A PATTERN ORIENTED LIGHTWEIGHT MIDDLEWARE FOR SMARTCARDS

J-M. Douin
CEDRIC-CNAM
292 rue St Martin
75141 Paris cedex 03 France
douin@cnam.fr

J-M. Gilliot
GET - ENST Bretagne
Technopole Brest-Iroise - CS 83818
29238 BREST cedex 3 France
jm.gilliot@enst-bretagne.fr

Abstract Smartcards are a very interesting means to include our own datas and code in a distributed system, during our interaction with it. To achieve this, smartcards integration must be ensured. A transparent usage of card services is necessary to a more wide-spread use. This usage should be available remotely within a distributed environment. Additionally other features such as possible upgrades of code or addition of new services, notification of connections and disconnections, structuration of numerous smartcards are key requirements.

Our middleware is described in three steps. First, we describe mechanisms to turn Java Card Applets into webservices and show how to implement them in a lightweight infrastructure. Secondly, a mechanism, based on a collaboration pattern to become JavaCards active and to permit spontaneous discovery of services. Thirdly, we define a structuration of numerous JavaCards and Java Card Applets, as services dynamically available, to give opportunity to access and manage them easily.

1. Introduction

Smartcards are a very interesting means to include our own datas and code in a distributed system, during our interaction with it. To achieve this, Smartcards integration must be ensured. As noticed in many papers (see for example (Kehr et al., 2000a)), the basic use of Java Card suffers from its specificities to access applets. A transparent usage of card services is needed for a more widespread

use. This usage should be available remotely within a distributed environment and with minimal effort. Additionally other features such as possible upgrades of code or addition of new services, notification of connections and disconnections, structuration of numerous smartcards are key requirements. We propose in this paper an approach to develop infrastructure or middleware able to undertake those requirements. Flexibility is a central point with smartcards, new services may be added or removed at run-time, as well as connections and disconnections. We have discussed in (Douin and Gilliot, 2003) main characteristics of embedded devices, we will here focus on the extensions that are specific to smartcards and include them in our framework.

Our approach is to give interoperability to smartcards and to provide at the same time a lightweight infrastructure. This infrastructure is based on Java as card manufacturers chose to migrate to it in smart cards to achieve cross-platform compatibility. Our first choice was to adopt as collaboration schemes, the basic patterns proposed in literature (Gamma et al., 1994). Those patterns are sufficient to implement various schemes, such as polling or notification. The second choice was to use a minimal transport layer for communications. HTTP is now widely adopted, and it is current that even a small device implements a web server. Last choice was to make an adaptation to web at the hosting device level and not at Java Card level as proposed in (Rees and Honeyman, 1999).

In a first section, we define the key design concepts and the possible directions to develop such middleware. Main points concern: full interoperability and integration by making Java Card Applets accessible in an unified way, lightweight approach to be compatible with every device that accepts smartcards. Then, in the next section, we describe mechanisms to turn Java Card Applets into web-services and show how to implement them in a lightweight infrastructure. In section 4, we propose a mechanism, based on a collaboration pattern to make JavaCards active and to permit spontaneous discovery of services. Last section is dedicated to the structuration of numerous Java Cards and Java Card Applets, to give opportunity to access and manage them dynamically.

2. Design considerations

The development of a distributed embedded solution imposes to make some tradeoffs. We hypothesize that resources are minimal and that portability has to be ensured. Scarce resources impose a lightweight approach on the Java Card side but also on the hosting device side.

2.1 Basic architecture

A Java Card, to be accessible, must be hosted on an acceptance or hosting device. The functionality can be provided by other appliances such as rings (DS1957B, 2004) or GSM SIM which look different but provide the same fea-

tures. We used them as classical smart cards. Each card can contain some Java Card Applets or applets, that are pieces of executable code on the JavaCard, provided it is called with a specific protocol named APDU.

On an acceptance device, different Java Cards can be connected. We will assume in the sequel of this paper that the hosting device is connected to the Internet. We don't suppose any specific kind of connection. The hosting device can be of course a PC, but we propose that this device can be any kind of smart object, or embedded device. This is resumed in figure 1.

Figure 1. Physical architecture

2.2 Possible approaches

TCP/IP in Smart Card. A first choice is to provide directly in the smart card the basic TCP/IP communication standard to enable any network development (Muller and Deschamps, 2002). The purpose is to give full interoperability, adaptability and autonomy to the Smart Card. However, this approach is limited to basic connected communications, and do not propose extensible services for distributed integration.

Webserver in Smart Card. A second choice is the support of the HTTP protocol. Some authors have proposed specific webserver on the SmartCard (Rees and Honeyman, 1999),(Guthery et al., 2000). This simplifies the support of basic communications and provides real possibility of integration as the Card becomes a full address on the Internet and thus can be accessed remotely. But still, the approach is limited to the low computing power of the smartcard, which do not allow additional features. We can also notice that some works propose an opposite approach attempting to externalize even the information dedicated to a smartcard (Biget, 1999).

RMI approach. JavaCard RMI was first proposed in (Vandewalle and Vétillard, 1998), and is accepted as a standard (Inc., 2002). This give remote access to JavaCard, but only as synchronous call. As we thought it doesn't ensure open interoperability to non Java applications.

Proxy Server in Card Acceptance Device. As an acceptance device is mandatory for use of the Smart Card and the event of the connection to the device has to be acknowledged. So it is obvious to propose some software on the acceptance device side.

A first step is to propose a translation from the APDU standard to a more standard protocol. This ensures the encapsulation of this specific way of communication. Providing this server as a web server allows to take advantage of the interoperability at HTTP level which is clearly as open as TCP/IP, and more readable. We will see how to extend such server to provide full middleware requirements.

MOM. Message-oriented middleware provides the right abstractions, such as discovery, asynchronous messaging and so on. However, the use of standard middlewares like JMS (Donsez et al., 2001) or Jini (Kehr et al., 2000a) suffer from extensive needs of resources on the Card Acceptance Device. As emphasized in (Vogt et al., 2004), the major requirements for smart card middleware are encapsulation of specific communications, interoperability, and the integration of the systems. In the sequel of this article, we propose a such middleware approach but based on a lightweight implementation enabling its execution in most embedded connected devices.

2.3 Mobility of Cards

Cards are mobile devices : the card may connect and disconnect anytime. As they are highly portable, they are carried by their owners anywhere they go. Those user may wish to connect and disconnect, anytime, anywhere, to allow the use of features offered by SmartCard. This implies some assumptions that should be taken into account :

- On arbitrary connections means that should function spontaneously and be integrated with the surrounding distributed environment as soon as it is inserted in a reader.
- Anytime disconnections means that one cannot rely on connected communications. As stated in (Mascolo et al., 2002), this implies that connections are asynchronous.
- Localization is another important feature. But, as the Java Card must be connected to a hosting device, it can be delegated to it.

2.4 Hosting device is a web server

Providing a web server as an URL programming interface (Giorgio, 1999) has already been proposed in the context of smartcard for payment applications (Barber, 1999). As we already noticed in (Douin and Gilliot, 2003), this approach can be embedded with relatively small web servers like (Brazil, 2000). Extending URL programming enables, as we will see, many interesting features for smart cards. We chose generic and embeddable web servers available (Giorgio, 2000).

The Java language proposes standard APIs for patterns (Observer, ...), thus we adapt them to our implementation choices. Let's notice that with the same pattern abstraction, we can propose different implementations with different behaviors as described in (Douin and Gilliot, 2003).

2.5 Abstractions for the system

In order to allow an uniform manipulation of the different elements of a JavaCards based system, we define them as objects. We use patterns to handle dynamics in the system and to be able to manage the scalability. We use Web connections in an uniform way. As a consequence, applets are presented as servlets which hides specificities like APDU protocol. Hence generality can be achieved in accordance with modern protocols and service descriptions (Donsez et al., 2001). We have chosen to focus on functionalities and how to achieve them with scarce resources, both at the Java Card and hosting device levels. Notice that abstraction also permits to vary the implementation transparently in order to take into account non functional requirements or to improve performance.

3. Accessing and Managing remotely a JavaCard

As JavaCard must be connected to a device to be able to compute, we argue that this device may play the role of a web proxy for the JavaCard. Then this device translates a HTTP request into APDUs and retransmit the response. Hence, we decouple the transmission part from the application part included in the JavaCard.

3.1 Abstract the JavaCard

APDUS are the standard access for smartcards. Their use is very specific, hence we propose the use of web servers as wrappers. Connections of the smartcard to a host initiate the web server as proxy for datas and Java Card Applets are proposed as servlets. Cards are then presented as a collection of services. Java Card Applet being those services available individually.

(Donsez et al., 2001) propose to provide access to those services with the standard models of communications, namely MOM for asynchronous messag-

ing, SOAP as protocol, UDDI for discovery and WSDL for service description. This would allow a full integration in the Information System.

Another direction is to integrate Java Card in a Jini environment thanks to specific surrogate servers on the hosting device, as proposed in (Kehr et al., 2000b).

We have chosen to investigate more minimal protocols to make sure that a minimal web server is sufficient to ensure an interface to connected Java Cards. Hence we handle simple URL and provide dedicated definitions. An access to a Java Card Applet (naming it as *cardlet*) with GET method will then simply be something like : `http://server/JavaCardId/cardletId/?cardletParam=value`

With this simple feature, the embedded Java Card Applet become a *servlet* of the hosting device. To realize this, a simple http web server, is needed on the hosting device. A specific translation in APDU calls is provided through handlers. In the rest of the paper we show that it is sufficient to provide a dynamic management of numerous connected Javadards in a system.

Figure 2. Basic software architecture to abstract Java Card

3.2 Access to Java Card Applets

The first step to access *cardlets* as *servlets* is to provide a basic description of the different Java Card Applets and their parameters. This description should be integrated in the Java Card Applet itself, as a signature, giving syntactic call parameters and possibly the version. We have chosen to provide it with a definition compliant to the HTML forms definition in a compact format. A signature request will look like `http://server/JavaCardId/cardletId/?sig=0` and will return an answer such as `5TaText20S`, interpreted as a parameter named *aText* of type *Text* with a 20 character size. This basic signature offers a compression factor of 80 % essential for scarce resource.

It also gives the opportunity to provide more advanced services :

- Discovery of Java Card Applet description and corresponding construction of dynamic web pages or dynamic use on the client side. As our hypothesis is that the server can be embedded on arbitrary small device, it is obviously a coherent feature.
- Construction of a dynamic web page on the server side enabling the use of the Java Card Applet. Brazil (Brazil, 2000) provides a lightweight mechanism called BSL, offering comparable basic features like the JSP (Java server Pages) technology. The template of the page can even be embedded in the JavaCard. In this way the use and the construction of a web page access is provided by the card itself and adapted to the hosting device.
- Construction of description in a more widely used language just as WSDL or Jini discovery mechanism is possible and can be delegated to a device playing the role of gateway.

3.3 Mutual exclusion

Multiple clients can address to the same URL i.e. to the same servlet, this fact requires for some applications a mutual exclusion mechanism (Kehr et al., 2000b). This is achieved by addressing the URL with the parameter `beginMutex` to enter in critical section and to exit from this section with the parameter `endMutex`. The host computer provides this mutual exclusion between distributed clients of smartcard's servlet. These two URL are an example of use:

- `http://server/JavaCardId/cardletId/?beginMutex`
- `http://server/JavaCardId/cardletId/?endMutex`

In our current implementation, on the host computer, a client is identified by its cookie and a lock mechanism is associated to a servlet. If a client uses or decides to block a URL for its use, a second client can wait with or without a timeout. A possible problem is that a client can block indefinitely an URL. In our implementation presented in this paper, we chose to ignore this problem. There are solutions, with other problems, one of these consists in using a fixed maximum amount of time for a client or the client installs in advance a latency time.

3.4 Dynamic reconfiguration of the JavaCard

Java Card will be owned by users who don't want to worry about code evolution, or technical ways to adapt services or options on their javacard. A basic distant mechanism can be provided to update Java Card Applets remotely. It is just an adaptation of APDU commands at the HTTP level.

3.5 Security considerations

Quoting Baber (Barber, 1999): “ Responsibility for security now lies in the card server, not the browser. The card server can implement the appropriate model for its intended use, with the Java class library providing enough functionality to implement mechanisms such as code signing, access control lists and so on.” In other words security should be possible in our infrastructure, but we didn’t explore specifically this issue specifically.

3.6 Summary

Connecting the Java Card Applets on the web has reified them as distributed objects which can be called like servlets, identified by checking their parameters and updated. As these services are interfaced in accordance to HTTP protocol they provide the widest interoperability with other systems.

4. Distributed applications with nomadic Cards

In this section, we will discuss the adaptation of the Observer pattern in order to solve the plug and play requirement for an easy use of Java Cards. We will show that implementing this pattern at the HTTP level gives the opportunity to acknowledge events such as connections and disconnections of a Java Card in a distributed infrastructure.

4.1 Handling connections and disconnections

As smartcards users just come and go, connections and disconnections are normal events of the system. Those events may be used by applications for specific functions such as acknowledging the presence of users in a room.... Additionnaly, on connections, the content of the smartcard has to be synchronized with the content of the different groups it belongs to.

Disconnections may represent different kinds of events. It may be intentional, meaning that the owner of the card is leaving the place. Or it may be unintentional in case of resource problems (connection, battery exhausted ...) as it is common in mobile context (Chen and Kotz, 2000)

4.2 Making JavaCard active on the Web

As Cards may come and go, one must provide a simple asynchronous service to give the opportunity to other services to be aware of the connection or the disconnection of JavaCards. This may be achieved by giving the opportunity to the device hosting the card to inform the potentially interested services of those events. Those services may vary in time. Thus, they need to be able to suscribe and to be acknowledged of date modifications. Moreover, we want

to provide a distant access to the card, to make datas and Java Card Applets available. This is exactly the definition of the Observer pattern. This pattern or its extension, the Model-View-Controller pattern (Gamma et al., 1994) is based on logically asynchronous acknowledgement of model modification to Observers or Views. Those Observers can be added or removed at anytime. At the Internet level, Observers are translated as URLs to which the updates can be sent directly (see (Douin and Gilliot, 2003) for details). Controllers and Models can also be located on different hosts as shown in figure 3.

Figure 3. Basic collaboration for MVC

We have defined the following basic Views :

- At a first level, we propose to maintain a local list of servlets available on a hosting device, generally located on the device itself.
- A discovery service can be defined as a View of a set of machines, to propose servlets available on fly.
- By adding a card reader on every workstation in a classroom, defined as controllers, a model of all those machines can be aware of students whose JavaCard are currently connected. As an illustrative example, we have included a web server in an applet which can visualize all events from a Model as soon it is declared as View of this model. Because in our system, every device is a web server this feature is possible as long as the standard java applet is downloaded from the server of the Model to a traditional browser. A screenshot of this applet is shown in figure 4.

Figure 4. An applet as a View

Information of different servlets available on a device is published to every application listening to a Model. This way any reactive behavior (the Plug and Play effect) can be implemented in a distributed way. A collaboration diagram explaining the diffusion of events is given in figure 5.

This scheme is simple and lightweight to develop in a web environment as we explained in (Douin and Gilliot, 2003). This could be adapted to a full Publish and Subscribe communication scheme (Eugster et al., 2003) if necessary.

5. Applying Design Patterns to manage numerous devices

Usual applications are based on a few servers delivering datas or computation to many clients. When managing sensor or embedded device networks, we face many data providers to a possibly limited set of services/applications. Moreover, different schemes of communications must be handled, Client/Server is one of them, but it is useful to provide event-based communications.

In the previous section, we have proposed as a first step to install a MVC scheme in order to handle the event notification of asynchronous events. Yet, JavaCards and Java Card Applets still need to be organized to enable the management. For this, one need to adress logically synchronously sets of JavaCard or Java Card Applets. This is the purpose of the second pattern implementation presented here.

In this section, we will detail the use of the coupling of Composite and Visitor patterns to demonstrate how to define a structure for the system and a flexible way to cross the structure and enhance the collaboration between elements.

Figure 5. functional handling of connections

5.1 Structure your JavaCards and Java Card Applets: the Composite pattern

In order to structure our objects, we have to take into account the following points :

- exhibit the physical structure, such as the fact that a Java Card embed a set of Java Card Applets or that a device can connect several JavaCards at the same time
- enable logical grouping such as all Java Card Applets corresponding to a specific service, like each student's schedule of lecture
- give the possibility to add new basic or structural elements
- define the structure hierarchically as it is obvious in the first point

The Composite pattern is the classical hierarchical answer for the design of such structure. Our proposition in (Douin and Gilliot, 2003) is a distributed composite implementation with URLs as references for the objects constituting the structure. This enables full and lightweight interoperability with any connected device. In figure 6, we propose a composite structure adapted to Java Card, the left side of composite diagram exhibits the different basic elements constituting the system. Notice that a new element may be added by specializing existing elements to take into account new products. The specialization can be carried out on three main directions, either by taking into account the nature of the

service (agenda modification, quotation, ...) or the specificity of the device or by taking into account the selected architecture. The right side of the diagram, shows the possible combination of elements to construct coherent structures. Structures can be mixed to meet application needs. Notice that in this structure, a SmartCard is a collection of Java Card Applets.

Figure 6. Composite Structure for Smart cards

This means a more systematic way to aggregate different devices in an application.

5.2 Accessing elements of the structure: the Visitor pattern

To be useful, a structure like a Composite structure should be crossable in a systematic and uniform way. This is the goal of Visitor pattern. The purpose is to develop a visitor whose behavior is adapted to each object visited and ensures to visit every element of a structure. Hence it is possible to visit composite objects such as previous collections or other structures and final element in an uniform way. The Composite and Visitor patterns couple can be seen as an extension of Collection and Iterator patterns, just like a tree is an extension of an array.

In Fig. 7, we show a possible hierarchy of visitors, one being able to print the list of the elements in a console, the other gathering status datas in a html format. Services can then easily be extended by adding new Visitor classes. For ex-

ample, we have implemented several visitors, one of them checks the firmware version of javacards, another allows information from the configuration of the moment, yet another installs a new version of a servlet,

Figure 7. Visitors of Smart cards Composite

5.3 Managing the structure

As JavaCards may come and go, as well as environment may vary in time, different mechanisms must be installed to maintain the correctness of the structure. In a first approach, we have developed two basic complementary mechanisms

- Controllers of the MVC scheme collect informations of upcoming Java Cards. Structures can be defined either as models of the MVC (for example the local collections of Java Cards and corresponding Java Card Applets present on the device), and being directly updated, or as views (for example the room or the building containing the devices). Hence, in this configuration it is sufficient to install a view on a classroom structure to be aware of students currently connected and then to access to each of them. Active disconnections of JavaCards can be propagated the same way.
- As passive disconnections, possibly transient, may happen silently, we have defined a visitor who polls the structure in order to check if declared Java Card Applets/servlets are still currently connected. Different policies can be easily defined to remove definitely or not, information from the structure (checking them specifically for example to treat recovery).

6. Conclusion

In this paper, we have motivated the need for a lightweight integration of smartcards into distributed systems. We have shown that a loose coupling system with HTTP connections and small webserver at hosting level can be extended using basic design patterns as key for design. Thanks to this solution,

we have provided a plug and play mechanism as well as asynchronous and asynchronous collaborations. This is achieved in a structured vision of host and JavaCards managed.

Basic examples of potentiality have been exposed and could be adapted to other applications. For example, deployment of new services or other downloadings or updates are possible in a systematic way.

An on-line prototype is available at the following adress: <http://vivaldi.cnam.fr:8765/javacard/monitor/C100900000126496/Cardis04/>, where <http://vivaldi.cnam.fr:8765/javacard/monitor> is the url for the server, C100900000126496 the identity of the JavaCard and Cardis04 is a servlet/Java Card Applet embedded in the card.

Our model can be easily supported by any device able to embed a minimal Web server, thus being deployable in most infrastructures. As communications are based on HTTP protocol, it can interoperate with any other platform. For example, interaction with JiniCard and use of kSOAP are under investigation.

References

- Barber, J. (1999). The smart card url programming interface. In *1st Gemplus Developer Conference, Paris, France*.
- Biget, P. (1999). How smarcards can benefit from internet technologies to break memory constraints. In *1st Gemplus Developer Conference, Paris, France*.
- Brazil (2000). Brazil documentation and api, <http://www.experimentalstuff.com>.
- Chen, G. and Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.
- Donsez, D., Jean, S., Lecomte, S., and Thomas, O. (2001). (a)synchronous use of smart cards services using soap and jms. In *Gemplus Developer Conference*.
- Douin, J. and Gilliot, J. (2003). Collaboration patterns for networked embedded servers. Lisbon, Portugal. ETFA2003 9th IEEE International Conference on Emerging Technologies and Factory Automation.
- DS1957B (2004). Java powered ibutton.
- Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. (2003). The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2): 114–131.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison Wesley.
- Giorgio, R. D. (1999). An introduction to the url programming interface. *JavaWorld*.
- Giorgio, R. D. (2000). Serve clients' specific protocol requirements with brazil, part 1-4. *JavaWorld*.
- Guthery, S., Kehr, R., and J., P. (2000). How to turn a gsm sim into a web server. In *CARDIS 2000, Bristol*. IFIP.
- Inc., S. M., editor (2002). *Java Card 2.2 Runtime Environment (JCRE) Specification, chapter 8: Remote Method Invocation Service*, chapter 8, pages 53–68.
- Kehr, R., Rohs, M., and Vogt, H. (2000a). *Issues in Smartcard Middleware*, chapter Java on Smart Cards: Programming and Security., pages 90–97. Springer-Verlag LNCS 2041.

- Kehr, R., Rohs, M., and Vogt, H. (2000b). Mobile code as an enabling technology for service-oriented smartcard middleware. In *Proc. 2nd International Symposium on Distributed Objects and Applications DOA '2000, Antwerp, Belgium, IEEE Computer Society*, pages 119–130.
- Mascolo, C., Capra, L., and Emmerich, W. (2002). Mobile computing middleware. *Lecture Notes in Computer Science*, 2497:20–??
- Muller, C. and Deschamps, E. (2002). Smart cards as first-class network citizens. In *4th Gemplus Developer Conference, Singapore*.
- Rees, J. and Honeyman, P. (1999). Webcard: a java card web server. In *CARDIS 2000, Bristol. IFIP*.
- Vandewalle, J.-J. and Vétillard, E. (1998). Smart card-based applications using java card. In *Proceedings of the 3rd Smart Card Research and Advanced Application Conference (CARDIS 98)*.
- Vogt, H., Rohs, M., and Kilian-Kehr, R. (2004). *Middleware for Communications*, chapter 16, *Middleware for Smart Cards*. John Wiley and Sons.