(1) Name of the workshop: Computer Networks

(2) Title of the paper: A Mobile Service Environment for Handheld Devices Using XML-RPC

(3) A short abstract:

In this paper, we design an integrated mobile service environment (MSE) for the handheld devices (e.g., PDAs). The integral MSE is a coherent and intact service architecture built on the front-end (handheld devices), which allows the mobile users to choose and subscribe their request services provided by the service provider. A real-time stock transaction system is implemented to experiment the feasibility and applicability of this MSE architecture.

(4) Name: Tsung-Chuan Huang
Current affiliation: Dept. of Electrical Engineering, National Sun Yat-Sen Univ., Kaohsiung.
Postal address: 70 Lien-Hai Rd., Dept. of Electrical Engineering, National Sun Yat-Sen Univ., Kaohsiung.
E-mail address: tch@mail.nsysu.edu.tw
Telephone number: (07) 5254140
Fax number: (07) 5254199 (Tsung-Chuan Huang)
Name of the contact author: Tsung-Chuan Huang

(5) A list of keywords indicating the content areas related to the paper:

Mobile service, XML-RPC, Service Agent, *Cache* mechanism.

A Mobile Service Environment for Handheld Devices Using XML-RPC

Tsung-Chuan Huang, Chu-Sing Yang**, Sheng-Wen Bai**, and Kuo-Chang Lin**

* Department of Electrical Engineering
** Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung 804, Taiwan
* tch@mail.nsysu.edu.tw, c8931313@ee.nsysu.edu.tw
** csyang, swbai@cse.nsysu.edu.tw

Abstract. In this paper, we design an integrated mobile service environment (MSE) for the handheld devices (e.g., PDAs). The integral MSE is a coherent and intact service architecture built on the front-end (handheld devices), which allows the mobile users to choose and subscribe their request services provided by the Service Provider.

The architectures of MSE consist of desktop agents (DA) and service agents (SA) at the top layer, a communication layer at the bottom and a virtual machine (VM) layer between them. The desktop agent is used as user interface and is responsible for managing local process; the service agent is downloaded from the back-end server when certain service is requested. The communication layer adopts XML-RPC as the communication protocol between the client and back-end server. In order to shorten the transmission time the data size of XML-RPC is reduced using our compression mechanism. The virtual machine supports cross-platform operation for the top layer. Besides, we devise a *cache* mechanism to retain recently used and frequently reused SAs in the limited memory space of handheld devices. This *cache* mechanism permits the mobile users to achieve the SA they want more efficiently.

Finally, we implement a real-time stock transaction system to experiment the feasibility and applicability of this MSE architecture.

Keywords: Mobile service, XML-RPC, Service Agent, *Cache* mechanism.

1. Introduction

Now-a-days, mobile devices in the form of laptop and personal digital assistant (PDA) have become widespread tools to provide data access for users on the move. With current trend, the number of wireless device subscribers will continue to grow. Mobile users, like the wireless device subscribers, hope to obtain the mobile services provided by the service providers according to their demand. But at present, most of the services are developed on standalone or are closed applications, which lack integrated service mechanism. In this paper, we develop a coherent and intact service environment to allow the mobile users to subscribe infinite services provided by the service providers. Mobile users can subscribe the services based on their need, download and install the service agents (SAs) on their handheld devices directly. On the other hand, due to the limited resource of the handheld devices, some issues must be considered when the services are executed:

- **Cross-platform -** Problems may arise when a service is to run on several different platforms (e.g., Win32^{TM}, Palm OS ^{TM}, Windows CE ^{TM}, etc.).
- **Computing power -** The processor of handheld devices is not as powerful as desktop PC. It can not perform complex arithmetic operations or large amount of computations.
- **Storage space -** Handheld devices can not store infinite services provided by the service providers due to limited storage space.

Concerning these issues, we propose an operating environment, called MSE (Mobile Service Environment), for the handheld devices. MSE supports cross-platform, powerful computation, and fast communication with the back-end server. To achieve cross-platform, we build a virtual machine (VM) under the application layer (the virtual machine will be explained later). To overcome the problem of poor computing power, we forward advanced arithmetic or large amount of computations to the back-end server and the results are returned using a fast communication interface. To deal with the problem of limited storage space, all the services provided by the service providers are stored in the back-end server. The corresponding service agent is downloaded from the server only on demand. To increase the access efficiency, we devise a *cache* mechanism in the handheld device to maintain the recently used and frequently reused services. Besides, in order to communicate with the back-end server efficiently, we use standard protocol XML-RPC (Extensible Markup Language [15]- Remote Procedure Call) [20] as the communication mechanism. In order to reduce the traffic between the client and server, we design a compression scheme, which encode the data format of XML-RPC to decrease the data size and speed up the communication as well.

The remainder of this paper is organized as follows: Section 2 introduces the related work. Section 3 describes the architecture of our mobile service environment. Section 4 explains our implementation. Section 5 demonstrates sample presentations. Finally, Section 6 concludes this paper.

2. Related Work

Currently, a variety of mobile information services have been developed for the mobile users. Most of these information services are browser oriented [2]; that is, they download the information pages for browsing on the handheld devices. In addition to browsing the web pages, some information services can download standalone programs on the handheld devices and execute them. Table 1 lists a number of such products in the current market.

Table 1: A list of service programs in the market.

NO	Applications	Specifications
1	WAPman 1.8[19]	WAP Browser.
2	YesPalm 1.1[21]	WAP Browser.
3	AvantGo.com 4.0 [3]	HTML&WAP Browser.
4	Blazer 2.0 [9]	HTML&WAP Browser, Compress Web Pages.
5	Palmscape 3.1.3[13]	HTML&WAP Browser, Cache saves pages.
6	Xiino 1.0.5[13]	HTML&WAP Browser, Download Program.
7	4thpass Kbrowser [1]	WAP Browser, Download Program.
8	Wapaka 2.0 [18]	HTML&WAP Browser, Download Program.
9	CAC 1.0[4]	HTML & WAP Browser, Compress Web Pages, Download Program, Cache saves pages.

Besides the commercial products, there are several systems which are developed using WAP interface in the literatures. InfoParco system developed by Colafigli *et al.* [6] supports handheld devices to access the information of public use, accommodations, and weather forecast from a sophisticated web for touristic purposes. Maclean and Dailey [11] reported a real-time transit vehicle information system, which delivers content to Internet-enabled mobile devices by WAP interface. The content is in the form of predicted arrival/departure times for buses at user-selectable geographic locations within a transit region. The wireless mobile monitoring (WMM) method proposed by Shan and Li [14] provides an effective round-the-clock network monitoring and failure warning to the wireless network users. Besides, Garmash [8] designed a mobile location-based application by using an XML-based description in a mobile environment. And in [10], Lan *et al.* proposed a mobile e-Commerce solution to do mobile banking, mobile shopping and even access data on the web via their mobile devices. They also use XML-based data model language for information retrieval.

The works mentioned above are all through the WAP interfaces to access information. They simply employed WAP interface to develop a variety of applications. In this paper, we aim at developing an integrated mobile service environment to access infinite subscription services from the service providers on the handheld devices. This environment uses the POST method of WAP as the communication protocol to attach XML-RPC message, so as to access mass information services quickly from the service providers on the handheld devices and have the advantages of cross-platform and thin-client.

3. Architecture of our Mobile Service Environment

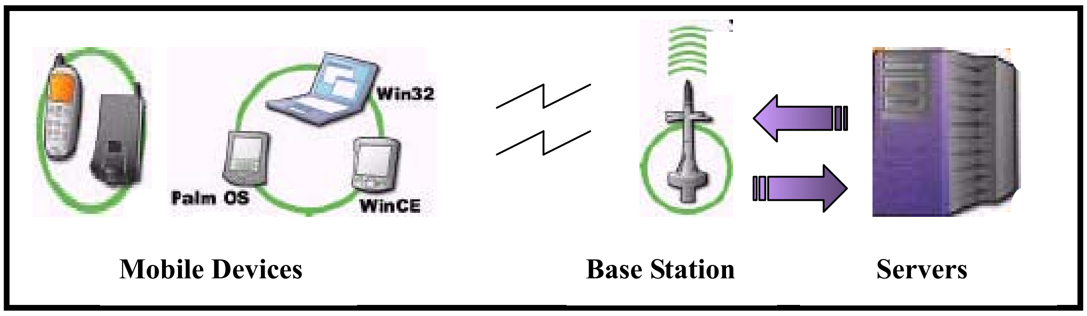


Fig. 1. The mobile communication system.

Fig. 1 shows a simple architecture of mobile communication system. Clients are the front-end mobile devices. Servers, including gateway and service provider, are in the back-end for providing services to the clients. Clients can execute the procedure call and link to the server for data query, web pages browsing, file downloading, etc.

Fig. 2 illustrates the architecture of MSE in mobile clients and shows the framework of each layer in the environment. There are two main components in the top layer: desktop agent (DA) and service agent (SA). DA is loaded by the virtual machine (VM) from server to handheld device once the handheld device is booted up. It is used as the user interface and is responsible for local process management. SA is provided by the service provider. When service provider releases a new service, the associated SA must be provided simultaneously. When a user subscribes a service, the corresponding SA is downloaded to the handheld device. In order to reduce the network traffic and the loading time of SA, we develop a *cache* mechanism to keep the most recently used and frequently reused SA in the mobile handheld device so as to reduce the downloading time and enhance the operation efficiency. The synchronization problem of *cache* between the client and server is also taken into account in our MSE. The *cache* mechanism will be described later.

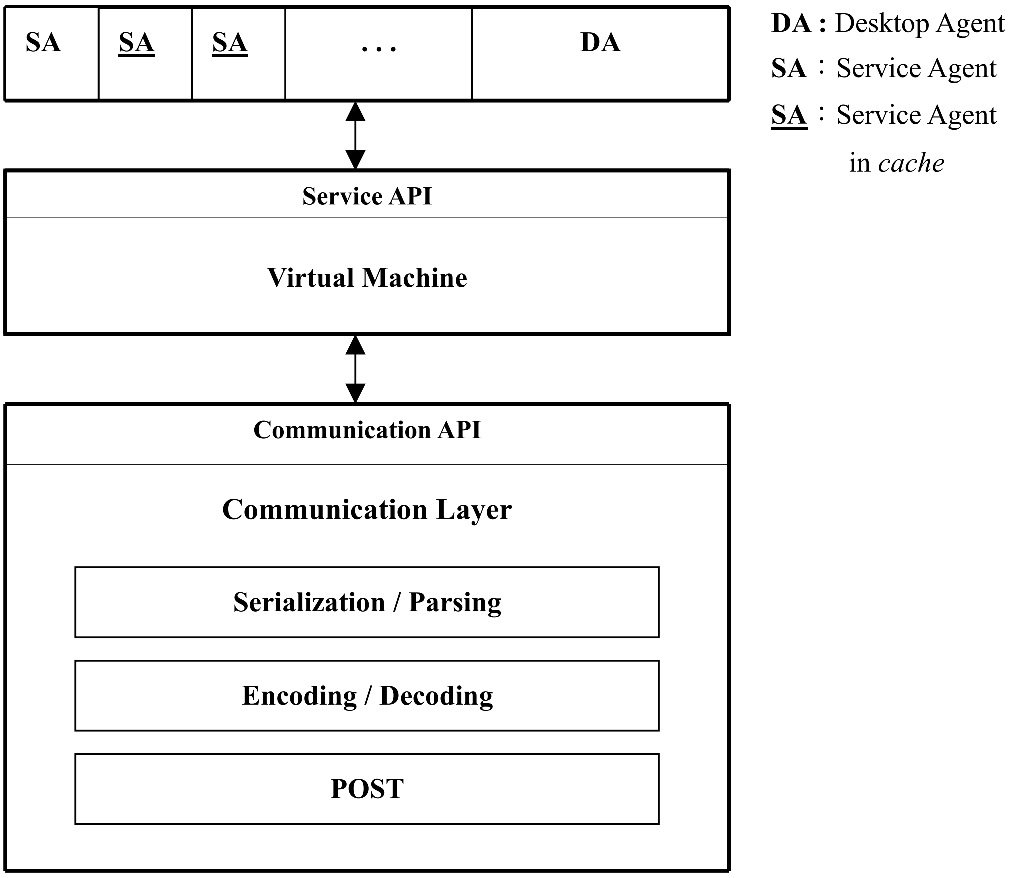


Fig. 2. Architecture of MSE.

Between DA/SA and virtual machine (VM) there is a service application interface (API). Virtual machine (VM) layer contains a service API responsible for the communication to DA/SA, whose data structures are shown in Fig. 3. An important task of virtual machine is to load DA from server in the boot-up sequence. VM supports cross-platform, allowing SAs to execute on different operation systems (e.g., Win32^{TM}, Palm OS^{TM}, Windows CE^{TM}, etc.).

```
public class Rpcaddr
{
  private String url;
  private String gateway;
}
Rpcaddr rpca= new Rpcaddr( );
public String xmlrpc_client_post
( rpca.url,rpca.gateway,String method_name,
  String format, ... )
```

Fig. 3. Data structures of service API.

Communication layer is the communication mechanism in the lowest layer. Within it, there is a communication API responsible for communicating with VM, whose data structures are shown in Fig. 4. The communication layer receives procedure calls from upper layers and POST message to the server. POST uses standard remote procedure call, XML-RPC, as the communication mechanism to provide information services between the client and server. There are three major components in the communication layer: Serialization/Parsing, Encoding/Decoding, and POST. Serialization packs the information into XML-RPC format; parsing extracts the information from XML-RPC format. Encoding compresses the XML-RPC packet; decoding decompresses the XML-RPC packet. The methods of compression and decompression will be introduced in Section 3.2. The POST component is responsible for sending the XML-RPC packet to the server and receiving the result from it.

```
Struct Rpcaddr
{
  char *url;
  char *gateway;
};
UInt32  xmlrpc_client_post
( UInt16 refNum,char *result[],
  Rpcaddr.url,Rpcaddr.gateway,
  char *method name,char *format, ... )
```

Fig. 4. Data structures of communication API.

Fig. 5 illustrates the communication procedure between the client and server. The architecture of client is as shown in Fig. 2. The server consists of XML-RPC Encoder/Decoder, XML-RPC Server and Service Agent Library. The seven steps of communication procedure are explained as follows:

Step 1: Client chooses URL and issues a request to download DA from the server.

Step 2: Client issues the request of search service; the server returns the service list.

Step 3: Client subscribes a service; the server returns the result (succeed or fail).

Step 4: Client requests to list the services that have been subscribed; the server returns the service list.

Step 5: User chooses a service. If the corresponding service agent is not in *cache*, the client will send a request to downloaded the SA from the server. Otherwise, the client launches the SA directly from *cache*.

Step 6: SA sends XML-RPC procedure call to the server.

Step 7: Server responds the result.

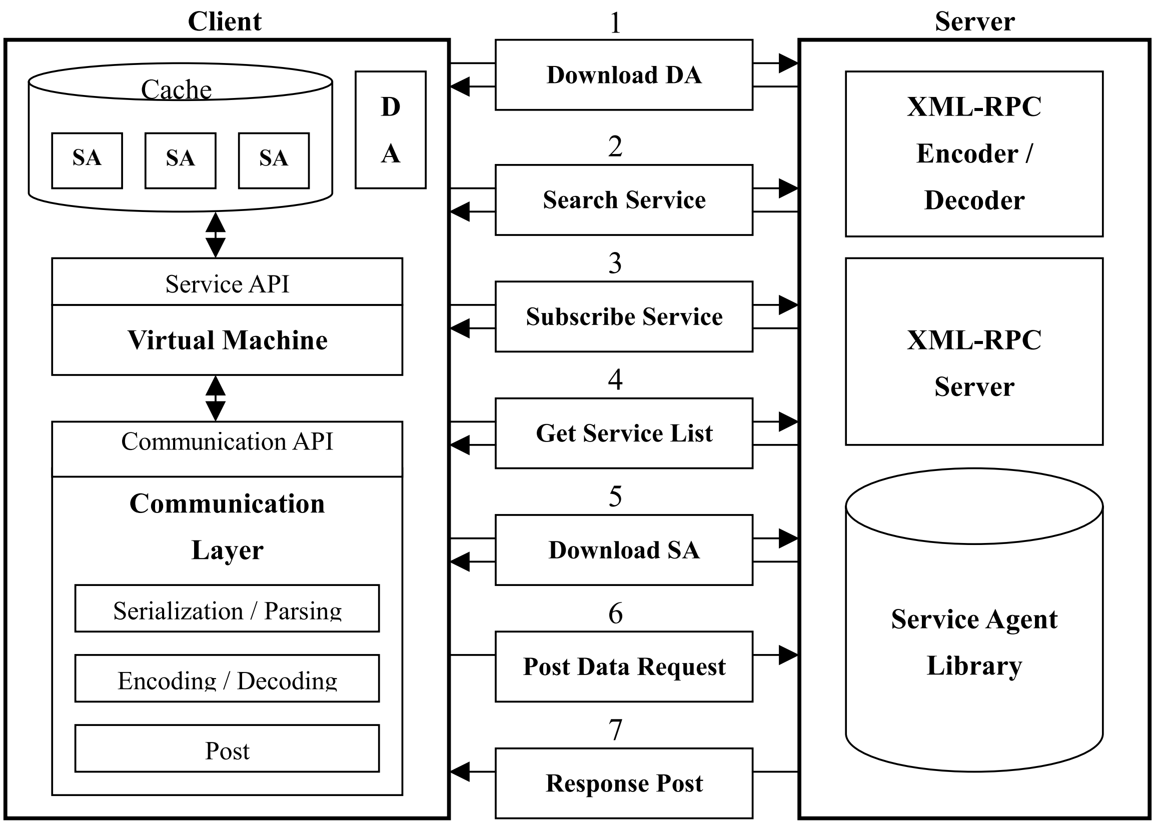


Fig. 5. Communication procedure between the client and server.

3.1 XML-RPC POST

XML-RPC is a mechanism of standard remote procedure call, which uses XML, a common readable language with infinite number of formats, to exchange message between the client and server by HTTP [15]. By standardized cross-platform approach, XML-RPC can make procedure call in many different operating platforms over the Internet like Fig. 6. The data format is shown in Fig. 7.

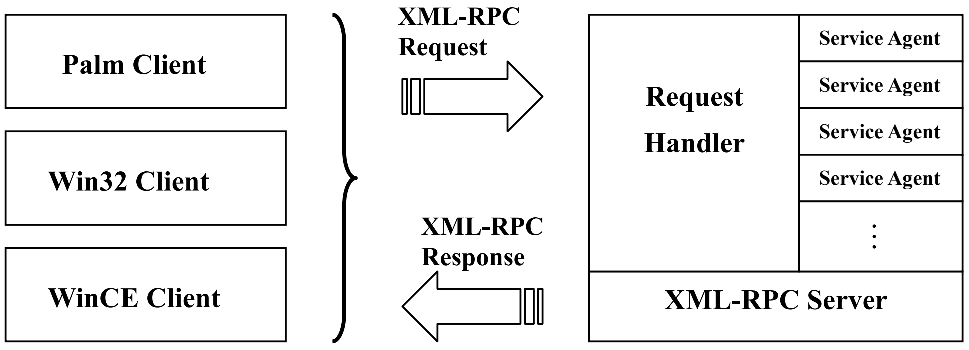


Fig. 6. XML-RPC Communication Architecture.

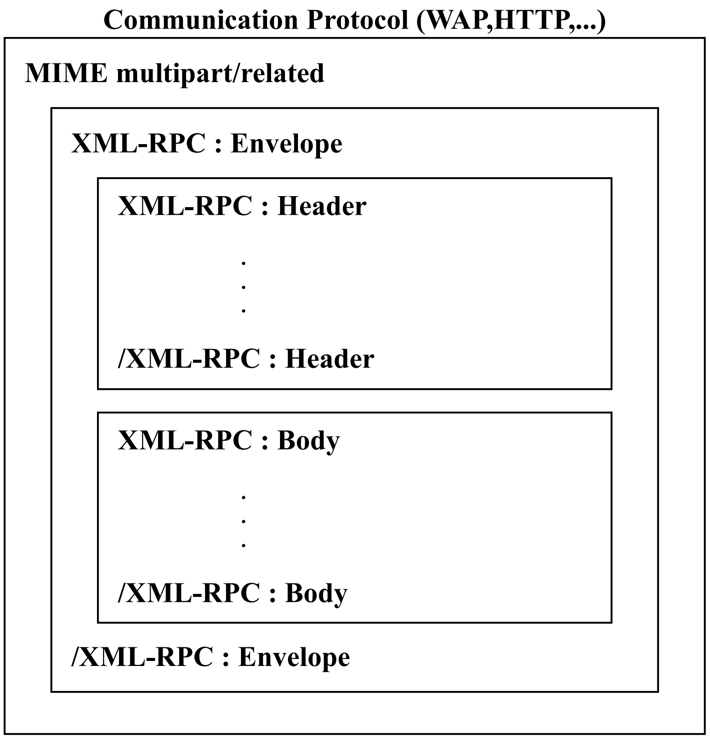


Fig. 7. XML-RPC Format.

The follows is a simple example of XML-RPC request / response :

```
1. Request
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181
<?xml version="1.0"?>
  <methodCall>
    <methodName>
      examples.getStateName
    </methodName>
    <params>
      <param>
        <value><i4>41</i4></value>
      </param>
    </params>
  </methodCall>
```

```
2. Response
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT
<?xml version="1.0"?>
  <methodResponse>
   <params>
    <param>
     <value>
        <string>South Dakota</string>
      </value>
    </param>
   </params>
  </methodResponse>
```

If the procedure call has parameters, the <methodCall> must contain a <params> sub-item. The <params> sub-item can contain any number of <param>s, each of which has a <value>. <value>s can be scalars whose type is indicated by nesting the value inside one of the tags listed in Table 2. A value can also be of type <struct> or <array> (the interested readers please refer to [20]).

Table 2: Value list.

Tag	**Type**	**Example**
<i4> or <int>	four-byte signed integer	-12
<boolean>	0 (false) or 1 (true)	1
<string>	ASCII string	hello world
<double>	double-precision signed floating point number	-12.214
<dateTime.iso8601>	date/time	19980717T14:08:55
<base64>	base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

3.2 Data Encoding and Decoding

In order to reduce the traffic between the client and server, we propose a scheme of data encoding and decoding (Fig. 8). Encoding is employed before sending a message and decoding is performed after a message is received. In encoding phase, each tag of XML-RPC is converted into a binary code according to the transformation table shown in Table 3. In addition, the SA to be downloaded will be encoded into Base64 format. In the decoding phase, the above procedure is reversed.

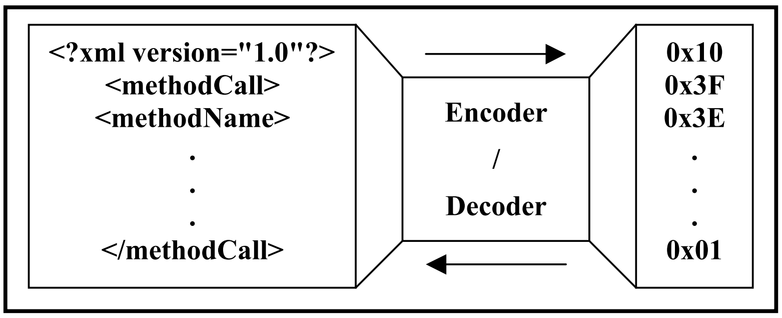


Fig. 8. Data encoding and decoding.

Fig. 9 shows an example of XML-RPC, in which the original message in Base64 format is "my password" (the encoding details is omitted due to limited space). After our encoding scheme, the result is shown in Fig. 10. Note that the initial tag <?xml version="1.0"?> has been encoded into 0x10. The original number of bytes in Fig. 9 is 209, which has been reduced to 48 bytes in total in Fig. 10. Large amount of transmission data was saved.

Table 3: Transformation table of tags in XML-RPC.

Tag	Encoding Code	Tag	Encoding Code	Tag	Encoding Code	Tag	Encoding Code
methodCall	0x3F	int	0x2E	faultString	0x1E	/i4	0x01
methodName	0x3E	boolean	0x2D	/methodCall	0x01	/int	0x01
params	0x3D	string	0x2C	/methodName	0x01	/boolean	0x01
param	0x3C	double	0x2B	/params	0x01	/string	0x01
value	0x3B	dateTime. iso8601	0x2A	/param	0x01	/double	0x01
struct	0x3A	base64	0x29	/value	0x01	/dateTime. iso8601	0x01
array	0x39	member	0x28	/struct	0x01	/base64	0x01
fault	0x38	name	0x27	/array	0x01	/member	0x01
method Response	0x37	data	0x26	/fault	0x01	/name	0x01
i4	0x2F	faultCode	0x1F	/method Response	0x01	/data	0x01

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>Service.Agent</name>
            <value><base64>bXlwYXNzd29yZA==</base64></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

Fig. 9. An example of XML-RPC (209 bytes in total).

```
10 37 3D 3C 3B 3A 28 27 53 65 72 76 69 63 65 2E 41 67 65 6E 74 01 3B 29 62 58 6C 77 59 58 4E 7A 64 32 39
79 5A 41 3D 3D 01 01 01 01 01 01 01 01
```

Fig. 10. The result of Fig. 9 after encoding (48 bytes in total).

3.3 *Cache* mechanism

As mentioned before, when we want to execute a service, whose SA will be downloaded from the server. It is not efficient and time-consuming if the SA is highly reused. To deal with this problem, we devise a *cache* mechanism in the client. Recently used and frequently reused SAs are kept in *cache* memory, so that the users can obtain the services they want quickly.

To achieve this, two integer values: RU (recently used) and FR (frequently reused), are associated with each SA in the *cache.* The RU of the first SA in *cache* is set to 1, and whose FR is set to 0. Hereafter, the RU of newly coming SA will be set to one plus the largest RU in the *cache*. In other words, the RU of currently used SA always is the largest among all the SAs in *cache.* When a new SA is loaded into *cache*, its FR is initialized to 0. The value of FR will increase one if the SA in the *cache* is reused once. If the SA of current service is not in *cache* and the *cache* has no enough space to accommodate this SA, the SA in *cache* with the smallest sum of RU and FR will be swapped out. Besides, if the SA in *cache* is an old version we must download the new version from server to replace its old copy in the *cache*. The detailed algorithm of this mechanism is presented in Fig. 11.

Step 1. If *cache* is empty, download the required SA from server. Set its RU to 1, FR to 0 and exit.

Step 2. If the required SA is not found in *cache*, go to Step 6.

Step 3. If the required SA in *cache* is not an old version, set its RU to the largest RU in *cache* plus 1 and increase its FR by 1. Exit.

Step 4. Remove the SA of old version from *cache*. If the free space of *cache* can not accommodate the SA of new version, remove the SA (may be more than one) in *cache* that has the smallest sum of RU and FR until the free space can accommodate the new version. If the RU of removed SA is the smallest one in *cache*, subtract it from each RU of SA.

Step 5. Download the SA of new version from server. Set its RU to the largest RU in *cache* plus 1. Set its FR to that of old version plus 1. Exit.

Step 6. Remove the SA (may be more than one) in *cache* that has the smallest sum of RU and FR until the free space of *cache* can accommodate the required SA. If the RU of removed SA is the smallest one in *cache*, subtract it from each RU of SA.

Step 7. Download the required SA from server. Set its RU to the largest RU in *cache* plus 1. Set its FR to 0. Exit.

Fig. 11. The algorithm of our cache mechanism in loading a service agent.

Fig. 12 is an example of our *cache* mechanism. Assume that four service agents S1, S2, S3, S4 resided in the *cache* in sequence and that none of them has been reused. Hence their RUs and FRs are as those shown in Fig. 12(a): S4 has the largest RU because it is the latest used service; all initial values of FR are set to zero. Some situations should be discussed:

Case (1): *If S5 (25k memory space assumed) is to be loaded into cache but the cache has no enough space* *to accommodate it.* In this situation, the SA with the smallest sum of RU and FR will be swapped out until the free space of *cache* is sufficient to accommodate S5. In this example, only S1 will be swapped out (if S5 is 35k, S2 must be removed also). Since S1 is the one *whose RU is the smallest*, its RU value will be subtracted from each RU of SA in the *cache.* The purpose is to avoid the RU value growing infinite. The result is shown in Fig. 12(b). After S5 is loaded into *cache*, the result becomes Fig. 12(c). Notice that S5 has the largest RU.

Case (2): *If S3 in Fig. 12(a) is reused.* The RU of S3 will become the largest one in *cache*, and its FR is increased. The result is shown in Fig. 12(d). If S3 is an old version, the new version will be downloaded from server.

Case (3): *Suppose that the cache configuration is now shown in Fig. 12(e) but not Fig. 12(a) and case (1) happens.* S2 will be swapped out because it has the smallest sum of RU and FR. But since the RU of S2 is not the smallest one in *cache*, it is not subtracted from the RUs of other SAs. The result becomes Fig. 12(f) after S5 is loaded.

SA	RU	FR	Size
S1	1	0	10k
S2	2	0	35k
S3	3	0	15k
S4	4	0	20k

Remaining space of cache: 20k

(a)

SA	RU	FR	Size
S2	1	0	35k
S3	2	0	15k
S4	3	0	20k

Remaining space of cache: 30k

(b)

SA	RU	FR	Size
S2	1	0	35k
S3	2	0	15k
S4	3	0	20k
S5	4	0	25k

Remaining space of cache: 5k

(c)

SA	RU	FR	Size
S1	1	0	10k
S2	2	0	35k
S3	5	1	15k
S4	4	0	20k

Remaining space of cache: 20k

(d)

SA	RU	FR	Size
S1	14	11	10k
S2	15	2	35k
S3	16	2	15k
S4	17	2	20k

Remaining space of cache: 20k

(e)

SA	RU	FR	Size
S1	14	11	10k
S3	16	2	15k
S4	17	2	20k
S5	18	0	25k

Remaining space of cache: 30k

(f)

Fig. 12. An example of our cache mechanism.

4. Implementation

We implement our mobile service environment (MSE) by the following development toolkits:

a. Waba SDK 1.12 and WabaVM 1.0 [16]

b. Code Warrior™ 7.0 [5]

c. Falch.net DeveloperStudio 2.6 for Palm OS [7]

d. Palm OS™ Emulator 3.3 and SDK 4 [12]

Waba SDK 1.12 and its class files are used to implement the desktop agent and service agent. The virtual machine in our MSE is revised from WabaVM 1.0 using Code Warrior™ 7.0. Falch.net DeveloperStudio 2.6 is employed to build the communication mechanism of XML-RPC in the lowest layer. All the MSE applications are executed on the Palm OS emulator in this experiment.

The reason of using Waba as our implementation language is that Waba is a free open source programming platform for small and mobile devices. With Waba, you can write one program that can run on a Palm™ Pilot device, Windows CE™ device and on any machine that supports Java™ (either the JDK 1.02, 1.1, 1.2 or 2.0). Waba virtual machines are available that are under 64K in size (including foundation classes) and that run programs in less than 10K of memory. With a native Waba virtual machine, the same program could run on a small device, such as the PalmPilot. Fig. 13 illustrates the entire architecture of Waba procedure call. A series of procedure calls using native method of classes can be built between Waba program and WabaVM to access the functions we defined. WabaVM invokes procedure call to the communication layer and then returns the response message to the Waba program.

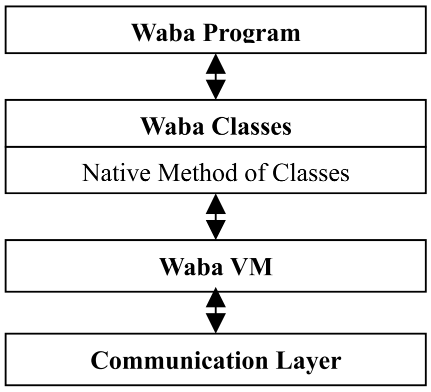
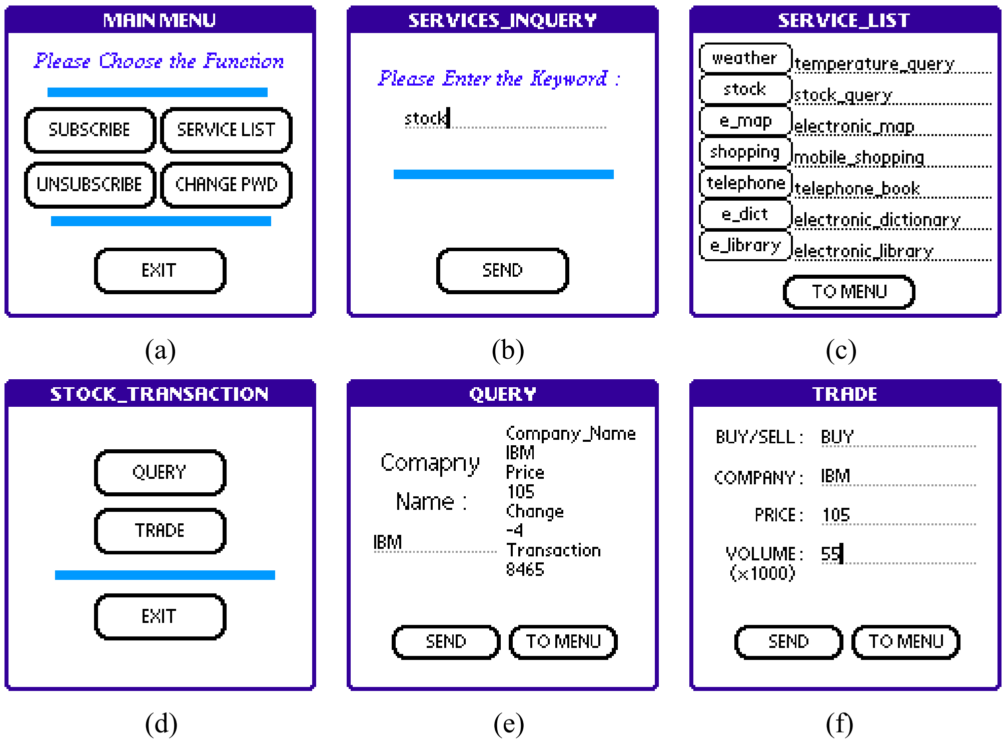


Fig. 13. The Waba procedure call.

5. Presentation

The purpose of this section is to demonstrate our implementation results. Fig. 14 shows samples of our system operation. Fig. 14(a) presents the main menu of MSE, which includes four functions: subscribing service, unsubscribing service, showing service list and changing password. “Subscribe” and “Unsubscribe” allow mobile users to subscribe or unsubscribe a service. When the mobile user wishes to subscribe a service, he can inquire the related services by entering a keyword. Fig. 14(b) shows this case. “Service list” lists all of the services that the mobile user has subscribed as Fig. 14(c) shows. Then, the mobile user can perform the service he chose. Fig. 14(d) shows the picture when “Stock” is the selected service. The functions of stock transaction include “Query” and “Trade”. Both of them are real time. The user can use “Query” to inquire the current transaction information of corporations in the stock market (Fig. 14(e)). If he wishes to make a trade, “Trade” allows him to buy or sell a stock (Fig. 14(f)).



(b) Please Enter the Keyword : stock

(c)

weather	temperature_query
stock	stock_query
e_map	electronic_map
shopping	mobile_shopping
telephone	telephone_book
e_dict	electronic_dictionary
e_library	electronic_library

(e) Comapny Name : IBM

Company_Name
IBM
Price
105
Change
-4
Transaction
8465

(f)

BUY/SELL :	BUY
COMPANY :	IBM
PRICE :	105
VOLUME : (x1000)	55

Fig. 14. System presentation.

6. Conclusion

In this paper, we propose an integrated mobile service environment (MSE) for the handheld devices and implement a prototype of the framework. The system supports cross-platform, is equipped with efficient *cache* mechanism and employs XML-RPC as communication mechanism to provide mobile users infinitive services from the service provider. By using the encoding/decoding scheme on XML-RPC between the client and server, the amount of data transmission is effectively reduced and the communication time is shorten accordingly. Through the *cache* mechanism, the most recently used and frequently reused SA are properly maintained to enhance the performance.

We successfully realized a real-time stock transaction system. Users can access real-time information in the stock market and make transaction (buy or sell) on the handheld device. In the future, we intend to enhance the capability of MSE. Especially, in data transmission between the client and server we hope to achieve the security supported by WTLS (Wireless Transport Layer Security [17]) which is widely use in WAP (Wireless Application Protocol). It is foreseeable that the bandwidth of wireless network will be getting larger and larger and makes the multimedia (i.e., text, picture, voice, animation, movie, etc.) information appear on the mobile devices possible. This urges us to develop this service environment toward more friendly, more powerful and more secure in the wireless circumstances.

References

[1] "4thpass Kbrowser," 4thpass Inc., http://www.4thpass.com, 2000.

[2] M.J. Albers and L. Kim, "User web browsing characteristics using Palm handhelds for information retrieval," In Proc. of IPCC/SIGDOC 2000 on Technology & Teamwork, pp.125–135, Sept. 24-27, Cambridge, MA, USA, 2000.

[3] "AvantGo.com 4.0," AvantGo Inc., http://www.avantgo.com, 2001.

[4] "CAC 1.0," CAMEO Accelerated Cyberspace Inc., http://www.mycameo.com, 2001.

[5] "Code Warrior 7.0," Metrowerks Inc., http://www.metrowerks.com, 2001.

[6] C. Colafigli, P. Inverardi and R. Matricciani, “ InfoParco: an experience in designing an information system accessible through web and WAP interfaces,” Proc. of the 34th Annual Hawaii International Conference on System Sciences, pp.3510–3515, Jan. 3-6, 2001.

[7] “Falch.net DeveloperStudio 2.6,” Falch.net Inc., http://www.falch.net, 2002.

[8] A. Garmash, “A geographic XML-based format for the mobile environment,” Proc. of the 34th Hawaii International Conference on System Sciences, pp.3492–3500, Jan. 3-6, 2001.

[9] “Handspring Blazer 2.0,” Handspring Inc., http://www.handspring.com, 2001.

[10] C.W. Lan, C.C. Chien, M.Y. Hsieh and I. Chen, “A mobile e-commerce solution,” Proc. of International Symposium on Multimedia Software Engineering, pp.215–222, Dec. 11-13, Taipei, Taiwan, 2000.

[11] S.D. Maclean and D.J. Dailey, “Real-time bus information on mobile devices,” Proc. of 2001 IEEE International Conference on Intelligent Transportation Systems, pp.988–993, Aug. 25-29, Oakland, CA, USA, 2001.

[12] Palm OS SDK Reference, Palm Inc., http://www.palmos.com, 2001.

[13] “Palmscape English 3.1.3,” “Xiino 1.0.5,” ILINX Inc., http://www.ilinx.co.jp, 2001.

[14] X. Shan and J.J. Li, “A case study of IP network monitoring using wireless mobile devices,” Proc. of Tenth International Conference on Computer Communications and Networks, pp.590–593, Oct. 15-17, Scottsdale, AZ, USA, 2001.

[15] W3C Consortium, http://www.w3.org, 2001.

[16] Waba Specification, http://www.wabasoft.com, 2001.

[17] WAP 2.0 Specification, http://www.wapforum.org, 2001.

[18] “Wapaka 2.0,” Digital Airways Inc., http://www.digitalairways.com, 2001.

[19] “WAPman (Palm) Classic 1.8,” EdgeMatrix Pte Inc., http://www.edgematrix.com, 2000.

[20] XML-RPC Standard Reference, http://www.XML-RPC.com, 2001.

[21] “YesPalm 1.1,” Yesmobile Inc., http://www.yesmobile.com.tw, 2001.