Turning Multi-Applications Smart Cards Services Available From Anywhere at Anytime: a SOAP / MOM Approach in the Context of Java Cards

Didier Donsez[1], Sébastien Jean[2], and Sylvain Lecomte[1]

[1] University of Valenciennes, LAMIH/ROI,
Le Mont Houy, BP 311,
59313 Valenciennes Cedex 9, France
{didier.donsez,slecomte@univ-valenciennes.fr}
[2] University of Lille, LIFL/RD2P,
Bat. M3/111,
59655 Villeneuve d'Ascq Cedex, France
sebastien.jean@lifl.fr

Abstract. This paper presents a way to improve smart card integration in distributed information systems. Multi-application smart cards are able to offer a lot of services, but at the same time they are mainly disconnected. Our main goals are to ease the use of such services and to increase their availability. In order to reach them, we propose a JMS-SOAP based platform that enables remote clients both to discover what services a smart card provides and to request any service either synchronously and asynchronously.

1 Introduction

Four years old, Java Cards announced a revolution in smart cards world. Smart card application design is no more a specialist job. Although smart card application *Time-to-Market* was before 6 months long, every programmer familiar with the Java Language and technology is now able to write and ship is first smart card application in few days. Consequently, open smart cards boosted smart card use.

Even if Java Cards are mostly used for simple client-servers applications, one of the most promising role for open smart cards in distributed informations systems is to act as mobile agents acting as application servers. Emerging technologies consider WWW as a universal medium for interoperable services deployment, discovery and use. We think that open smart cards, and particularly Java Cards, can smartly take a place in such a model. Nevertheless, smart cards are disconnected most of their lifetime, even if Java-SIM cards (in mobile phones) are exceptions. This characteristic of smart cards has to be taken into account. In this paper, we present how multi-applications smart cards, and distributed applications where they are involved, can take benefits of middlewares that provide interoperability and asynchronous messaging.

Next Section briefly presents SOAP and related technologies as well as Message Oriented Middlewares (MOM). After this overview, Section 3 take a look at how the Java Card application model has evolved since four years. Then, Section 4 explains why and how to involve Java Cards in heterogeneous information systems where exchanges are either connection-oriented or connectionless, and gives implementation issues. Finally, Section 5 concludes and presents future work.

2 MOMs and services over the Web

In this Section, we discuss technologies and concepts involved in our proposal. We briefly overview MOM technology and present emerging standard for deploying and using services aver the Web.

2.1 Message Oriented Middlewares

Message Oriented Middlewares (MOM) are based on asynchronous messages as the single structure for communication, coordination and synchronization, thus allowing desynchronized execution of components. Reliable communication is guaranteed by message queuing techniques or specific communication protocols that can be added independently from the programming of software components. Asynchronous communication property decouples producers of information from consumers. They do not need to be both ready for execution at the same time. MOM's goal is also to maximize the portability of the transmission of messages between several applications.

JMS [13], which is a specification of a MOM based on Java Technology, defines a set of interfaces for messages queuing. JMS provides the application designers with two messaging models. the first is *Point-To-Point*, where a producer and one ore more consumers are highly coupled[1]. The other is *Publish-Subscribe*, where a producer delivers messages related to a defined topic and where consumers have just to subscribe to receive next messages of the subscribed topic. JMS is just a standard *de facto*, it is not a product by itself. However, there are a lot of commercial implementations compliant with this specification and a lot of open-source projects (as the example of JORAM[7]).

2.2 SOAP, UDDI and WDSL : deploying services over WWW

The WorldWide Web was originally created to enable information exchange in a simple and portable way across the Internet. It resides in a combination of four elements:

1. HyperText Transport Protocol (HTTP), a client-server protocol on top of TCP-IP
2. Uniform Resource Locator (URL), a universal binding system

[1] However JMS, as a MOM, guarantees that a message is delivered only once even if several consumers are connected to the producer's queue

3. HyperText Markup Language (HTML)
4. Web browsers

From sharing information between scientists, WWW moved to mass market. Server-side Scripting technologies, like CGI (Common Gateway Interface), were developed turning the Web as a universal medium for client-server applications. These technologies have however some drawbacks, like the difficulty to manage sessions and the lack of interoperability. So, Client-server programming over the web had to evolve to reach the goal of interoperability, simplicity and portability: the answer was SOAP.

SOAP [12] is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls, responses and errors. SOAP can potentially be used in combination with a variety of other protocols. However, the bindings defined for the moment describe how to use SOAP in combination with HTTP, and how to wrap RPC on SMTP and HTTP. An example of SOAP messages(transported with HTTP) is presented below, in the case of a quotation service. The request is transported in an HTTP's POST request (Figure 1) and the answer comes back in an HTTP response (Figure 2).

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 1. Quotation service request

A service request using SOAP/HTTP is completed according to the followings steps:

1. the client builds the SOAP message, according to the service description

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Body>
      <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 2. Quotation service response

2. the SOAP message is encapsulated in an HTTP POST request and transported to the Web server hosting the service
3. the Web server forwards the SOAP message to an *RPC router* server-side script or servlet
4. the RPC router:
   (a) parses the SOAP message
   (b) realizes the invocation
   (c) gets the response
   (d) sends back and HTTP response containing the SOAP response (normal or error)
5. the SOAP response is sent back to the client
6. the client parses the SOAP response and processes it

SOAP is an easy and extensible way to request services across the Web. It is independent from operating systems, programming languages and transport protocols. SOAP is not an object-oriented distributed system, there is no grabage collection neither object activation (parameters are not object references but values).

However, having a simple and interoperable way to request services is not useful if there is no way to know what a service looks like. UDDI and WSDL are recent standards that address this feature. The Universal Description, Discovery and Integration (UDDI) specification [15] describes a conceptual cloud of Web services and a programmatic interface that define a simple framework for describing any kind of Web service. The specification consists of several related documents and an XML schema that defines a SOAP-based programming protocol for registering and discovering Web services. WSDL [17] is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate, however, the only bindings described in WSDL documents describe how to use it in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

3 JavaCard and its Integration in Information Systems

The arrival of Java Card [5], and others open smart cards like Smart Card for Windows (SCW) and Multos was a kind of revolution in the smart cards world. Java Card technology allows applets written in the Java language to be executed on a smart card, within the Java Card Runtime Environment (JCRE), and provides a wide API to help developers to create applets. Both JCRE and APIs are modeled after the smart card ISO 7816 specification [4]. Java Card offers code sharing between applets, isolation with a sandboxing mechanism called *applet firewall*, and comes with a cryptographic API. Java Card is the most widely accepted open smart card. This is firstly due to its symbiosis with an increasing Java computing world. The integration of Java Card Technology into mobile phone technology [1] is another key of its success.

As Scott Guthery titled in [3], Java Card was initially thought as a mobile platform for Internet Computing. Sometimes, like the example Java language originally dedicated to washing machines, technologies are not firstly used for what they have been though. Java Card is in this case, and four years after the first Java Card announcement it is not yet widely used for Internet purposes. The first applications of Java Cards were maybe loyalty ones. The application model was a client-server one, were the client is usually on the terminal where the card was plugged. Designing applications involving smart cards was originally not an easy task because the most part of the code was used to manage the communication between the card and the client parts of the application through the card reader. Hopefully, nowadays this task is much more easier using frameworks like OCF [?].

Even if Java Cards can offer helpful services in an off-line mode, their use takes another dimension if plugged over a wired or wireless network. Java card can then be seen as a mobile agent serving its owner and representing him over distributed information systems. Open smart card is a young technology, and for the it does not yet play this role. However, as what Weiser called *ubiquitous computing* becomes a reality, mobile services provider is the most promising future for Multi-application smart cards.

Many ways were investigated to interact with Java Cards. All the mechanisms provided are however very similar because they consider the smart card as a server. Some researches have successfully turned the Java Card as a mobile Web server [2]. Close to the problematics of the WebSim project, The WebCard [11] is a Java Card applet implementing a lightweight IP/HTTP stack. WebCard is promising because it shows that a smart card can be seen as a traditional internet platform speaking IP. The two previous examples consider Java Cards like data servers, but another approaches try to integrate Java Cards as parts of distributed applications. A first example is the JC-RMI [16] technique that gives a Remote object view of embedded applets. It goes further in easing smart card application design, enabling for Java Cards the well known RMI tools used to build *classical* Java distributed applications. Here, applets as seen as Java remote objects. JC-RMI tools alleviate the burden of application designer by automatically generating stubs and skeletons. These proxies transparently manages the communication protocol and just let distributed object access to applets. One step more is Java Card and JINI enclosure. Some work, done [?]] or still in progress [16], intends to turn Java Card applets into Jini-based services. In such an approach, when a Java Card is plugged somewhere, its services are automatically registered and become available for distributed applications that are able to discover them through Jini's lookup service. Once discovered, the services can be used, as far as the smart card offering these services grant their access. Each step in open smart card integration in distributed information systems lets the card plays a smarter role. In the future, we argue that smart cards should be much more interactive and not only passive servers [6].

4 Toward a Generic, Dynamic, Connectionless Access to Smart Card Services

4.1 Motivations

Smart cards are disconnected 99.9 percent of their lifetime, but it can however be useful for a requester (which can be an application provider, a card issuer,. . .) to be able to alert the card or make updates when needed (i.e. not only when the smart card is connected). Card Management Systems (CMS) and Application Management Systems (AMS) now take an important place in smart card world because controlling and managing a fleet and thousands issued smart card that embed several evolving applications is not an easy task. Ideally, an AMS should not wait for smart card insertion to decide for application update. A better way should be to asynchronously notify smart cards that a later version of a given application has been issued and to automatically upgrade it if necessary. As a smart card is not able for the moment to emit request to a reference server in order to poll for updates, using an asynchronous messaging platform should be the easiest way to update applet version or personalization info for thousand cards at the same time.

As the Java Card is becoming a more and more "common" computing platform, a simple and interoperable mechanism of interaction is needed in order to turn the embedded services available from any platform. Frameworks such as OCF [10] address such problem but reduce the platform range (here to java-powered ones). Moreover, it might be interesting to assume service discovery and induced dynamic operation invocation. For example, when a smart card becomes plugged in a CAD that is part of the user's Personal Area Network (PAN), a daemon discovers what it is and what it can do. After this discovery time, the smart card authentication and privacy service is used to ensure trust in the user's PAN.

4.2 Requirements

In order to turn embedded services available to remote hosts, some requirements have to be fulfilled. First, we have to provide a generic mechanism to invoke operations . In the case of connection-oriented invocations, this is done using SOAP by encapsulating operations invocations into XML messages transported from point to point (i.e. from remote host to a reference host for the smart card) inside HTTP requests. The same mechanism is used to send back the invocation return value or an exception. Several transport protocols for SOAP messages have already been implemented (with HTTP or JavaMail). Our goal is to implement the transport of asynchronous SOAP messages by using a MOM. Connectionless invocations are then done using an intermediate asynchronous messaging layer that consists here in both JMS clients and servers. A smart card representation agent (one by particular card) must be deployed in order to inform of smart card presence, but also to forward queued messages were the card comes back. This agent can have a static address, or can be retrieved using directories such as JNDI or LDAP. It has to be combined to the MOM card-side client who receives incoming messages and send outgoing responses.

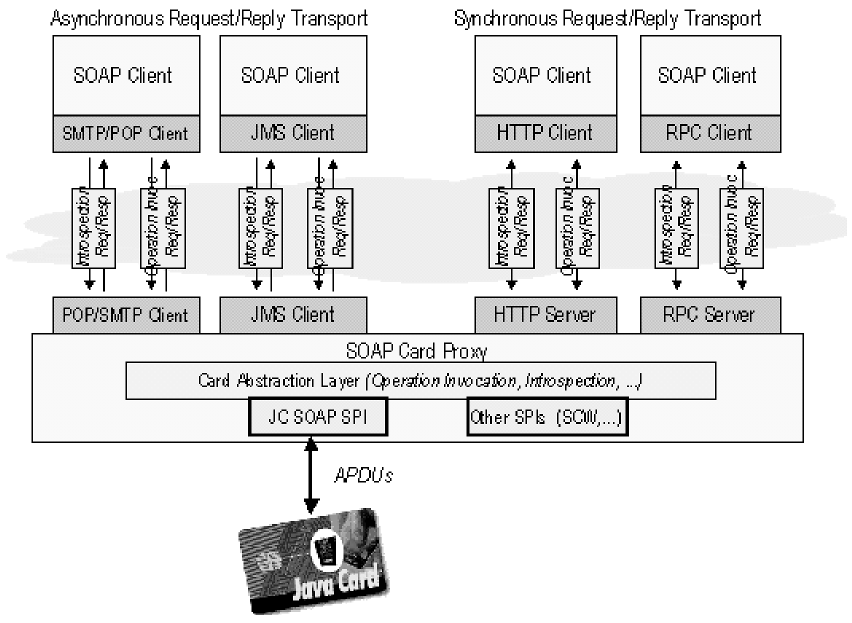

Fig. 3. Architecture Proposal

Turning these invocations dynamic means that the embedded services interfaces must be known or discovered at invocation time. So, in other words, service introspection has to be provided. We propose two different ways to get introspection. The first is to store the description and interfaces of available services for a given smart card. Due to interoperability issues, an adequate XML DTD, maybe adapted from those defined for WSDL and UDDI, should be used to structure this information. This repository-oriented could be interesting to avoid replication in the case of standard services descriptions. However an image of smart card description is required in the case of connectionless invocation and has to be coherently managed by the smart card agent. Another way is to add introspection facility to the card. In the case of the Java Card we propose to embed a directory applet that manages applications description (written using the same XML DTD or a more compact language). The directory, which interface enables clients to get smart card contents and applets/operations description at invocation time, has to be updated each time an applet is installed or removed. However, this update can be part of the smart card's RAD.

4.3 Prototype

Our architecture proposal is resumed on Figure 3. It is based on the use of SOAP technology to realize operation invocations and responses. The prototype that has been developed for testing purpose is based on the use of a Java Card. The operation invocations / responses are transported in a connection oriented way with HTTP or in a connectionless way with JORAM. The HTTP prototype is based on a subset of classes from Apache/SOAP package and Jakarta/Tomcat for servlet execution platform. The MOM prototype is based on JORAM [7], an open source implementation of JMS specification.

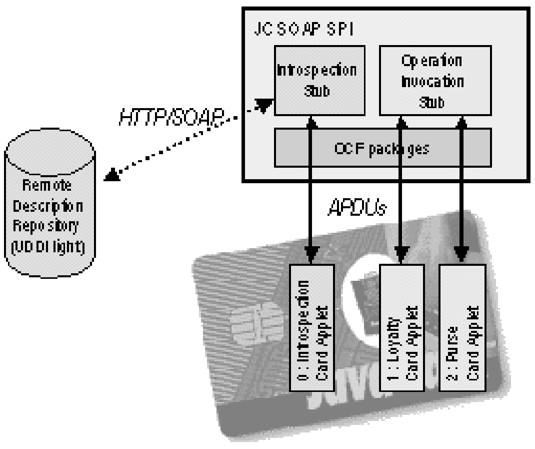

Fig. 4. SOAP Proxy for Java Card

As described below on Figure 4, a card-side SOAP proxy, implemented by a servlet provides introspection and invocation facility. It relies on the use of OCF to manage readers and cards access. The discovery mechanism is implemented inside the card; by the way of the Introspection applet that manages XML-based descriptions. However, this can be done using UDDI external repositories. The test application consists of a distributed client based on a Web interface that connects to a targeted Java Card and dynamically discovers inside applications (here, a purse and a loyalty application) and invokes operations on them.

5 Conclusion and Further Work

The objective presented in this paper is relevant with regard to necessities in communication. More and more services are available on the Internet network and on mobile telephony. Smart card take a more and more important place in service offers over these networks. To seamlessly integrate multi-application smart cards in order to turn them in common object-oriented execution platforms (regarding to client applications), interoperable operation invocations must be provided. As smart cards are not so frequently connected, the use of asynchronous messages can open new perspectives. In this paper, we have presented how to enable interoperability and asynchronism for distributed applications that involve multi-applications smart cards. To achieve this goal, we take benefits of SOAP and MOMs technologies. A SOAP proxy has been defined in order to enable SOAP-based operation invocations for smart cards. This proxy provides introspection facilities that, combined with an on-card introspection mechanism, make able distributed clients to dynamically discover and use the embedded services offered. We also define a MOM layer which, placed on top of SOAP, enables connectionless invocations. Although we focus on the Java Card case, the approach and platform we describe can be easily generalized to others multi-applications smart cards such as the Smart Card for Windows [14] or Multos [8].

Appendix: SOAP/HTTP envelopes for a smart card loyalty application

Invocation

```
POST /cardproxy HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: 334

SOAPAction: "invoke/loyalty:0:A0000000FF01"

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
   <SOAP-ENV:Body>
       <loy:GetBonus xmlns:loy="http://schemas.loyaltycard.org/">
           <service>ACME</service>
       </loy: GetBonus>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: 347

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
   <SOAP-ENV:Body>
       <loy:GetBonusResponse xmlns:loy=" http://schemas.loyaltycard.org/">
           <bonus>1900</bonus>
       </loy:GetBonusResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

References

1. European Telecommunications Standards Institute (ETSI): Digital cellular telecommunications system (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2 (GSM 03.19 version 7.1.0 Release 1998). May 2000.
2. Guthery, S.B., Kehr, R., Posegga, J.: How to turn a GSM SIM into a Web Server. In: Domingo-Ferrer, J., Chan, D., Watson, A. (eds.): Fourth IFIP TC8/WG8.8 Working Conference on Smart Card Research and Advanced Applications (CARDIS'2000), sept 2000, Bristol, United Kingdom. Kluwer Academic Publishers (2000) 209–222, ISBN 0-7923-7953-5.
3. Guthery, S.B.: Java Card: Internet Computing on a Smart Card. IEEE Internet Computing, **1** (1997).
4. International Standard Organisation (ISO): Information Technology - Identification cards - Integrated circuit(s) cards with contacts. ISO/IEC 7816-1,2,3,4,5,6,7,8. 1987-1999.
5. Java Card Forum : http://www.javacardforum.org.
6. Jean, S., Donsez, D., Lecomte, S. : Smart Card Integration in Distributed Information Systems: The Interactive Execution Model. In: Proceedings of IEEE 1st International Symposium on Advanced Distributed Systems, (Guadalajara, Mexico), March 2000.
7. ObjectWeb : JORAM homepage. http://www.objectweb.org/joram/joramHomePage.htm.
8. Maosco Ltd: Multos homepage. http://www.multos.com.

9. Merle, P., Vandewalle, J.J., Dufresne, E.: Intégration d'environnements hétérogènes : WWW, cartes à microprocesseur et Corba. In: Actes du 14e congrès INFORSID, Bordeaux, France, 1996.
10. Opencard Framework : http://www.opencard.org.
11. Rees, J., Honeyman, P.: Webcard : a Java Card Web Server. In: Domingo-Ferrer, J., Chan, D., Watson, A. (eds.): Fourth IFIP TC8/WG8.8 Working Conference on Smart Card Research and Advanced Applications (CARDIS'2000), sept 2000, Bristol, United Kingdom. Kluwer Academic Publishers (2000) 197–207, ISBN 0-7923-7953-5.
12. Scribner, K., Stiver, M.C., Scribner, K.: Understanding SOAP: The Authoritative Solution. In: Sams (eds), Jan 2000. ISBN: 0672319225.
13. Sun Microsystems : Java Message Service API Specification, v. 1.0.2. 1999.
14. Talvard, L.: The API services provided by the SCW. In: Proceedings of 1st Gemplus Developper Conference (GDC '99), Paris, France, 1999.
15. UDDI homepage. http://www.uddi.org.
16. Vetillard, E.: Tools for Integrating the Java Card API into Jini Connection Technology. In: Sun's Worldwide Java Developer Conference (JavaOne 2000), 2000.
17. Ariba, International Business Machines Corporation, Microsoft: Web Services Description Language (WSDL) Specification v 1.0 . 2000.