E-wallet Software Architecture with Decentralized Credentials

RAÚL HERNÁNDEZ[1] and STIG FRODE MJØLSNES
{raulh@stud.ntnu.no} {sfm@item.ntnu.no}
Department of Telematics, NTNU

Abstract

This paper reports on results from a Master thesis project undertaken to develop software architecture for *decentralized credentials*, a generalization of the concept of an electronic wallet system developed in the European research project CAFÉ [1]. Within this new model you can leave most of the content of your electronic wallet (credentials, keys and passwords) at the security of your residential keeper, while roaming with your favourite mobile terminals. The main aim for this work has been to design a software architecture based on PersonalJava [2] (portable code) with emerging web technology through the Apache project and SOAP, the new RPC paradigm based on XML (portable data) for web services. The experiments were carried out using WLAN and demonstrating that the SOAP protocol shows a great feasibility to implement this architecture for real e-commerce websites compared to other middleware options (RMI, CORBA). Finally, an authentication protocol developed in a parallel thesis work [3] was implemented in the software architecture.

Keywords: mobile commerce, ubiquitous computing, security, digital credentials, SOAP.

1 Motivation

Small mobile devices such as mobile phones and Personal Digital Assistants (PDAs) empowered with the capacity to connect to the Internet and exchange information become available. Furthermore, with the continuing progress in computing and communication is believed to move toward "ubiquitous computing" [4], that is to say, making many services available throughout the physical environment, but making them effectively invisible to the user and many service providers try to give and answer to this need. One significant business area is mobile commerce involving applications and information delivered to the mobile device in digital format (figure 1).

Currently, the underlying technology used is Wireless Application Protocol (WAP) (i.e Nokia wallet [5]) over a network operator-provided bearer, like Circuit Switched Data (CSD), General Packet Radio Service (GPRS) or SMS (Short Message Service). Digital content is predominantly being paid for by means of mobile operator billing, i.e. ordered by SMS and delivered via SMS or WAP downloads. However, the natural evolution is the transition towards web services using eXtensible Hypertext Markup Language (XHMTL) as the standard browser language and the switch to using the TCP/IP stack as the transport layer to increase the number of remote transactions because they provide a richer browsing experience as well as faster and more secure connections (for example SSL).

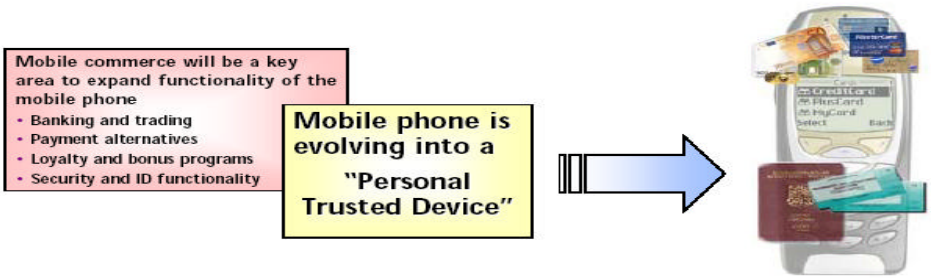


Figure 1. New vision for mobile devices

[1] Affiliated with Polytechnic University of Valencia, and master thesis work carried out at Dep. Telematics, NTNU Spring 2003.

Many companies and ICT industries have invested heavily in developing of new mobile services and concepts like m-payments and m-wallets employing the technology mentioned above. Activities for promoting open, non-proprietary technology standards include mobile commerce groups such as Mobey Forum (MF) [6], Global Mobile Commerce Interoperability (GMCIG) [7], and Mobile Electronic Transactions (MeT) [8]. However, the scenario is still quite confusing to the user because there are many service providers but not any workable standard to operate among them.

2 Related work

Some of the basic ideas for the development of this e-wallet architecture are taken from the research project CAFÉ [1]. The goal of CAFÉ was to develop innovative solutions for conditional access, in particular related to digital payments by electronic wallets. One of the key questions within this project was not to force the user to trust other parties, and not to deposit his credentials into the hands of someone else (a notably distinct approach relative to credit card companies), maintain an open system with multiple operators, and guarantee interoperability among many service providers.

In addition to interoperability our architecture should operate with existing and new protocols and payment instruments. This idea is also present in some e-wallet architectures like SWAPEROO [9]: it should be able to accommodate legacy and future payment protocols. The architecture must not be fixed to a specific set of protocols and force the user to use different wallets depending upon the payment instrument he wants to use.

The master thesis of Steinsholt [10] studied the problem of architecture design to achieve decentralized credentials software based on RMI. This work moves toward a web architecture based on SOAP and introduces the idea of an extensible wallet.

3 The model

The proposed solutions and standards for m-wallets include mobile terminals with credit card-slots, the creation of special, separate accounts on remote servers, or using the SIM card in GSM mobile phones as identification and storing of other credentials. However the provision of a multitude of mobile services and credentials quickly produces scalability problems for the users. This architecture also tries to provide a solution to the practical problem of the multitude of cards, credentials and passwords that the user must carry so in this new system remembering all the different PINs will not be necessary.

The proposed architecture has three basic elements:

The credential keeper. It contains and protects the credentials. It can be implemented using a networked computer.

The wallet. It is the instrument that hides the complexity: once a user decides to make a purchase / show his credentials to get access to some kind of service, the wallet drives the protocol to transport information from the keeper to the service. The e-wallet terminal should be a mobile phone, a PDA or a dedicated hardware module. When a user needs his credentials, for example when paying for or getting access to a service, the e-wallet terminal is responsible for retrieving the credentials required by the service. Instead of having the credentials stored in the e-wallet terminal, the e-wallet retrieves the credentials from its corresponding credential keeper and presents them for the

service application. This way the user can keep all his credentials physically safe, but still have access to them anywhere and anytime, as long as the e-wallet can get connected to its credential keeper. The operations are performed according to a protocol interacting with the user through a graphical user interface (GUI).

The service. The service is presented as a remote entity that may be an end-user, vendor or bank that requires a proof or identity from the user.

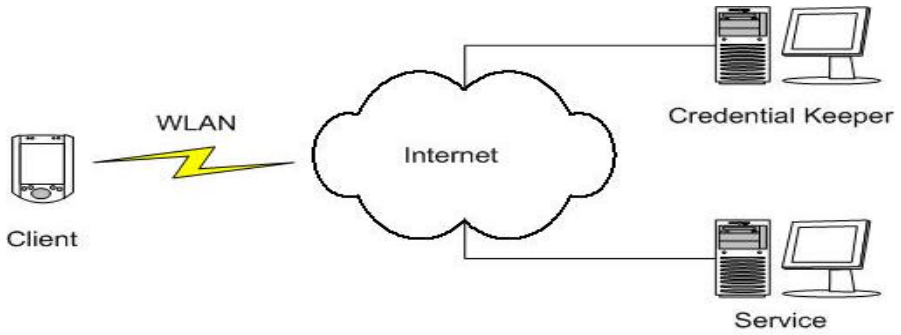


Figure 2. Symmetrical architecture.

These three elements are supposed to be physically distributed and communicated using a wireless network (for this experiment a WLAN was chosen but the results are easily extrapolated to Bluetooth or cellular packet switching like GPRS).

The paper tries to give an answer to several questions to develop this architecture. Section 4 describes the platform (language programming) to implement the architecture. Section 5 explains SOAP as middleware to communicate keeper, mobile and server. Section 6 describes the implementation for the keeper, mobile and server.

4 The platform

Sun Microsystems is promoting Java as an excellent platform for developing wireless applications. With wireless devices varying highly in system configurations, portability is an important advantage. Applications need only be written once and then can be theoretically distributed to any Java enabled device. Portability is more critical in embedded devices or mobile devices with small footprint because unlike desktop systems no one operating environment dominates the market. The range of processors and specialized operating systems makes the environment across these devices much more diverse (e.g. Symbian, PalmOS, WinCE) although Java components don't care what kind of computer, phone or operating system they run on (“write once run everywhere”).

This advantage is particularly important in considering the possibility of dynamic “Over-The-Air” (OTA) [11] deployment of wireless applications for e-wallets. On the one hand, when downloading applications, users will expect to be able to have confidence that the program will not cause harm or disruption to their device, thus the security advantages of Java (sandbox) become important. The portability is essential, furthermore, due to the multitude of wireless devices. It would be almost impossible to write and maintain a single application that could be downloaded and used by everyone, especially without the need for additional configuration. Java makes the application portable easily.

The platform chosen was PersonalJava [2], a much more general application environment than Java 2 Platform Micro Edition (J2ME) [12]. The target devices for Personal Java have up to 2 MB of ROM and at least 1 MB of RAM available for the Java platform itself (high end devices), with more required for application software. Some of the larger PDAs and communicator devices, such as the Compaq iPAQ and the Nokia 9210 cell phone, are currently

using the Personal Java environment. Personal Java is based on JDK 1.1.8 and includes a fully featured Java VM. The specification designates each of the core JDK 1.1.8 packages as required, modified, or optional. Similar designations may also be applied to individual classes and methods. A required package must contain all of the classes from its JDK 1.1.8 counterpart, and each class must be a full implementation. However, many developers complain about Personal Java because it is not a good platform to develop secure communications (absolutely necessary for e-wallet architectures). Personal Java is based on JDK1.1.8 and it does not support RMI over SSL nor JSSE (Java Secure Socket Extension) or JCE [13] (Java Cryptography Extension) so many developers are expected to use The Connected Device Configuration (CDC) [14] as a migration path to the Java 2 platform (CDC is based on JDK1.2).

The development of the application was done using a Compaq iPaq 3870, a high end PDA, suitable for Personal Java. The next step is the choice of a Java Virtual Machine that supports the Personal Java specification. Steinsholt [10] demonstrated that the Jeode Virtual Machine [15] shipped together with the Compaq iPaq 3870 and therefore free of charge, works well for Java applications. One of the most interesting features of this virtual machine for our purposes was the possibility of using Java Plugin technology to install Java applications dynamically.

The wallet architecture we propose can interoperate with multiple existing and newly developed instruments and protocols. It is supposed that the user has a collection of applets for different available protocols on the keeper. Every time the user requires an e-wallet service it is possible to download an ad hoc applet for the specific service through an applet dispatcher installed on the keeper. Within this architecture, the applet will establish two network connections (keeper and service) so it is also important to emphasize that the security policy defined in a java.policy file must be modified to grant permissions (for the iPaq this file is located in the folder `$JDKROOT/jre/lib/security`).

The application is executed as a Java applet within a web page and it is executed bypassing the web browser in the mobile device. The deployment of the application is similar to the Over-The-Air (OTA) mechanisms for mobile phones (now instead of downloading a Java program called midlet, an applet is downloaded) so it is easy for a user to download and install it (provisioning).

5 Middleware

SOAP

As the name suggests, the Simple Object Access Protocol (SOAP) was designed as a simplified mean of making remote procedure calls across the Internet. SOAP is gaining much attention in the field of distributed systems because it also underpins the new Microsoft's .NET architecture. For simplicity and easiness of integration with web services, it was the middleware option chosen for our architecture and it will be examined with more depth.

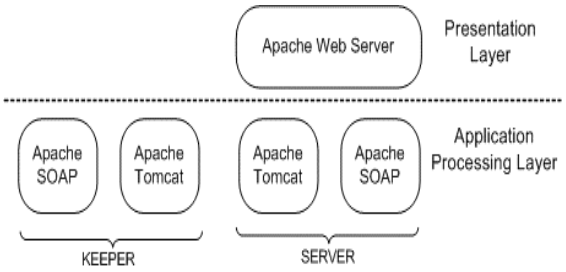
The protocol is primarily intended to be used over HTTP (HTTP POST), but it can also be used with SNMP and other application layer protocols. The philosophy for sending objects between client and server is the same as Java Remote Method Invocation, RMI (serialization). SOAP uses XML for serializing objects between client and server (generally RMI is faster than SOAP because RMI serializes objects into a stream of bytes instead of XML which produces more overhead) and for binding the remote method (for every service a deployment descriptor, which is an XML file that contains information about the service and the remote methods, must be created) so SOAP solves easily than other middleware platforms like RMI and CORBA the problem of interoperability (everything can generate XML).

One advantage of using SOAP over HTTP is that it is easy for the service providers to integrate it into existing network infrastructure that is designed for allowing clients to retrieve web pages. The developer does not need to modify the existing network and web services model to create SOAP applications (For this architecture the keeper was implemented with a Tomcat server [18] configured to serve Java Server Pages and deal with SOAP requests at the same time) and has the potential to create a transparent web of services and applications which can be accessed from anywhere by anyone on demand.

Another advantage of using SOAP compared to other middleware is that it requires fewer resources to process a request and a response. This makes it possible to use SOAP for devices with small footprint even with the basic MIDP profile (vs RMI that requires at least Personal Java) with some limitations.

The SOAP implementation chosen for the iPaq was Wsoap [16] (Wingfoot), based on SOAP 1.1. It is tested against major SOAP server implementations and can be used in MIDP/CLDC (kvmwsoap_1.04.jar), PersonalJava/CDC, J2SE and J2EE platforms (j2sewsoap_1.04.jar). For the credential keeper and the service the SOAP implementation chosen was Apache SOAP [17] (not available for PersonalJava yet as it requires Java 1.2 at least).

6 Software architecture



The implementation of the credential keeper and the server for our e-wallet architecture is simply a networked PC. This implementation is almost symmetrical as the entire request - responses are processed by the same SOAP implementation (Apache SOAP).

Figure 3. Two layer architecture

Web architecture was chosen for these elements, splitting the application into two logical layers (figure 3): a presentation layer and an application-processing layer.

The presentation layer is responsible for the interaction with the user and display (for example through a Graphical User Interface (GUI) or web pages). Apache web server at the server side represents this layer. Using this web server, all the services can be published and the users can access to them through a web browser (for example Pocket Internet Explorer for the PDA) using the e-wallet device. This layer is not necessary for the keeper as all interactions between user and keeper are driven by SOAP only.

The application-processing layer is responsible for implementing the logic of the operation. Java Server Pages (JSPs) and Apache SOAP realize the credential keeper application. The goal of JSPs is a simplified creation and management of dynamic web pages, by combining Java code and HTML in the same file. An applet dispatcher may download applets dynamically and on the fly, thereby supporting specific services and user's payment instruments.

Now all the elements have been described and it is possible to give an overall view of our architecture. The applet dispatcher has been implemented with a Tomcat JSP/Servlet [18]

engine that downloads applets to the mobile device on demand depending on the service selected by the user. Once the applet is downloaded, it performs all the necessary operations using SOAP as middleware.

We can describe the operation into the following steps (figure 4):

1. *Service discovery*. The user browses for a service: once the web service has been created, a service consumer must be able to locate it in order to be able to use it. When the user selects a service, he downloads the applet that "understands" the protocol for that specific service [2].
2. *Provisioning*. The Tomcat engine must be able to generate dynamic HTML code through JSPs. A JSP must process the request from the browser and get the parameters that identify the applet needed to run the service.
3. *Processing*. The applet drives the protocol according to the service demanded.

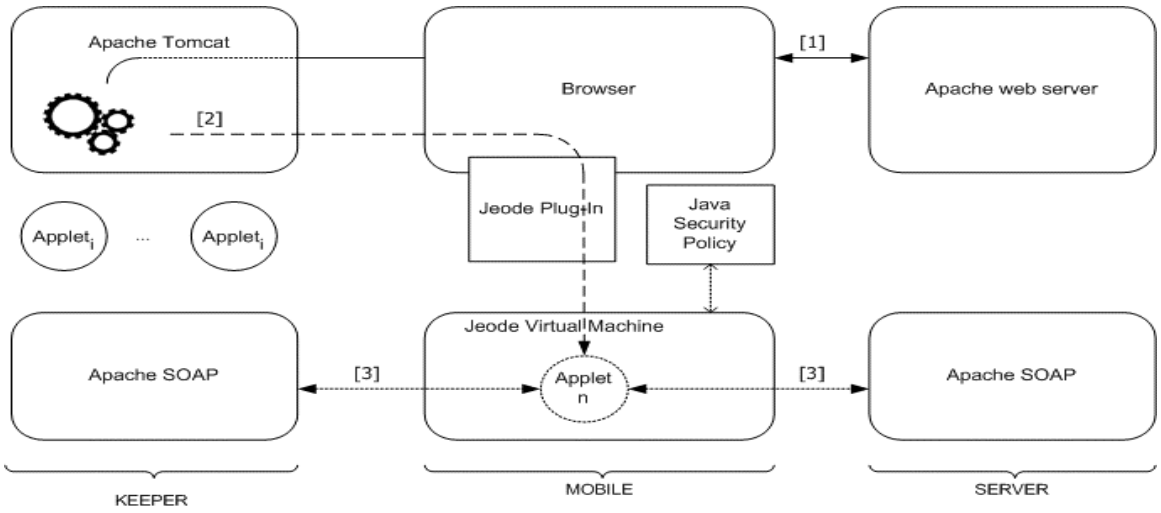


Figure 4. Software architecture

7 Protocol Design

A crypto protocol for authentication

The situation of "open the door" service was chosen to demonstrate this architecture. This situation is described on Mjølsnes' paper [19]: the user is about to enter the door to his company carrying only his mobile on-line wallet device. The credential corresponding to the door access has been issued in the form of a key but the user does not carry this key with him, it is localized at the credential keeper at home so the user connects to the door access control with his mobile device using in this case WLAN.

Figure 5. "Open the door" scenario.

Authentication protocols (Zero-knowledge)

Zero-knowledge protocols, as their name says, are cryptographic protocols that do not reveal information about secret parameters and keys during the protocol execution. They have some very interesting properties, i.e as the secret itself is not transferred to the verifying party, they cannot try to masquerade the user to any third party. The verifier does not learn anything from the protocol except that the prover knows or holds the secret key. This is the central zero-knowledge concept.

Authentication protocol

The protocol implemented is a Schnorr's modified authentication proposed in [3] and very suitable for our decentralized architecture (Figure 6).

1. p is a large prime, i.e. p˜ 2_{1024}.
2. q is a large prime divisor of p – 1, i.e. q ˜ 2_{160}.
3. g . Z_p has order q.
4. t is a security parameter such that q > 2. For most practical applications t = 40 will provide adequate security.

We will also use two hash functions, h as 'Message Detection Code, MDC' and h_k as 'Message Authentication Code, MAC' being k a secret shared between keeper and mobile. S and P= g^S mod q represent the keeper's private and public key respectively.

Java Math API provides classes for performing arbitrary-precision integer arithmetic (BigInteger) needed in the cryptographic operations. The hash functions are calculated using the Java Cryptography Extension (JCE) [13], a set of packages that provide a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. The Mobile is a simple booster between keeper and service, but it is also necessary a cryptographic API to perform the MAC operation (â) .The option chosen was Bouncy Castle Crypto API [20], as JCE requires JDK 1.2 at least.

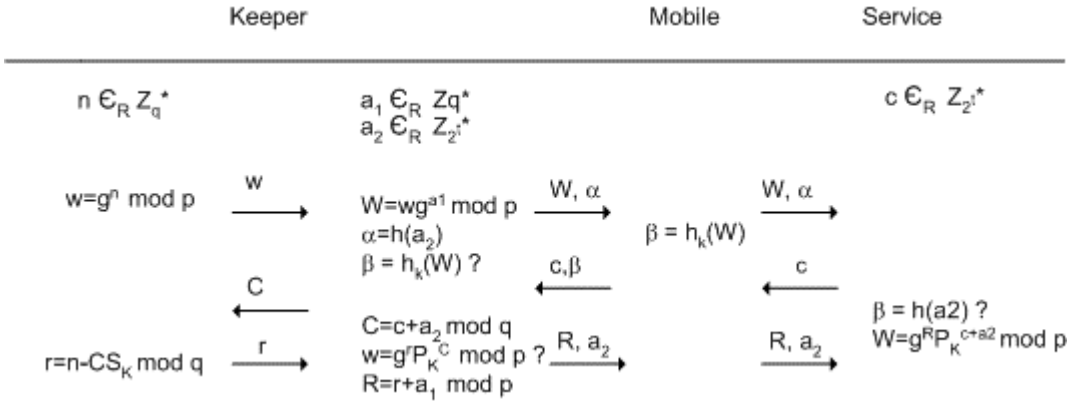


Figure 6. Proposed authentication protocol

The authentication protocol based on SOAP

All of these actions have been designed as finite state machines (specifically three finite state machines: keeper, mobile and server). Finite State Machines (FSMs) have become a standard model for representing object behaviour and numerous specification notations are based on the concept of FSMs (for example UML and SDL incorporate FSM notations).

For the authentication protocol we have defined a finite set of states and every state has associated a set of actions, for example calculate witness, throw a challenge, check the response… (Basically the keeper and the server perform all the operations and the mobile device is a booster between keeper and server although it has also been implemented as a state machine). The events that produce transitions (inputs) and the outputs are SOAP calls carrying the appropriate parameters.

All the Requests and Responses have a common format, a header field, showing the kind of the packet and the data field containing the payload.

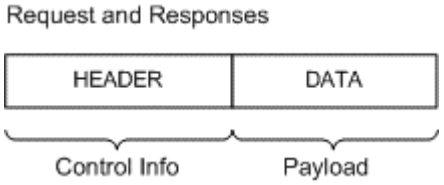


Figure 7. Packet

We have chosen the following notation for the calls: the first letter indicates if the request / response is / comes from the keeper or the service. The second letter indicates if it is a request or a response and the small letter indicates the state (i.e KQ_i and SQ_i indicate requests (Q) to the keeper (K) and the server (S) respectively, analogously KR_i and SR_i indicate responses (R) to the keeper and the server for the state i.

The following figure shows the specific type for the requests and the responses

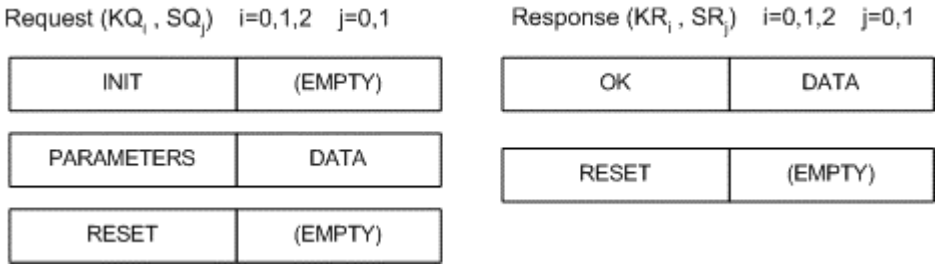


Figure 8. Request and Responses

The state machines have been implemented as Moore FSMs

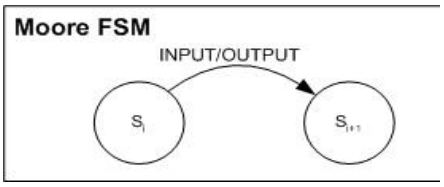


Figure 9. Moore FSM

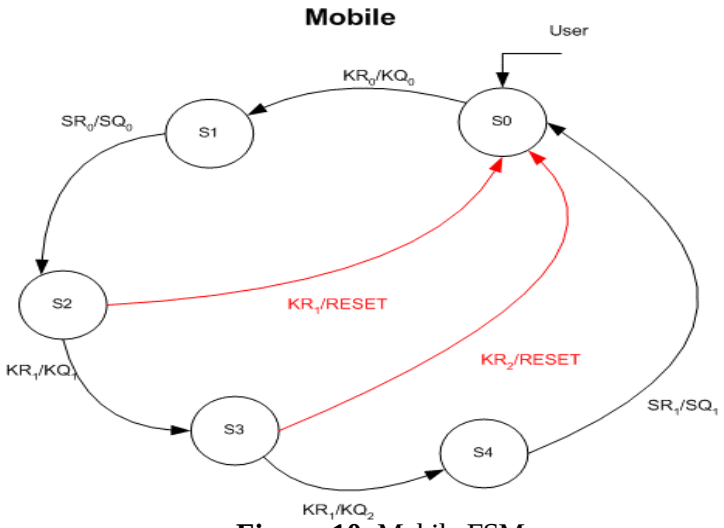


Figure 10. Mobile FSM

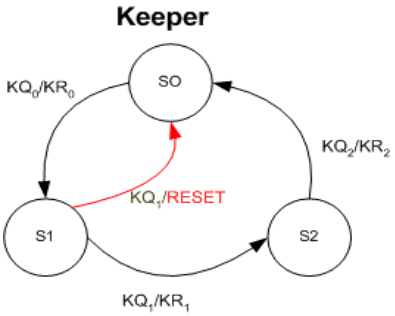


Figure 11. Keeper FSM

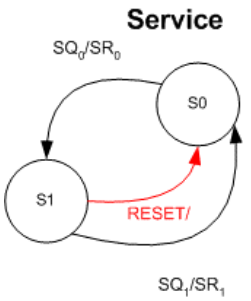


Figure 12. Service FSM

The most complete description for the protocol is showed below (figure 13). The operations described in figure 6 are shown together with the appropriate state (in brackets). The messages in red are possible errors.

Note also the three steps (service discovery, provisioning and processing) described previously (section 6).

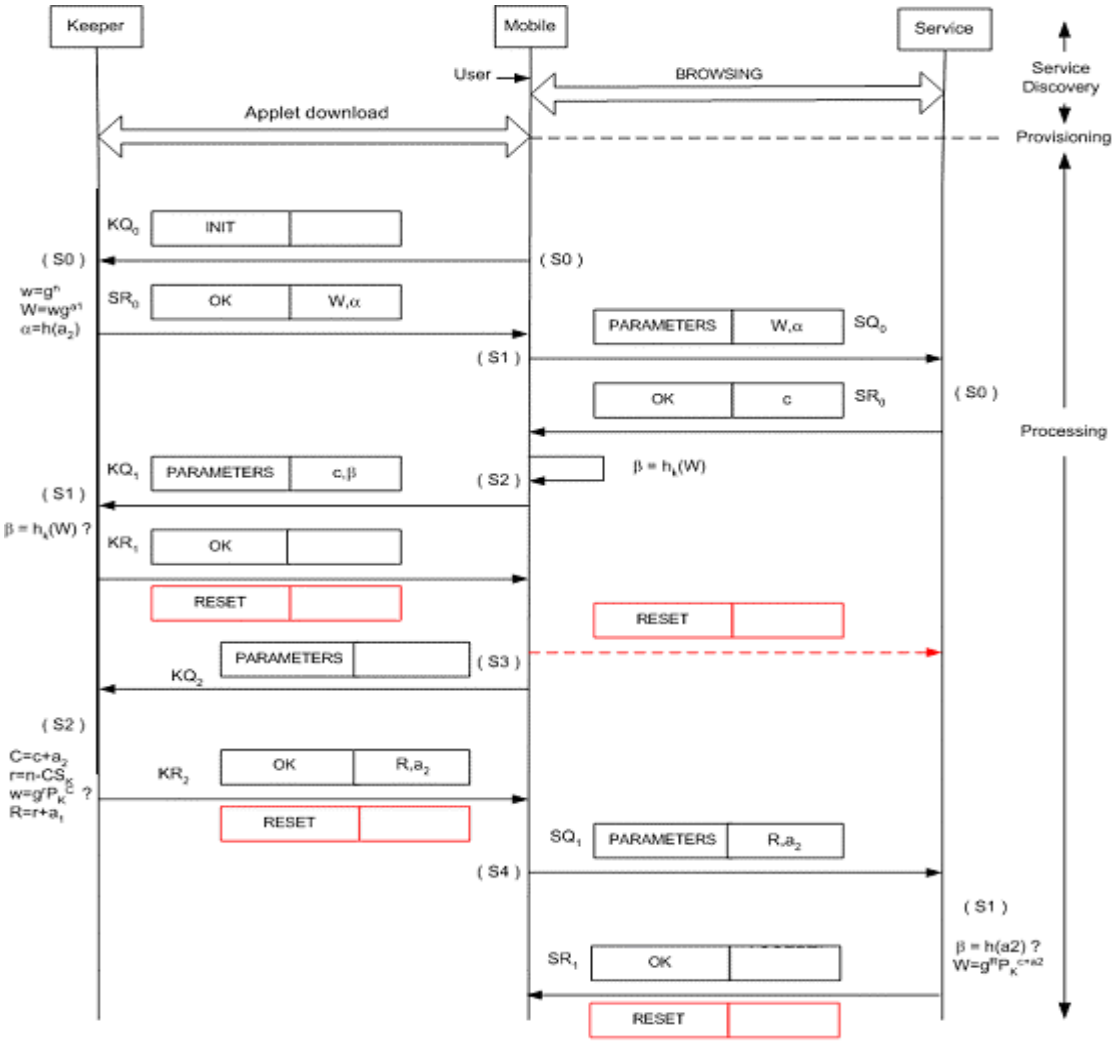


Figure 13. Message sequence diagram for the authentication protocol based on the SOAP primitive.

8 Discussion and further development

The main aim of this project, a decentralized and open software architecture, is claimed to be achieved: our software architecture uses SOAP, the most standardized middleware to retrieve the credentials from the keeper and reply to the challenges by the server and it is also an extensible architecture: the protocols are not fixed since the mobile can download the appropriate java code (applet/midlet) according to the service. It can also get interoperability with proprietary protocols and evolve with new services and protocols without losing compatibility.

The deployment is also really feasible since it is not based on proprietary software: we have just used software open source (Apache project) and a free SOAP implementation (Wingfoot SOAP).

So far, the gap between Personal Java (JDK1.1.8) and the Java 2 platform (JDK1.2) has been traumatic for Java programmers writing applications for devices with limited memory resources and processor power such as cell phones and PDAs as they have in common the fact that J2SE, the standard Java platform used on desktop systems is not supported due to the small footprint.

Many developers complain because some important issues, most importantly related to networking security, have not been considered: the use of "secure middleware" as RMI over SSL or SOAP over HTTPS is not a possible option working for Personal Java as in both cases it requires the Java Secure Socket Extension (JSSE) available only for the Java 2 Platform.

Looking at the middleware options, the reasonable solution is SOAP: it is easy to deploy web services and at the same time integrate them with SOAP as middleware to perform the business logic. SOAP is a lightweight protocol (vs CORBA and RMI) and it is available not only for high end PDAS like iPaq but for low-end devices (MIDP/CLDC profiles) too. The main handicap is performance: SOAP universality (XML) requires large messages because of its ASCII encoding. Security is one particular area that could be improved by employing SOAP. One interesting option to improve SOAP security could be its integration with other initiatives based also on XML like Oasis Consortium' SAML [21], the Security Assertions Markup Language, an XML grammar for expressing the occurrence of security events, such as an authentication events.

Another area to investigate is the service discovery: it would be interesting to try to develop m-services moving the communication from WLAN to Bluetooth as it opens new possibilities like the Bluetooth Service Discovery Profile to discover services. Currently the JSR-82 [22] lead by Motorola tries to hide the complexity of the Bluetooth protocol stack behind a set of Java APIs (JABWT [23], Java APIs for Bluetooth) being developed through the Java Community Process (JCP). It would be also interesting to test some proprietary solutions: some firms like Atinav [24] have implemented its own Bluetooth stack and Java APIs for Windows CE.

Further research should also test new virtual machines like NSIcom's CrE-ME VM [25] (beta version) for PocketPC and the new Jeode's VM [26] both including CDC compatibility (CDC is based on CVM, "Compact Virtual Machine", a virtual machine that supports all the features of the full J2SE VM). Surprisingly, Sun does not provide a version of CVM for PocketPC platforms during our study, only a reference implementation that can be compiled for Linux (strictly speaking, only Red Hat Linux Version 6.2 is supported) and VxWorks, a real-time operating system.

References

[1] Boly et al., The ESPRIT project CAFE – High Security Digital Payment Systems. *ESORICS 94 Proceedings* (Springer-Verlag, 1994)
[2] Personal Java Application Environment, http://java.sun.com/products/personaljava/
[3] Marius Gjerde, Decentralized credentials. Dep.Math, NTNU Master thesis (June 2003).
[4] Mark Weiser, Ubiquitous Computing. IEEE Computer (October 1993)
[5] Nokia wallet, http://www.nokia.com/nokia/0,5184,3494,00.html
[6] The Mobey Forum, http://www.mobeyforum.org
[7] Global Mobile Commerce Interoperability Group, http://www.gmcig.org
[8] Mobile Electronics Transactions, http://www.mobiletransactions.org
[9] Daswani,Boneh et al. SWAPEROO: A Simple Wallet Architecture for Payments,Exchanges,Refunds,and Other Operations
[10] Steinsholt, Vegard, Software Architecture for on-line electronic wallet. NTNU master thesis (February 2003)
[11] Over-the-Air (OTA), http://wireless.java.sun.com/midp/ttips/wtkota/
[12] Java 2 Platform Micro Edtion (J2ME), http://java.sun.com/j2me
[13] Java Cryptography Extension (JCE), http://java.sun.com/products/jce/
[14] Connected Device Configuration (CDC), http://java.sun.com/products/cdc/
[15] Jeode Virtual Machine, http://www.insignia.com/content/products/jvm/pda.shtml
[16] Wingfoot SOAP, http://www.wingfoot.com/products.jsp
[17] Apache SOAP, http://ws.apache.org/soap/
[18] Apache Tomcat, http://jakarta.apache.org/tomcat/
[19] Mjølsnes, Stig Frode and Rong Chunming
On-Line E-Wallet System with decentralized credential keepers
Mobile networks and applications 8,2003
[20] Bouncy Castle Crypto APIs, http://www.bouncycastle.org/
[21] Security Assertion Markup Language
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
[22] Java Specification Request 82 (Java API for Bluetooth),
http://www.jcp.org/en/jsr/detail?id=082
[23] Java APIs for Bluetooth Wireless Technology (JABWT),
http://www.jabwt.com/
[24] Atinav's aveLink Bluetooth Protocol Stack,
http://www.avelink.com/modules/bt_windowsce.htm
[25] CrEme Virtual Machine http://www.nsicom.com/products/creme.asp
[26] Jeode Virtual Machine http://www.esmertec.com/products/products_jeode.shtm