An Adaptive Mobile Application Development Framework

Ming-Chun Cheng[1], Shyan-Ming Yuan [1]

[1] Department of Computer and Information Science, National Chiao Tung University,
1001 Ta Hsueh Rd, Hsinchu 300, Taiwan
{native, smyuan}@cis.nctu.edu.tw

Abstract. Although wireless networks and mobile devices become popular these days, the diversity of mobile devices and unsteadiness of wireless networks still cause software development much trouble. Thus, when developing a mobile application, developers are forced to expose to these problems and to be familiar with these technologies and therefore it will spend a lot of time to write a mobile application. Furthermore, a mobile application often needs to be ported to different platform (for example from Java to .NET) which is also burdensome. In order to overcome these problems, the author proposes an adaptive framework to help developers build mobile application effortlessly and rapidly. The mobile application developed on this framework can run in different devices and operating system, so developers does not need to worry about portability. As a result, a mobile application developed on this framework can enjoy the benefit of "write once, run everywhere, and access any services."

1 Introduction

In the past decade, the computing environment has changed tremendously. The bandwidth of wired network increases from 100M to Giga bps, and wireless networks become more and more popular. People often talk about mobile or pervasive computing, because mobile handheld devices get much more powerful and cheaper.

Accordingly, much ink has been spent on pervasive computing. In fact, it is not really so close to us. Owing to the size and battery consumption, processing powers of handheld devices are still insufficient to replace desktop computers do. In addition, the diversity of the computing environments is also a big trouble. Developers have to consider the differences among them and portability is an important issue. Thus, there are still considerable dilemmas needed to be overcome.

In order to solve the problems, the concept of virtual home environment (VHE) has been proposed by 3GPP. Although a large number of studies have been made on how to achieve VHE, such as MExE, OSA and USAT, little is known about providing an adaptive mobile application development framework by leveraging thin-client computing concept [1][2][3]. X window [17][20] and web technologies [4][5][6][7][8][9] are classic samples of thin-client computing. Nevertheless, the previous lacks support for handheld devices and the later lacks better interactions with end-users.

The aim of this paper is to design and implement a mobile application development framework capable of supporting heterogeneous environments, and the system is called ART. The main idea is to divide an application into two parts, UI and logic, in development time and run time. This idea makes development easier and adaptation possible. Hence, in this framework, the same application could be written once and used everywhere and developers have not to consider portability.

The rest of this paper is organized as following. In the next chapter, system architecture is introduced. Next, adaptation mechanisms are presented in chapter 3 and the workflow of the development process is illustrated in chapter 4. Finally, chapter 5 shows conclusions and future works.

2 System Architecture

First of all, we have to explain the scenario how end-users put our system to use. End-users might own not only desktop computers but also several mobile devices, such as WAP phone, Java-enabled handset, Smart phone, PDA and so on. When outdoors, they probably need some resources or continuing some programs on their desktops and might wish to control appliances, i.e. TV, air conditioner and light, through their mobile devices also. According to the previous scenario, the authors leverage a 3-tiers architecture to fit these requirements. It includes front-tier, middle-tier and backend-tier; it is represented diagrammatically in Fig.1.

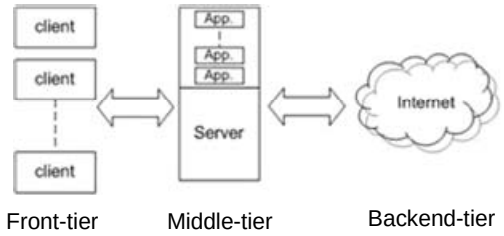

Fig. 1. The 3-tiers architecture

In this architecture, end-users use their own mobile devices or desktops in front-tier to access resources provided by applications run in middle-tier. In addition, the applications might access third-party services, e.g. X10, UPnP and web services [21], in backend-tier by calling some system services provided by server.

According the 3-tier architecture, ART system is designed as a client-server model, shown in Fig.2, and participates in the two tiers of the architecture, front-tier and middle-tier. Client run in front-tier is responsible for user interface as well as server run in middle-tier is responsible for logic computing. They communicate with each other by using asynchronous message delivery mechanism (the top part of the Fig.2) or other existent protocols (the bottom part of the Fig. 2), for example HTTP. In other words, this model could be viewed as a thin-client computing with adaptive capability. This section will present individual parts first and then introduce how they cooperate with each others. In order to simplify terminologies, in ART system, the

server is called ARTServer, the client is called ARTClient, and the application is called ARTApp.

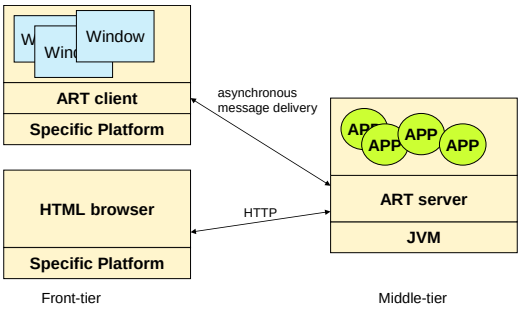

Fig. 2. The client-server model in ART system

2.1 Asynchronous Message Delivery Mechanism

Since wireless networks are not stable enough, the authors designed an asynchronous message delivery mechanism for transmission between ARTServer and ARTClient. This mechanism decouples the ARTApp and low-level network protocols, so it can help ARTServer to handle the situation of disconnection and prevents ARTApp from accessing network directly. Besides, this mechanism provides the communication capability among ARTApps also and it supports one-to-one, one-to-many and many-to-many modes. The four main components in this mechanism are ARTMesg, Queue, Message Dispatcher and Daemon Thread. ARTMesg is a message which carries some information about source, destination, command, request and user etc. For example, when user presses a button in client-side or ARTApp orders the client-side to create a new UI widget, the corresponding ARTMesg will be generated and then routed to correct place. Moreover, Queue is a data structure capable of storing and aggregating the ARTMesg. Finally, Message Dispatcher and Daemon Thread are both working threads which route or forward the ARTMesgs put in the Queue. The details will be described in section 2.4.

2.2 ARTApp

The applications which comply with ART Programming Framework run on ARTServer are called ARTApp and they might provide different services for ARTClient, e.g. search engine, appliance controller, or games. In development time, an ARTApp is separated into two parts, UI and logic, to cut down the development time. ART system leverages XUL to describe the user interface and exploits Java Language to write the logic part. The details about how to write an ARTApp is presented in Chapter 4.

In Runtime, every ARTApp owns an independent runtime space but they could communicate with others through asynchronous message delivery mechanism

mentioned previous. Furthermore, an ARTApp owns a Windows Manager (WinMngr) to maintain the relationship with its UI widgets described in XUL files.

In ART system, each ARTApp is encapsulated as a Java Jar file. When necessary, it is loaded dynamically by ARTServer and ARTServer will assign a thread to run it.

2.3 Backend-tier

There are many resources in internet or other networks, such as X10, UPnP and Jini. In ART system architecture, these resources are all in backend-tier. Currently, ART system only provides different APIs to access these various resources.

2.4 Middle-tier (ARTServer)

The ART system in this tier includes two components, ARTServer and ARTApp. ARTServer provides the runtime environment for ARTApp and ARTApp can use these services provided by ARTServer to access what they want. The architecture of ARTServer is showed in Fig. 3 and it could be considered as a layered architecture; they are Application Runtime Layer, Message Routing Layer and Adaptive Transport Layer. The authors hope ART system can be modularized very well and the layered design makes maintenance and upgrade easier.

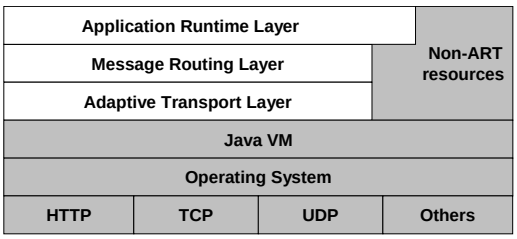

Fig. 3. The layered architecture of ARTServer

2.4.1 Adaptive Transport Layer

Adaptive Transport Layer gives ARTServer the ability to communicating with different kinds of ARTClients. Fig. 4 may help us to understand the detailed structure of ARTServer.

The main role of this layer is Communication Manager (CommMngr) which is a super daemon capable of handling many different protocols, such as TCP, UDP, HTTP and cHTTP, and it has two missions. First, it establishes the relationship between ARTServer and ARTClient when ARTClient sent the login request to ARTServer. Secondly, if login successful, CommMngr creates a logic process (i.e. a user process, including a UserOutD, a UserInD and a UserOutQ) for that ARTClient. Every logic process might have different components or functionalities depending on what kind of ARTClient it serves. UserOutD is responsible for picking ARTMesgs from UserOutQ, and sending them to the corresponding ARTClient or translating ARTMesgs to specific format. The UserInD handles or translates incoming requests

from its client and put them into InnerQueue. The details of adaptation mechanisms in this layer will be introduced in chapter 3.

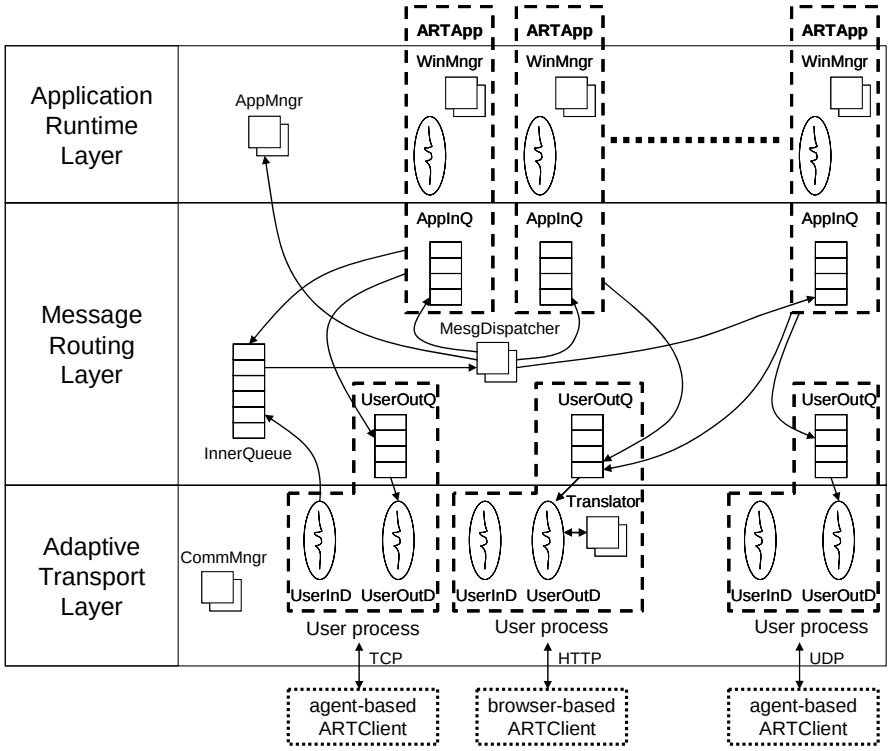

Fig. 4. The detailed architecture of ARTServer.

2.4.2 Message Routing Layer

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

All the messages received by UserInD are placed in InnerQueue. Then MesgDispatcher will dispatch them to the some AppInQ which ARTApp will handle these ARTMesg in.

Once an ARTApp generates an ARTMesg whose destination is an ARTClient, the message will be placed in UserOutQ of the user process that serves the client. UserOutD in this user process will send the message to its client later.

However, an ARTApp might need to negotiate with other ones on the same ARTServer; ARTMesgs in this situation will be put back to InnerQueue and wait for dispatching by MesgDispatcher again.

2.4.3 Application Runtime Layer

An ARTServer can serve many different ARTClients at the same time, and each ARTClient can also have many ARTApps run on the server at the same time. Application Manager (AppMngr) is responsible for loading, resuming and stopping ARTApps. Before starting an ARTApp, AppMngr will check if any instance of the ARTApp already exists in the memory. If it does, AppMngr will then check what running mode of the ARTApp is and decide to create a new instance or bind the client to the existent one.

2.5 Front-tier (ARTClient)

The front-tier part of ART system is called ARTClient which is responsible for presentation. It provides user interface to receive orders from end-users and displays results from middle-tier. In ART system, end-users can use many kinds of devices in front-tier and ART system could automatically tailor results to fit diverse environments. In these days, ART system can support four popular kinds of devices, including J2ME MIDP [12][13][14], .NET CF, WAP and HTML browser. Because of diversity of these devices, authors divide these devices into two categories, agent-based and browser-based, and use different mechanisms to design and implement them.

An ARTClient can access many ARTApp simultaneously, so ART system provides a facility, shown in Fig. 5, like Task Manager within Microsoft Windows XP. It helps end-users to select what ARTApp to start, stop or switch to.

Fig. 5. The screenshot of the task manager within ART system.

2.5.1 Agent-based client

An agent-based client is defined as a device capable of executing author-developed ARTClient on it. According the definition, it has to provide some programming environments for programmers to develop some programs; nowadays, the most popular programming environments of handheld devices are J2ME MIDP and .NET CF. In order to support both them, two different agent-based clients are implemented and they are all compliant with “ART agent-based client specification”.

2.5.2 Browser-based client

A browser-based client is defined as a device which is browser enabled. In other words, end-user could use these existent browsers to access many sorts of services in

internet. Currently, almost mobile devices are browser enabled. Moreover, owing to adaptive transport layer, ART system can be accessed by these clients.

3 Adaptation Mechanisms

In ART system, adaptation is applied in two positions, network protocols and UI, and both two are implemented in Adaptive Transport Layer. This chapter will introduce how adaptation mechanisms work.

3.1 Network protocols

Because of diversity of mobile devices, the networks capabilities are not the same. Some devices support TCP connection but some devices might support only HTTP. In order to hide the detailed from mobile application developers, the ARTServer exploits a loosely couple design and the adaptive transport layer is proposed. The layer defines the unified transport interface and the implementation which complies with it could be plugged into ART system easily. At the moment, ART system supports TCP, UDP and HTTP.

3.2 User Interface

In ART system, ARTApp doesn't know what kind of client it servers and it only handles and generates ARTMesgs no matter the type of client. Hence, in order to serve browser-based clients, ART system needs a mechanism to translate ARTMesgs to other formats, such as WML or HTML, and this mechanism is implemented in adaptive transport layer also.

In the section 2.4.1, the CommMngr is introduced and one mission of it is to create a user process for a client. When a client sends login request to ARTServer, CommMngr will analyze CC/PP [11] profile provided by the client. If the client is browser-based, a translator component is within the user process and it is combined with many convert functions and a layout manager.

Every convert function has different capability to translate a XUL [10] widget to the corresponding widget of other formats. Table 1 is a widget mapping among XUL, MIDP and WML.

Table 1. The mapping table among XUL, MIDP and WML

XUL	**MIDP**	**WML**
<window>	Form	<wml>+<card>
<canvas>	Canvas	<wml>+<card>+4 direction button+<img src=””>
<listbox>	ChoiceGroup	<select>
<button>	Buttom	<a href>
<textbox>	TextField	<input>

The layout manager is similar to WinMngr within agent-based ARTClient but it is capable of arranging the positions of the widgets. Another functionality of layout manager is to call convert functions to translate XUL widgets to specific formats. When serving a browser-based client, ARTMesg does not send to client directly instead of being handled by the layout manager.

4 Programming Framework

This chapter shows how to write an ARTApp. An ARTApp is combined with two different parts, UI and logic, and these two parts could be developed simultaneously by different developers. The development workflow is showed in Fig. 6.

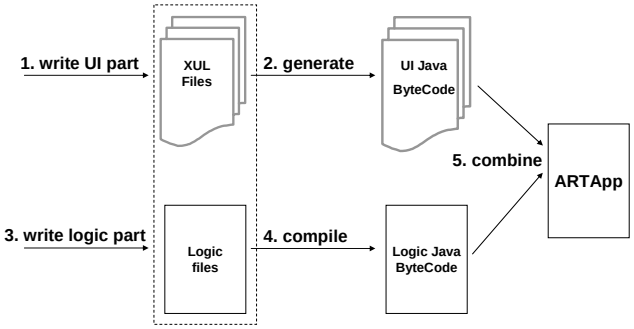

Fig. 6. The application developers have to writes two kinds of files within the dashed box.

4.1 Write User Interface Part

In ART system, developers have to use XUL to describe the user interface. Fig. 7 shows a sample; left is a XUL file and right is the presentation result.

When finishing the XUL files, developers use code generator provided by ART system to generate some Java Bytecode and these codes can be used by logic part

```
<?xml version="1.0"?>
<window id="Example">
        <listbox id="CHOICE" label="Choose one">
                <listitem id="1" label="1"/>
                <listitem id="2" label="2"/>
        </listbox>
      <textbox id="INPUT" label="Input Here"/>
      <button id="BUTTON" label="OK"/>
</window>
```

Fig. 7. A sample XUL file and the presentation result.

4.2 Write Logic Part

In ART system, Java language is used to write logic part and developers who are familiar with Java have not to learn anything. The only requirement is that the main class of application has to extend the ARTApp class. Fig. 8 shows an example.

```
public class Main extends ARTApp{
    Example window=null;
    public void startApp(){
        //get the reference of the window named "Example"
        window=Example.getInstance();

        //create a listener
        class Listener implements ArtButtonListener{
            public Listener(){}
            public void actionPerform(){
                //change the text of the button named "BUTTON"
                window.BUTTON.setText("Hello");
            }
        }

        //register a listener to the button named "BUTTON"
        window.BUTTON.setArtButtonListener(new Listener());
        //bring the window named "Example" to foreground
        window.show();
    }
    public void stopApp(){}
}
```

Fig. 8. A logic codes written by Java language.

5 Conclusions and Future Works

So far, we have described our system, ART system, and how it achieves the objectives mentioned in chapter 1. The key point of design is separating GUI from the application body. Developers write UI descriptions in XUL instead of writing them in program bodies. Then ART will generate the UI objects automatically according to XUL documents. Users can launch numerous applications via a small-size client on mobile devices and ART system will automatically tailor the result to fit different clients.

In the future, the authors hope the agent-based clients could be more powerful. The idea is that ART system can send some scripts to ARTClient and some logics could be run in client-side. This method makes the ART system more flexible and more efficient.

Another issue is how to integrate the services in backend-tier. Currently, the authors are designing a unified interface capable of accessing Jini, UPnP and X10 by a unified method.

References

1. Krikke, J., “Thin clients get second chance in emerging markets”, Pervasive Computing, IEEE, Volume 3, Issue 4, Oct-Dec 2004, pp. 6-10
2. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A., “Virtual network computing”, Internet Computing, IEEE, Volume 2, Issue 1, Jan.-Feb. 1998, pp. 33-38
3. Volchkov, A., “Server-based computing opportunities”, IT Professional, Volume 1, Issue 2, March-April 2002, pp. 18-23
4. L. C. Martins, “A Framework for Filtering and Packaging Hypermedia Documents”, Adaptive Hypermedia and Adaptive Web-Based Systems Second International Conference, Malaga Spin, May 2002, pp. 274-283
5. Ricky Robinson., “Context Management in Mobile Environments”, World Wide Web, http://www.itee.uq.edu.au/~ricky/thesis.doc, October 2000.
6. Mark H. Bulter., “Current Technologies for Device Independence”, World Wide Web, http:// www-uk.hpl.hp.com/people/marbut/currTechDevInd.htm, March 2001.
7. Mark Bulter, Fabio giannetti, roger gimson, and Tony wiley, “Device Independence and the Web. IEEE internet computing”, October 2002.
8. W3C, “Device independence working group charter”, World Wide Web, http://www. w3.org/2002/06 /w3c-di-wg-charter-20020612.html .
9. E. Kirda, C. Kerer, “Supporting multi-device enabled Web services: challenges and open problems”, 10th IEEE Intl. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Cambridge MA, USA, June 2001, pp. 49-54
10. eXtensible User Interface Language, http://www.xulplanet.com/tutorials/xultu/.
11. W3C, “Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation”, http://www.w3.org/TR/NOTE-CCPP/.
12. Sun Microsystems, “Java 2 Platform Technologies”, http://java.sun.com/.
13. Sun Microsystems, “Java 2 Platform, Micro Edition (J2ME)”, http://java.sun.com/j2me/.
14. Sun Microsystems, “Mobile Information Device Profile (MIDP)”, http://java.sun.com/products/cldc/.
15. Sun Microsystems, Java Technology and Web Services, http://java.sun.com/webservices/.
16. 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Virtual Home Environment / Open Service Architecture (3G TS 23.127 version 1.10.0).
17. X.Org, “X11 technologies”, http://www.x.org/X11.html
18. P. de Bra, et al., Adaptive Hypermedia and Adaptive Web-Based Systems, Springer, May 2002.
19. User Interface Markup Language (UIML), http://www.uiml.org/index.php.
20. R. W. Scheifler, “X Window System Protocol, X Consortium Standard, X Version 11, Release 6”, Massachusetts Institute of Technology, Laboratory for Computer Science, 1994
21. W3C, Web Service Architecture, http://www.w3.org/TR/2003/WD-ws-arch-20030514/.