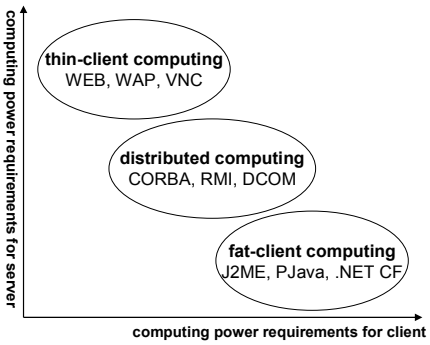


Fig. 1. Computing power requirements for client and server.

A server supports weak devices in this study, helping them do something they can not do. Thus, every GMA application, called GMApp, can be viewed as client-server computing [16]. Fig. 1 shows the computing power requirements for three different computing paradigms derived from client-server computing, and every computing paradigm has many different state-of-the-art technologies. These three computing paradigms have different computing power requirements for clients. By adapting an application to one of the three computing paradigms, all kinds of devices can be well supported regardless of their computing power and functionalities. In addition, a universal service interface is proposed in this paper to integrate different backend-services such as UPnP, Jini and

Web services. To support different end-devices, a GMApp is designed to run in any of three different modes: BROWSER, STANDALONE and MASTER-SLAVE.

1. The computing power of the end-device is not good enough or the device cannot run application other than built-in applications. BROWSER (thin-client computing) mode is suitable for this situation and the device is responsible for user-interface only.
2. The computing power of the end-device is good enough and the device supports all functionalities which the application requires. STANDALONE (fat-client computing) mode is suitable for this situation and entire application codes are executed by the device independently, like running a J2ME MIDP [17] application.
3. The computing power of the end-device is good enough but the device does not support all functionalities which the application requires. MASTER-SALVE (distributed computing) mode is suitable for this situation and the application has to be divided into two parts. One part is executed by the device and the other part is handled by the other powerful host (server).

2. SYSTEM ARCHITECTURE

GMA framework uses a three-tier architecture, as Fig. 2 shows, to solve the problems of diverse computing power and functionalities. End-users use their own desktops or mobile devices, called GMAClient, in the front-tier to access mobile applications. In order to fit different end-devices, there are four kinds of GMAClient in this framework: (1) built-in Browser, (2) GMABrowser, (3) GMAppSlave, and (4) GMAppStandalone. There is at least one application server, called GMAServer, in the middle-tier, which provides necessary execution environments and services for running applications and end-devices. An application, named GMApp, in the GMA framework is designed to be run in the front-tier (STANDALONE mode), in the middle-tier (BROWSER mode), or even in both tiers (MASTER-SLAVE mode) simultaneously depending on the computing power and functionalities of end-devices. More computing power in the front-tier means more codes will be run in the front-tier (implicitly fewer codes will be run in the middle-tier).

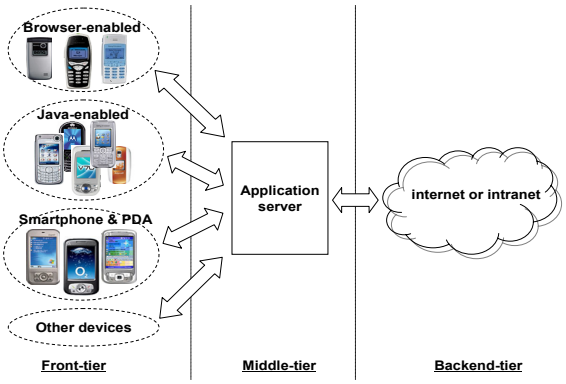


Fig. 2. A diagram to sketch the three-tier architecture.

2.1 GMAServer

The GMAServer plays an important role in BROWSER and MASTER-SLAVE running modes. Some GMApp codes are executed by the GMAServer in these two modes. GMAServer architecture is shown in Fig. 3 and it is based on OSGi [18] platform and J2SE [19]. Furthermore, GMAServer can cooperate with other installed bundles (UPnP, Jini, and Web services) to access backend-services. To simplify maintenance, GMAServer is designed as a layered architecture; with an Application Runtime Layer, Message Routing Layer and Adaptive Transport Layer from bottom to top.

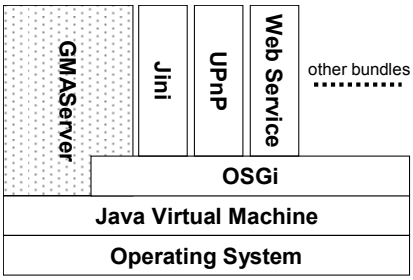


Fig. 3. The architecture of GMAServer.

2.1.1 Adaptive transport layer

The Adaptive Transport Layer enables the GMAServer to communicate with different kinds of GMAClients. Fig. 4 helps illustrate the detailed GMAServer structure. This layer's primary role is Communication Manager (CommMngr), which is a super daemon capable of handling many different networks protocols including TCP, UDP, HTTP and cHTTP. It has two missions. First, it establishes the relationship between the GMAServer and the GMAClient when the GMAClient sends a login request to the GMAServer. Secondly, if login is successful, CommMngr creates a logic process (*i.e.* a user process, including a UserOutD, a UserInD and a UserOutQ) for the GMAClient. Every logic process might have different components or functionalities depending on which protocol it uses. The UserOutD thread is responsible for picking GMAMesgs from the UserOutQ queue and sending them to the corresponding GMAClient or translating GMAMesgs to specific formats [20]. The UserInD thread handles or translates incoming requests from its client and put them into the InnerQueue queue.

2.1.2 Message routing layer

This layer is the asynchronous message delivery mechanism core. The main components are Queue and Message Dispatcher (MesgDispatcher). The MesgDispatcher is responsible for routing GMAMesg to the correct queue. There are three kinds of queues on GMAServer: InnerQueue, AppInQ and UserOutQ.

All messages received by UserInD are placed in InnerQueue. Then MesgDispatcher will dispatch them to the some AppInQ in which GMApp will process these GMAMesg.

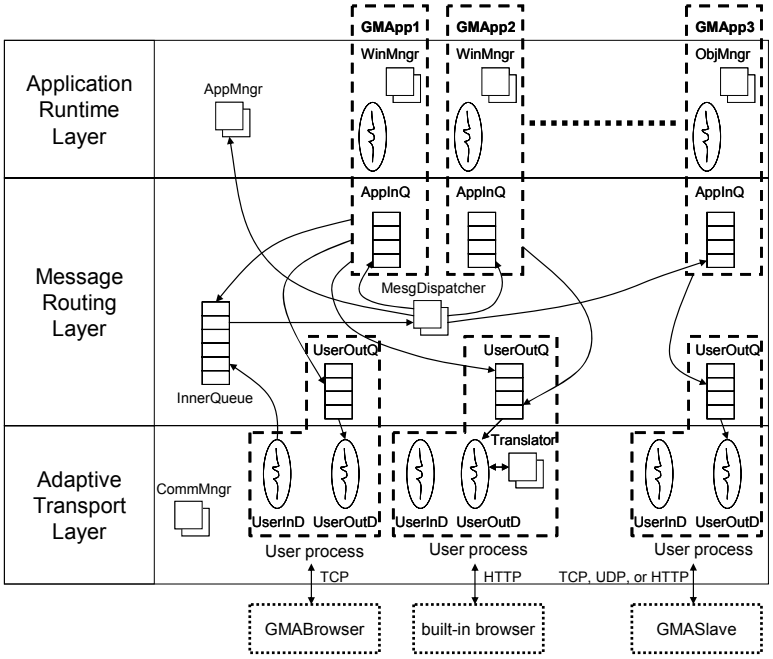


Fig. 4. The detailed architecture of GMAServer. In this example, GMApp1 and GMApp2 run in BROWSER mode. GMApp3 runs in MASTER-SLAVE mode.

Once a GMApp generates a GMAMesg whose destination is a GMAClient, the message will be placed in the user process UserOutQ. UserOutD in this user process will later send the message to its client or pass the message to the translator.

2.1.3 Application runtime layer

A GMAServer can serve many GMAClients at the same time. Also, the GMAClient can access several GMApps run on the GMAServer at the same time if the GMAClient is a GMABrowser or a built-in browser. The Application Manager (AppMngr) is responsible for loading, resuming and stopping GMApps. Before starting a GMApp, AppMngr will check if any instance of the GMApp already exists in the memory. If it does, AppMngr will then check the startup setting of the GMApp and decide to create a new instance or bind the GMAClient to the old one. This is useful when a network is temporarily broken. When the GMAClient re-connects to the GMAServer, previous work can continue. AppMngr uses different class loader instances to load a GMApp every time to maintain independent space between them. This lets every GMApp have its own space.

2.2 GMAClient

According to the above discussion, there are four kinds of GMAClient: built-in browser, GMABrowser, GMAppSlave, and GMAppStandalone. Different GMAClient uses different protocol to communicate with GMAServer.

2.2.1 Built-in browser

When a GMApp is deployed as BROWSER mode, it can be accessed by built-in browser. There are many different kinds of built-in browsers, such as XHTML browsers, WAP browsers, and others. They may use different network protocols to communicate, including HTTP, cHTTP, and WAP. This means that GMAServer must support these different protocols. Currently, most mobile devices have a built-in browser and users can use these browsers to interact with GMApps without installing any extra application. In the other words, most devices can access GMApp by this way.

2.2.2 GMABrowser

When a GMApp is deployed as BROWSER mode, it can be accessed by GMABrowser also as Fig. 5 illustrates. A GMABrowser is a mobile application capable of drawing UI widgets and handling end-user actions. A GMABrowser only can be installed on programmable end-devices. Currently, the most popular mobile device programming environments are J2ME MIDP and .NET CF. Two editions of GMABrowser are implemented in GMA to support both environments.

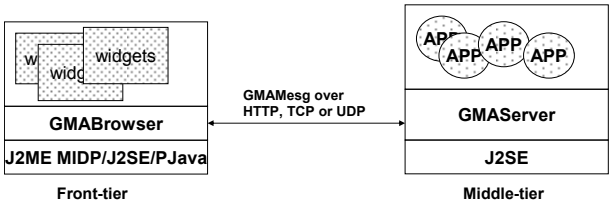


Fig. 5. The diagram about relationship between GMABrower and GMAServer.

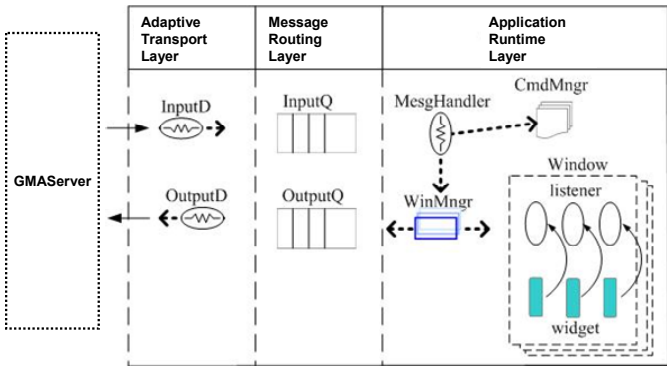


Fig. 6. The detailed architecture of GMABrowser.

GMABrowser architecture is similar to the GMAServer as Fig. 6 illustrates. Because a GMABrowser can communicate with only one GMAServer at a time, there are only a couple of InputD and OutputD in the Transport Layer. InputD always listens for an arriving GMAMesgs; if it gets any, it will add the GMAMesgs to the InputQ. At the

same time, MesgHandler retrieves GMAMesgs from InputQ asynchronously and passes them to the Command Manager (CmdMngr) or WinMngr. The functionality of WinMngr is similar to the one on the GMAServer. The difference is that there is only one WinMngr here. CmdMngr plays almost the same role that the GMAServer AppMngr does, but it does not physically load or stop GMApp instances. It only issues those requests to the GMAServer and waits for the results.

Every widget has an associated listener. Whenever the status of a widget is changed by its user, the listener is triggered and generates some corresponding GMAMesgs. WinMngr then puts these GMAMesgs into OutputQ, and OutputD will later send them to the GMAServer.

2.2.3 GMAppSlave

When a GMApp is deployed as MASTER-SLAVE mode, the application is divided into two parts: master part and slave part. The slave part is a Java jar file capable of installing on mobile devices. It is executed by GMAClient and is named GMAppSlave. The architecture of GMAppSlave is similar to GMABrowser. All communications between GMAServer and GMAppSlave are handled by Adaptive Transport Layer. The only difference is in the Application Runtime Layer. In GMAppSlave, this layer only contains an ObjMngr which is responsible for object management.

2.2.4 GMAppStandalone

When a GMApp is deployed as STANDALONE mode, the application is a full mobile application capable of running on end-devices independently, and it is named GMAppStanalone. It is just like a J2ME MIDP application.

3. ADAPTATION AND INTEGRATION

This section will express how a GMApp can adapt to different running modes and how to integrate different kinds of backend-services.

3.1 Computing Model Adaptation

Generally, a Java application is consisted of classes and all the classes will be executed by the same host. However, in GMA, because some classes may not be executed by GMAClient, these classes have to be handled by GMAServer. Hence, classes within a GMApp have to be separated into two parts (MASTER-SLAVE mode). One part is executed by GMAClient and the other part is executed by GMAServer in runtime. GMApp developers do not need to worry about which computing model is applied and do not need to write any interface description file such as CORBA IDL [21] in development time. All things are handled by GMA automatically and the minimum dividable unit in GMA is class file.

To separate an application into two parts, two problems must be solved. The first problem is how to intercept all actions which act on non-local classes or objects which

associate with non-local classes. The second problem is how to reflect these actions on the corresponding remote classes or remote objects. The GMA framework must intercept these actions and delegate them to their corresponding remote classes or remote objects.

Because many end-devices are J2ME-enabled and J2ME dose not support dynamic class loading [22], the GMA framework generates proxy classes in advance to solve the first problem. Every proxy class has the same class name, skeleton and inheritance relationship as the original class, but there are no fields in the proxy class as Fig. 8 shows. Moreover, the codes within a class and its corresponding proxy class are different. The former is practical business logic and the latter is responsible for delegating intercepted actions to their corresponding object managers on the other side. When a method within a proxy class is called, the codes within the method are run as the following steps:

1. (Marshaling) Encode action type, target object id, method id, and all parameters into a specific command format.
2. Transfer the command to the object manager on the other side and wait for return results.
3. (Unmarshaling) Decode results into original return type and return it to the caller.

Because J2ME does not support Java reflection [23], the GMA framework generates an object manager class, named ObjMngr, for a GMApp in advance to solve the second problem. All ObjMngr classes are responsible for delegating actions to the corresponding classes or objects. A method table is hard-coded in every ObjMngr class and is generated in deployment time. When receiving a command from proxy classes on the other side, ObjMngr will traverse into the method table and then invoke the corresponding methods. The steps are as follows:

1. (Unmarshaling) Decode command from proxy classes.
2. Traverse into the method table.
3. Invoke the corresponding methods within the practical object or class.
4. (Marshaling) Encode the results into a specific command format.

Java is an object-oriented language and all objects are created from classes. Every object is an instance of a class. In Fig. 8 (a), if ObjectX is created from ClassB, ObjectX will have three fields: field1, field2 and field3. This obeys inheritance associations. If ClassA and ClassB is placed on the same host, in this example, HostA, there is no trouble. In Fig. 8 (b), ClassA in HostB (GMAClient) is replaced by a proxy class and ClassB in HostC (GMAServer) is also replaced by a proxy class. In Fig. 8 (b), if ObjextY is created from ClassB, ObjectY will have only two fields: field2 and field3, because of no fields in proxy classes. To solve the problem, every object in GMA will have a complementary object in the other host. In the example, the complementary object of ObjectY will be created from ClassB on HostC, and it will have one field: field1. This means that a logical object may be divided into two parts physically. Traditionally, when an object is created from a class, its constructor will be called and the constructor will call the constructor of the super class. For example, in Fig. 8 (a), when an object is created from ClassB, field1 is initialized first, constructor ClassA( ) is completed , then field2 as well as field3 are initialized, constructor ClassB( ) is completed finally. The order of this initialization

has to keep after replacing some classes with proxy classes. To achieve that, the every original class has to be modified as follows:

1. Add a special constructor which is used when creating a complementary object. This constructor does nothing but only call the special constructor of its super class. Thus, a complementary object created from the special constructor only initializes its fields only and the original constructors are never called.
2. Move all codes within an original constructor to another new method.
3. Modify the original constructor. The codes within the original constructor are modified to call the new method created in step 2.

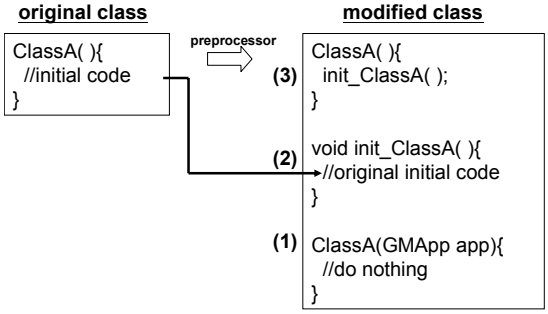
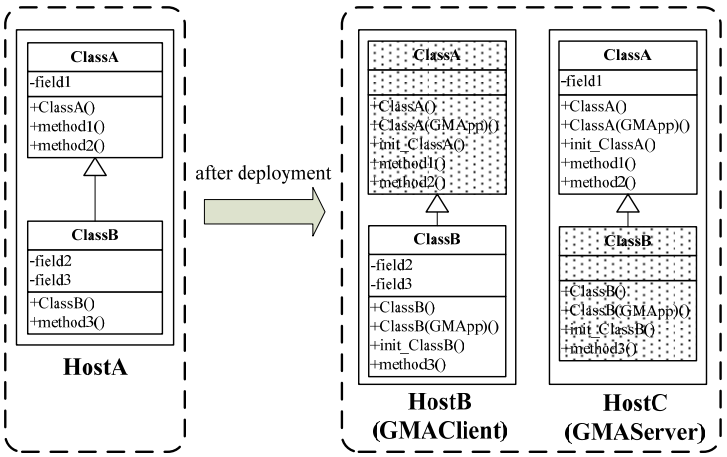


Fig. 7. An example to demonstrate the above steps.



(a) (b)

Fig. 8. An example to demonstrate how to separate two associated classes into two different hosts; (a) describes original class relationship and (b) expresses the result after replacing some classes with proxy classes. A class with shadowed color is a proxy class.

Fig. 8 (b) is an example after modifications. When an object is created from ClassB on HostB, the order of initialization is as following:

1. Initialize fields of ClassA on HostB. Because ClassA is a proxy class, no field will be initialized.
2. The constructor of ClassA on HostB is called (from the constructor of ClassB in HostB). Because ClassA on HostB is a proxy class, the constructor of ClassA is responsible to create a complementary object in the other host (HostC). The details will be discussed in section 3.1.1.
3. A complementary object is created from ClassB on HostC. The special constructor ClassB(GMApp) is used to do that.
4. Initialize fields of ClassA on HostC. The complementary object has one field: field1.
5. The remaining codes within the constructor of ClassA on HostB are executed. It will call the remote method init_ClassA( ) of ClassA on HostC to initialize. The details will be discussed in section 3.1.1.
6. Initialize fields of ClassB on HostB. The object has two fields: field2 and field3.
7. The remaining codes within the constructor of ClassB on HostB are executed.

These steps are similar to original initialization process but original object is divided into two objects.

In the Java Virtual Machine, four kinds of action can act on a class or object: instance creation action, method invoke action, field manipulation action, and synchronized action. Moreover, the last two actions have to be converted to method invoke action first and then they can be treated as method invoke action. The following content describes how to generate proxy class and how they can intercept the first two actions individually. All modifications are made on Java bytecode [24] level.

3.1.1 Instance creation action

According to the previous discussion, every object in GMA will have a complementary object in the other host. All instance creation action can be intercepted by building a corresponding proxy class which has all constructors the original class has. When a proxy class constructor is called, it will check whether the complementary object had been created on the other host. If it was not, these constructor codes within the proxy class will delegate instance creation actions to the corresponding object manager on the other side to create the corresponding complementary objects.

Every object manager maintains an object table which saves the relationship between object ID and object reference. Every object created by object manager has a unique ID and will be saved in the object table. After a complementary object is created, the object ID of its corresponding proxy object will be set the same object id.

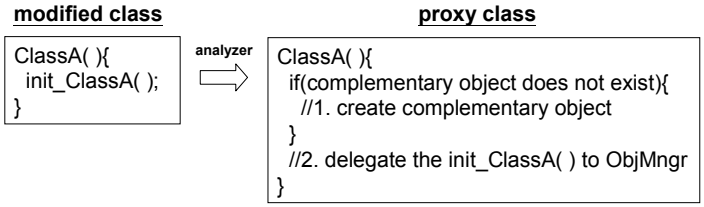


Fig. 9. How to intercept instance creation action.

After that, the remaining codes within the constructor of proxy class will do a method invoke action to initialize the complementary object. In other words, every constructor in proxy class has two missions as Fig. 9 depicts. First, check whether the corresponding complementary object exist on the other host. If not, notify remote object manager to create it. Second, notify remote object manager to call the corresponding initialization method.

3.1.2 Method invoke action

Intercepting the method invoke action is almost the same as intercepting the instance creation action mentioned above. When a method of the proxy class is invoked, it means the method invoke action has to reflect on the complementary object. The codes of these methods within the proxy classes are responsible for delegating these actions to the corresponding object manager in the other side as Fig. 10 shows.

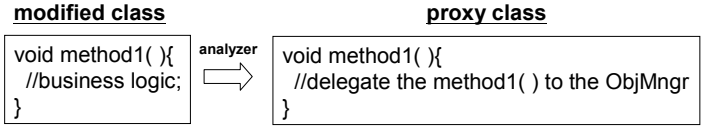


Fig. 10. How to intercept method invoke action.

Every proxy object has the same object ID as its remote object, and GMA gives every method within proxy classes a unique method id. When receiving invoke commands, the object manager can get object references from the object table by the object ID encoded in the command.

3.2 Universal Service Interface

A universal service interface is provided in GMA and Fig. 11 is its diagram. Adapter design pattern [25] is used to integrate different service interfaces. They convert the interface of a class into another interface clients expect. With the universal service interface, every GMApp can use unified API call to control physical devices or access services in Internet without proprietary protocol knowledge. There are three kinds of API in the universal service interface: discovery, invoke action, and event. Discovery API is used to discover available devices or services. After discovery, invoke action API is used to control or access these devices or services. Event API is used to provide an interface to receive events fired by devices or services.

It is worth taking notice that some descriptors are necessary for some adapters, such as Web services adapter. Every descriptor is an XML [26] document and it describes related information about target services, such as URL of WSDL [27] for Web services.

4. DEVELOPMENT AND DEPLOYMENT PROCESS

Different running mode requires different deployment processes, as Fig. 12 shows. Three important components participate in the deployment process. They are preprocessor,

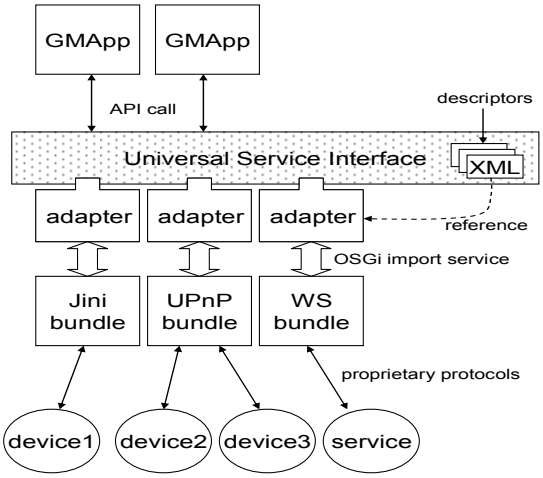


Fig. 11. The architecture of universal service interface in GMA.

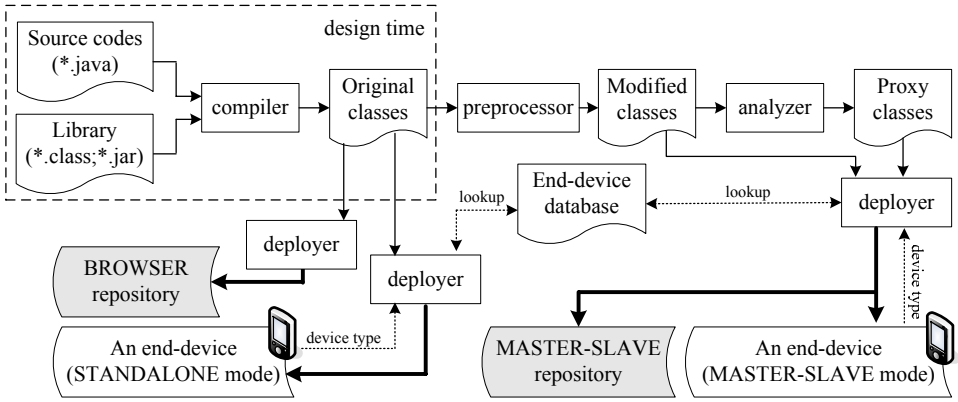


Fig. 12. The GMApp development flow.

the analyzer and the deployer. The preprocessor modifies the bytecodes within original classes. Every modified class is equivalent to the original class, but bytecodes have small differences. What preprocessor has to do had been discussed above and the following is a summary.

1. add special constructor which is used to initialize a complementary object.
2. add initialization method which is called when its corresponding constructor of the proxy class is called.
3. replace all field manipulation action with method invoke action.
4. replace all synchronized action with method invoke action.

The analyzer generates corresponding classes, including proxy classes and ObjMngr classes, by analyzing the bytecodes as the previous discussion (sections 3.1.1 and 3.1.2). Both preprocessor and analyzer exploit a bytecode manuiplation tool, BCEL [24], to

handle all bytecode modifications. The deployer packages necessary classes together in a Java jar file by adding or replacing some classes. It will lookup end-device database during deployment. The end-device database is consisted of two XML documents : device profile and class profile. The former describes capabilites about end-devices and the latter describes the requreiments of classes. The deployer use informatin in end-device database to choose sutiable classes which can be original or generated classes.

Write a GMApp is similar to write a J2ME MIDP application as Fig. 13 shows. Every GMApp has a main class which must inherit from the org.dcslab.gma.app.GMApp class. In different running modes, different edition of GMApp class is chosen and they have different initial process. Every GMApp class is responsible for initializing necessary resources and architecture for different running modes.

```
public class TestMIDlet
  extends javax.microedition.midlet.MIDlet {

  public TestMIDlet() {
    //constructor
  }
  public void startApp() {
    //this will be called, when MIDlet is started
  }
  public void pauseApp() {
    //this will be called, when MIDlet is paused
  }
  public void destroyApp(boolean unconditional) {
    //this will be called, when MIDlet is destroyed
  }
}
```

```
public class TestGMApp
  extends org.dcslab.gma.app.GMApp {

  public TestGMApp() {
    //constructor
  }
  public void startApp() {
    //this will be called, when GMApp is started
  }
  public void pauseApp() {
    //this will be called, when GMApp is paused
  }
  public void destroyApp(boolean unconditional) {
    //this will be called, when GMApp is destroyed
  }
}
```

Fig. 13. The left portion is a J2ME MIDP sample code and the right portion is a GMApp sample code. They are almost the same except extending different class.

5. CONCLUSION

In this paper, a novel development framework GMA, which is capable of tailoring mobile applications to fit different end-devices and environments, is proposed and how it works is discussed in the previous sections. By using GMA, when developing a mobile application, developers do not need to concern about the computing power as well as functionalities of the target end-devices and these resources will be effectively used. Besides, a universal service interface is proposed also. Developer can use unified API to access different backend-services without background knowledge. In addition, because XML document is flexible and extensible, anyone can easy to extend the end-device database to support more end-devices.

REFERENCES

1. WAP, http://www.wapforum.org/.
2. J2ME – Java Micro Edition, http://java.sun.com/javame/.

3. Microsoft .NET Compact Framework, http://msdn.microsoft.com/netframework/programming/netcf/default.aspx.
4. Jini, http://www.jini.org.
5. UPnP, http://www.jini.org.
6. JSR-172, J2ME Web Services Specification, http://jcp.org/aboutJava/communityprocess/final/jsr172/.
7. 3GPP TS 22.057 V6.0.0. Mobile Execution Environment (MExE) service description; Stage 1, 2004, http://www.3gpp.org/ftp/Specs/html-info/22057.htm.
8. Attribute Programming, http://msdn2.microsoft.com/en-us/library/dcy94zz2.aspx.
9. M. Butler, F. Giannetti, R. Gimson, and T. Wiley, "Device independence and the Web," *IEEE Internet Computing*, Vol. 6, 2002, pp. 81-86.
10. W. Mueller, R. Schaefer, and S. Bleul, "Interactive multimodal user interfaces for mobile devices," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.
11. J. Plomp, R. Schaefer, W. Mueller, and H. Yli-Nikkola, "Comparing transcoding tools for use with a generic user interface format," in *Proceedings of the Extreme Markup Languages*, 2002.
12. J. Grundy and J. Hosking, "Developing adaptable user interfaces for component-based systems," in *Proceedings of the 1st Australian User Interface Conference*, 2002, pp. 175-194.
13. J2ME Polish, http://www.j2mepolish.org/.
14. T. H. Kao and S. M. Yuan, "Designing an XML-based context-aware transformation framework for mobile execution environments using CC/PP and XSLT," *Computer Standards & Interfaces*, Vol. 26, 2004, pp. 377-399.
15. T. H. Kao and S. M. Yuan, "Automatic adaptation of mobile applications to different user devices using modular mobile agents," *Software Practice and Experience*, Vol. 35, 2005, pp. 1349-1391.
16. J. Jing, A. S. Helal, and A. Elmagarmid, "Client-server computing in mobile environments," *ACM Computing Surveys*, Vol. 31, 1999, pp. 117-157.
17. JSR-118, Mobile Information Device Profile 2.0, http://jcp.org/aboutJava/communityprocess/final/jsr118/.
18. OSGi, http://www.osgi.org/.
19. J2SE – Java Standard Edition, http://java.sun.com/javase/.
20. M. C. Cheng and S. M. Yuan, "An adaptive mobile application development framework," in *Proceedings of the Embedded and Ubiquitous Computing*, 2005, pp. 765-774.
21. Y. S. Chang, R. S. Wu, K. C. Liang, S. M. Yuan, and M. Yang, "CODEX: content-oriented data EXchange model on CORBA," *Computer Standards & Interfaces*, Vol. 25, 2003, pp. 329-343.
22. S. Liang and G. Bracha, "Dynamic class loading in the Java virtual machine," in *Proceedings of the 13th ACM SIGPLAN conference on Object-Oriented Programming*, 1998, pp. 36-44.
23. G. T. Sullivan, "Aspect-oriented programming using reflection and metaobject protocols," *Communications of the ACM*, Vol. 44, 2001, pp. 95-97.
24. M. Dahm, "Byte code engineering with the BCEL API," Technical Report No. B-17-98, Freie Universitat Berlin, Institit fur Informatik, 2001.

25. Y. S. Chang, M. H. Ho, and S. M. Yuan, "A unified interface for integrating information retrieval," *Computer Standards & Interfaces*, Vol. 23, 2001, pp. 325-340.
26. XML – Extensible Markup Language, http://www.xml.it:23456/XML/REC-xml-19980210-it.html.
27. Web Services Description Language, http://www.w3.org/TR/wsdl.

Ming-Chun Cheng (鄭明俊) was born on September 4, 1977 in Taoyuan, Taiwan, R.O.C. He received his B.S. degree in Computer and Information Science from National Chiao Tung University, Taiwan, in 1999. Currently, he is a Ph.D. candidate in the Institute of Computer Science, National Chiao Tung University, Taiwan. His research interests include web technology, distributed system, and mobile computing.

Shyan-Ming Yuan (袁賢銘) was born on July 11, 1959 in Mauli, Taiwan, R.O.C. He received his B.S. degree in Electrical Engineering from National Taiwan University in 1981, his M.S. degree in Computer Science from University of Maryland, Baltimore County in 1985, and his Ph.D. degree in Computer Science from the University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he has been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include distributed objects, internet technologies, and software system Integration. Starting from Feb. 2007, Professor Yuan was temporarily on leave from National Chiao Tung University and became the Chairman of Computer Science & Information Engineering Department, Asia University.