

Smart Cards and Residential Gateways: Improving OSGi Services with Java Cards^{*}

Juan Jesús Sánchez Sánchez¹, Daniel Díaz Sánchez¹, José Alberto Vigo Segura²,
Natividad Martínez Madrid¹ and Ralf Seepold¹

¹ Universidad Carlos III de Madrid, Departamento de Ingeniería Telemática
Av. Universidad 30, 28911 Leganés, Madrid, Spain
{jjsanchez, dds, ralf, nati}@it.uc3m.es

² Sequel Business Solutions, Peninsular House, 30 Monument Street.
London EC3R 8LJ, United Kingdom
JVigo@sequel.com

Abstract. This article proposes an integration of Smart Cards into an environment controlled by a Residential Gateway. In a common scenario, the Residential Gateway offers services with different characteristics. Some services belong to profiles of a user and thus these services have a mobile behavior. As a consequence, these profile-related and thus user-specific services are configured via a Smart Card. The Smart Card serves as a medium easy to transport but it added more features to the scenario like the possibility of cryptographic services, secure payments for example for pay-per-view or environment's customization. The core of this work has been carried out in order to achieve an integration between two existing technologies: OSGi (Open Service Gateway Initiative) and Java Card.

1 Introduction

Nowadays, the application fields for residential gateways are increasing as network operators are facing an increasing need for technology that offer customers end-to-end services like triple-play [1]. These new services are ranging from telemedicine [2] to video-on-demand, thus residential gateways will become a key element in the information society.

Currently, a residential gateway is basically an embedded computer with two network cards one that connects to devices on a home network and the other to connect to the Internet. E-services as telemedicine imply that sensitive information is passed via a residential gateway, and this needs specific protection. Since it is not the core of this work to undertake risk studies, more information can be obtained at [3] and [4].

Smart Cards technologies have been proven to be a robust solution for security services based on their cryptographic capabilities and in their almost tamper-proof nature [5]. Furthermore, Smart Cards are suitable for storing users' sensitive personal

* This work is partly supported by the Spanish Ministry of Industry via the MEDEA+ projects TRUST-eS (A-306) and Planets (A-121).

information as medical records or private keys and furthermore, it is easy to carry them with you.

This work proposes an integration of Smart Cards into a platform for residential gateways. As an example for a highly flexible and open environment, the OSGi (Open Service Gateway Initiative) technology has been chosen as the target system. Also the fact that the Smart Card (Java Card) and the residential gateway support JAVA for programming, the integration task can be performed without a huge effort during the implementation of the interfaces. Once both technologies are integrated, several new functionalities as cryptographic services, secure payments for pay-per-view [6] or environment's customization [7] can be offered via the platform and thus it can be used independent from the service.

In the following section 2, an overview of relevant technologies and a brief study of related work is presented. In section 3, the work performed is detailed. The design phase and the different development strategies and test cases are documented. Once the feasibility is proven, in section 4 several practical applications are described. Also the implementation description is given here. Finally, section 5 presents the conclusion and the future work.

2 State of the Art

This section reviews the main aspects of Smart Card and Java Card technologies. In each case a brief description of principal features and applications is given. Also the OSGi technology is briefly presented, since it has been chosen as the platform for integration. At the end, relevant related work is referenced and summarized, while the difference to this proposal is stated as well.

2.1 Smart Cards

Smart Cards (defined in ISO-7816 standards [8]) are credit card size devices that are able to store and process information using an embedded integrated circuit. It is transmitting information to an external application. These cards do not only store data but they are able to protect them from unauthorized access or tampering. Typical applications are implementing security modules for banking services, mobile telephony or access control.

The smart card's CPU typically has 8 bits and compares to embedded processors its speed typically is slower, i.e. in the range of 10 MHz in case of the ST19XT34 MCU based smart cards [9]. The Smart Card has a ROM (96KB) and a RAM (4KB) memory. The first one is for the operating system and the second for user applications (EEPROM with 32KB or 64KB). Smart Cards have no interface to keyboards, monitors and other peripherals but they provide an I/O interface to communicate with a PC or a set-top box.

The communication between smart cards and external applications is realized via a protocol stack. Here, direct communication between the application within the card and an external application takes place. Control and data exchanges are application

specific. At the next (lower) level, the communication is based on the exchange of Application Protocol Data Units (APDUs) [10] these are used to convey commands and data from and to the card based application. In general, the communication model follows a master-slave architecture. The card takes over the *slave* role and waits for APDU commands from an external application. Once the card executes the instruction conveyed in the command, it sends the result of an APDU response.

2.2 Java Cards

The development of a smart card application is always tuned to a specific architecture since each card has a different internal behavior depending on the manufacturer, i.e. low-level communication protocols, memory management and hardware specific details may be different. The aim of Java Card [11] technology is to offer an interoperable high-level interface that should be available for Java Card application (written in Java and called applets) in any Java Card compatible smart card.

Java Card specifications include the Java Card Virtual Machine (JCVM) that defines the Java virtual machine and language suitable for Smart Cards, the Java Card Runtime Environment (JCRE) that describes the Java Card execution behavior (for example the memory management and the application management), and finally, the Java Card Application Programming Interface (JCAPI) that describes Java classes and packets available.

In the implementation of the current version 2.2.1 of Java Card and the Java language version several limitations apply, like: there is not dynamic class load neither a garbage collection, threads are not implemented and the packet `java.lang` is reduced and does not include `String`, `char`, `double`, `float` or `long` classes.

For this work, the Cyberflex Access 32K cards from Axalto [12] (former Schlumberger) have been used. All on-card applications (applets or cardlets) are Java Card 2.1.1 compliant.

2.3 OSGi

The Open Services Gateway Initiative (OSGi) is an independent, non-profit corporation working to define and promote open specifications for the delivery of managed services to networked environments, such as homes and automobiles. This initiative created an Open Service Platform Specification that defines the OSGi Service Platform, which consists of two parts: the OSGi framework and a set of standard service definitions. The OSGi framework, which is located on top of a Java virtual machine, represents the execution environment for services

The central component of the OSGi architecture is the Service Platform that works as the execution environment for services. It provides a platform that service providers can use as their own environment using the devices on a local network.

Initially, the targets for the OSGi specifications were digital and analogue set top boxes, Service Platforms and cable/DSL modems. As the standard has developed, it has first applications in consumer electronics, PC's, industrial computers, cars and

other areas where the benefits of uniform operating environments, hardware abstraction and service lifecycle management are appreciated.

The OSGi Framework handles the life cycle management of applications and components. It therefore provides the following functions:

- A packaging format for the applications: The OSGi specifications provide the Bundle format. Bundles are applications packaged in a standard Java Archive (JAR) file, which format is fully compatible with the ubiquitous supported ZIP files.
- Install a bundle: The bundle must be prepared and the diverse components installed in the OSGi Framework, ready to be executed.
- Start/Stop a bundle: Installed bundles can be started and stopped in an OSGi Framework. It is important to remark that in a Service Platform, all applications are started in the same JVM, thereby saving memory, resources, and CPU cycles.

Security issues in this framework are addressed only at bundle level by specifying three different security permissions for the bundles, *AdminPermission*, *PackagePermission*, and *ServicePermission*. The purpose of each of these permissions is to grant the authority to the bundle to carry out specific actions. OSGi tries to extend security issues beyond authorization policies and to add new functionalities to the framework.

For a detailed study of the specifications, it is recommended to read the OSGi specifications [13] and the global vision of the whole architecture that is completed in [14].

In this work *Oscar* [15] has been chosen as the implementation platform. Oscar is an open source implementation of the OSGi framework specification. Currently, Oscar is compliant to major components of the OSGi release 3 specifications. As a remark, "Oscar" was also the name of a smart card OS developed by GIS in the UK in 1989 that can be seen as a precursor to Java Card technology.

2.4 Related Work

Before starting with a detailed description of the implementation, other initiatives and proposals driven to take advantage of smart card potential in a residential gateway have been evaluated. As a result, two articles had been classified to be relevant: the first one is from the area of telemedicine and the second one focuses on authentication issues in Residential Access Networks.

In [16] an electronic-prescription system for home-based telemedicine is described. This article describes a health-prescription application running on a smart card that communicates with a Personal Digital Assistant (PDA). It uses OSGi as a central coordinating point among the devices. The OSGi environment is aimed to allow inter-communication between the card reader, the patient's PDA application and other devices but there is no detailed description about how the system is implemented neither on the specific security needs required for medical applications with respect to OSGi platforms.

The second approach [4] presents a quite different scenario in which authentication on network access is addressed. In this case, a smart card is used as a mere certificate

and key container inserted in the residential gateway. This article is focused on describing an authentication protocol in which smart card encryption and decryption capabilities are used. In this approach, the portability of data and configuration is less important for the application case.

3 Integration of OSGi and Smart Card

This section describes in detail the work carried out to create a bundle that adds new capabilities to OSGi gateways. With this new capabilities it is possible to design a Smart Card based application that takes advantage of all the functionalities offered by these devices like secure storage of data or cryptographic operations and thus to port configurations into different residential environments.

3.1 First Approach: MUSCLE Applet Loader Integration into an OSGi Gateway

In order to check the feasibility of this proposal, several tests have been made; the more complete was based in the MuscleCard Applet Loader [17] developed by MUSCLE (Movement For The Use of Smart Cards in a Linux Environment) project [18]. The original idea was to develop an OSGi bundle able to load the MuscleCard Applet into a Java Card.

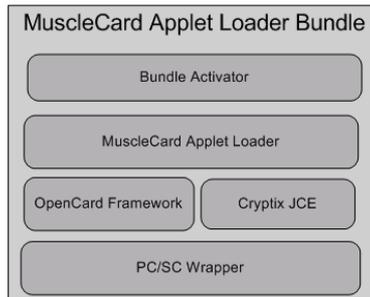


Fig. 1. MuscleCard Applet Loader Bundle structure

3.1.1 MuscleCard

MuscleCard can be conceptually divided into two parts: A cryptographic card edge definition for Java Cards describing the behavior and the protocol of a MuscleCard applet and an API for accessing Smart Card services. Together, they provide a powerful key and object storage solution on smart cards with cryptographic functionality. The range of applications of MuscleCard ranges from logon purposes to document signatures.

The complete specification of the API is defined in [17]. The MuscleCard applet protocol definition is presented in [19]. Additionally, a good overview of MuscleCard technology can be found in [20].



```
jjssanchez@CARUSO: /home/jjssanchez/Desktop/Trabajo/Oscar - Shell - Konsole
Session Edit View Bookmarks Settings Help
Welcome to Oscar.
=====
Enter profile name: OCF
INFO : 10 : Logger.start : 1.0.0
INFO : 10 : BundleEvent.STARTED
Uses ISOTPDU
Uses ISOTPDU
Address:
Name: E-Gate 00 00
Type: PCSC10
Slots: 1
Info for slot ID: 0
card present: yes
ATR: 3B 75 94 00 00 62 02 02 01
Insert your card ...
[DEBUG ] smartcard.OCF.factory.CFlexAccessCardServiceFactory.getCardType
--- message known card type opencard.core.terminal.CardID@e5855a ATR: 3B 75 94 00 00 62 02 02 01
--- thread Thread[main,5,main]
--- source class smartcard.OCF.factory.CFlexAccessCardServiceFactory
[DEBUG ] smartcard.OCF.factory.CFlexAccessCardServiceFactory.getClasses
--- message card type is 16
--- thread Thread[main,5,main]
--- source class smartcard.OCF.factory.CFlexAccessCardServiceFactory
[DEBUG ] smartcard.OCF.service.CFlexAccess32Loader.sendAPDU
```

Fig. 2. Execution of *MuscleCard Loader OSGi bundle* in an OSGi gateway.

3.1.2 MuscleCard Applet Loader

Inside the MUSCLE project, several applications and tools have been created in order to support the application of MuscleCard. The Java based MuscleCard Applet loader developed by Martin Buechler has been chosen to become the test OSGi bundle. With this loader it is possible to load the MuscleCard applet onto a smartcard. The different components of this application and their functionalities will be described with a higher level of detail in the subsection 3.2.

Since the Java application distribution includes all the necessary components and native libraries to be execute both in Linux and Windows environments, it is a stand-alone application suitable to be used as a stand-alone OSGi bundle.

3.1.3 MuscleCard Applet Loader OSGi Bundle Description

The created bundle's structure is depicted in figure 1. It consists of the Applet Loader components plus a BundleActivator that will execute the applet loader application. These elements are packaged with a manifest file into a bundle file that can be loaded by an OSGi gateway.

An example of the execution of this bundle is shown in figure 2. This figure is based on a screen snapshot showing how an Oscar gateway loads and executes the

bundle and how it establishes a communication with the smart card and loads the MuscleCard Applet.

This test also shows the feasibility of the planned implementation and thus the following step is introduced: the design of a specific bundle that serves as a library to others bundles.

3.2 OCFBundle Description

It is required to provide a bundle that serves as a communication bundle for on-card applications. This bundle will provide a global interface between OSGi applications and their respective Java Card applets.

Therefore, this new bundle allows the communication between applications running on a gateway (as OSGi bundles) and applications running in a Smart Card.

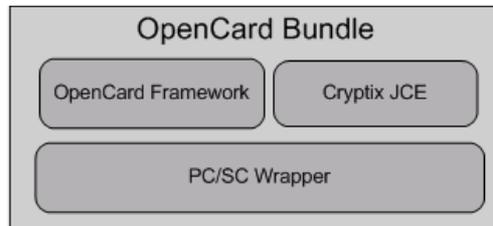


Fig. 3. *OpenCard bundle composition*

The bundle will be called *OCFBundle* (OpenCard Framework Bundle) and it is based on the basic components of the Applet Loader Bundle. The following components are part of the bundle:

1. **OpenCard Framework block:** The OpenCard Framework provides a common interface for both the smart card reader and the card's application. It was created as a standardized framework for implementing Smart Card enabled solutions and Smart Card based services. It provides an open architecture and a set of common APIs (Application Program Interfaces). Also CardTerminal and CardService concepts are defined. Both of them are specifically implemented for a particular Smart Card based application. The former is an API to access to the card and reader through standardized protocols such as ISO7816-3 (i.e., T=0 or T=1 protocols) and the latter is an application that uses such an API.
Any Client Bundle developed will use these functionalities for communication with applications inside the smart card.
2. **Cryptix JCE block:** The Java Cryptography Extension (JCE) [21] is a set of packages that provides a framework and implementations for encryption, key generation, key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects. For this develop-

ment, Cryptix JCE [22] has been chosen since it is a complete open-source implementation of the official JCE API published by Sun. Through this API a Client Bundle is able to fulfill the cryptographic operations needed to establish a secure channel between the application and the applet in the card.

3. PC/SC Wrapper block (PC/SC CardTerminal): This wrapper [23] was initially developed by IBM and Gemplus in order to integrate PC/SC in OpenCard Framework [24]. Through this wrapper, an OpenCard based application can be used operating the system's PC/SC capabilities to establish communication with an on-card application.

Finally, the new bundle has been tested on an Oscar 1.0.5 OSGi Gateway running on a Java Runtime Environment 1.5.0_04. The tests were successful, but adjusting and tuning processes are ongoing.

4 Applications

In this section, several applications of the proposed architecture are described. In all cases, new functionalities are added to the OSGi Framework, and the majority of these functionalities are related to security issues.

The communication with Smart Card devices is carried out through a further OSGi bundle, the integration of smart based applications can be done seamlessly when designed it as a Java architecture.

An example of how OSGi can improve its functionalities by adding a PC/SC wrapper bundle is depicted in Fig. 4. There are many possible applications fields but only three functionalities are proposed for implementation:

- Environment customization. This has been the first practical application of the proposed architecture; here, smart cards have act as a mean of storing and managing user's preferences in different environments. This application is detailed in the following subsection.
- User's authentication. Smart Cards are able to store users' certificates and private keys and thus they have been widely used as authentication means. With this extension an OSGi bundle is able to check a user's identity.
- Pay-per-view, Micro e-payments and DRM. These three applications fields are currently under investigation by different research centres or companies' alliances. It is foreseen that this technology targets at a growing market.

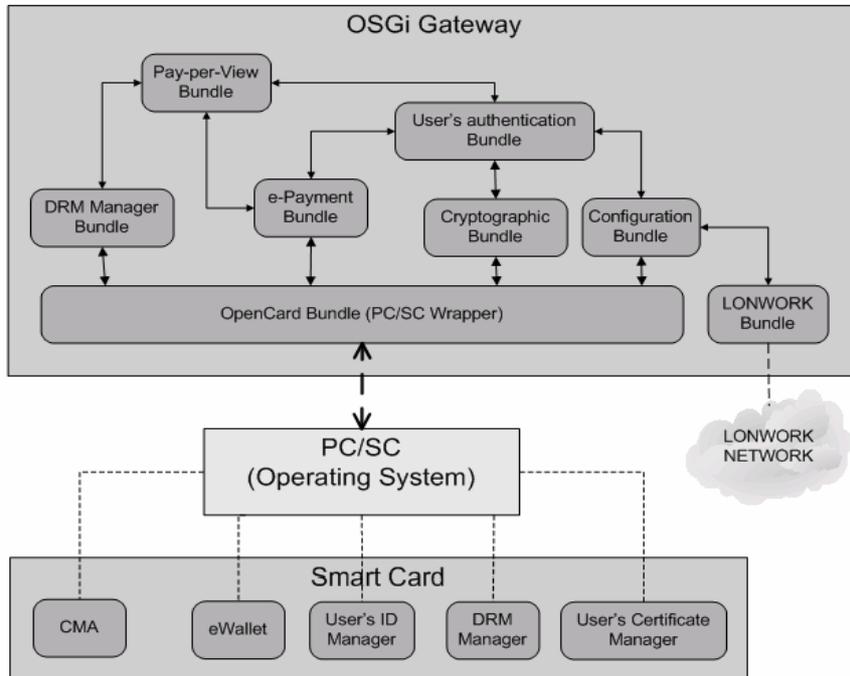


Fig. 4. Architecture to integrate Java Card based applications into OSGi Framework

4.1 Environment's Customization

This application is based on a previous work [25] in which an architecture for customizing automation was proposed. A possible architecture for a customizable automation environment was described, in such a way that environment's configurations for each user depend on the information stored in a Smart Card device.

Following this scheme, a user will be able to configure and control devices that are connected to the network with the help of a smart card. Many different scenarios are possible; with the help of a Smart Card, a user can control the power of the home lights, the maximum and the minimum temperature of the room's heating, the blinds behavior, preferences in the TV program, etc.

Originally, the proposed solution is based on a specific type of control networks from LonWorks ([26], [27]), which provides a distributed, powerful and open architecture in order to control and manage any kind of sensors and actuators. In this approach, the on-card application, the so-called CMA (Configuration Manager Application) manages the different user's configurations via applets.

When the card is inserted, the gateway has to select the right application to operate with the card (one card can store different applications). Once the CMA application is successfully selected, it is possible to establish a communication between the program running in the gateway and the on-card application as it is shown Fig 4.

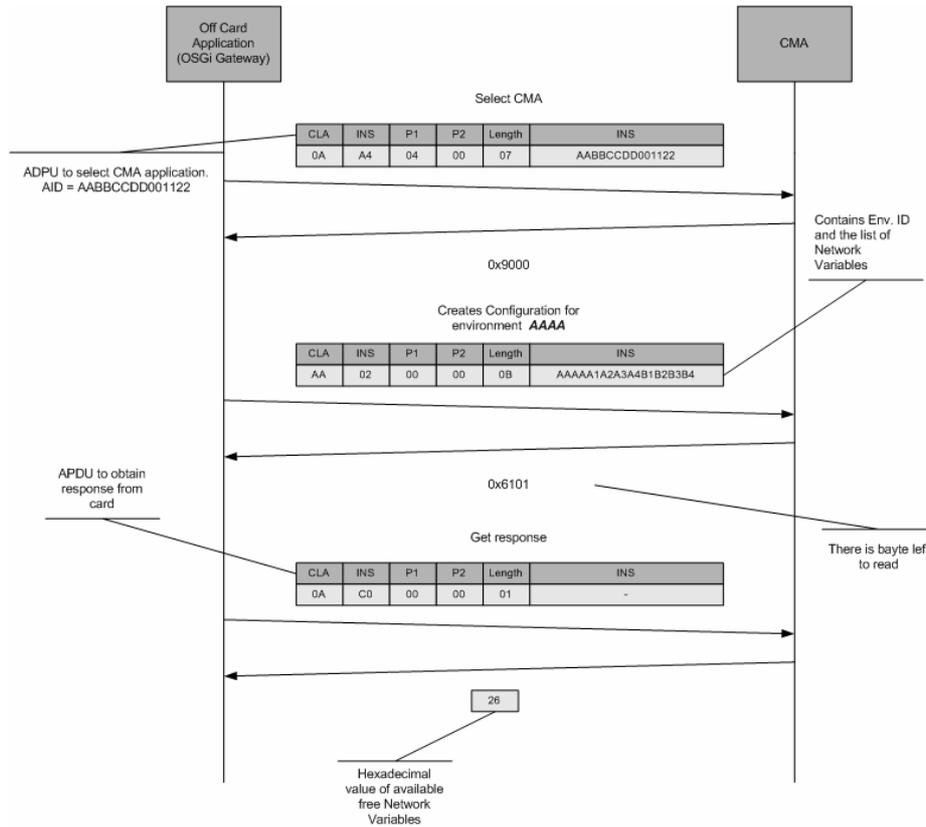


Fig. 5. APDU exchange example for CMA application

4.1.1 CMA Functionalities

In this example, the CMA is a Java Card applet which is responsible element to manage all possible configurations in different environments.

For each given environment, a configuration can be defined as a set of values for different variables: each device can be represented via a group of variables. The values of these variables will define a device's status and the configuration of an environment is made up by the set of all devices' status.

In the implementation, the CMA stores and manages several configurations for the same environment and these configurations may consist of different sets of variables. That implies that when a new configuration is used to set up a specific environment, non-updated variables keep their current values.

The CMA can manage configurations for different environments. This allows users (a cardholder) to have their private configuration for different environments (home, office, car,) stored in the same portable device.

In Figure 6, the different operations available for the current CMA version are shown.

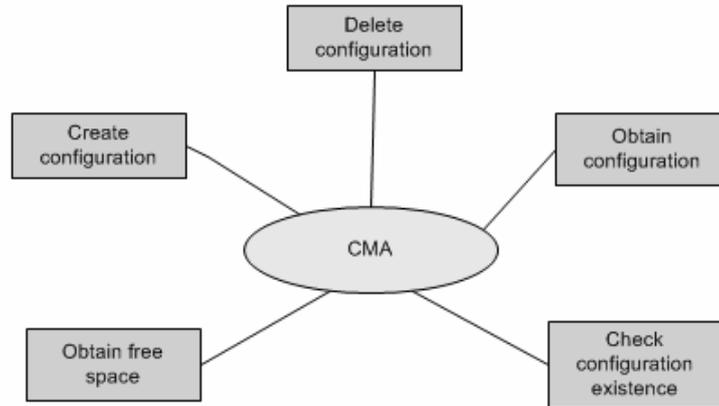


Fig. 6. Current *CMA* available *operations*

As shown below, current CMA implementation allows carrying out simple management operations with stored configurations but future efforts will be aimed to add new functionalities.

4.1.2 Configuration Bundle

This OSGi bundle will be the one that uses the configurations stored inside the smart card to set up the environment where it has been deployed. That implies of course that any given environment must have an OSGi gateway as one of its core elements since it is the “more intelligent device”.

A configuration bundle communicates with the CMA applet through OCFBundle, whose API is used to compound the messages (APDUs) sent to the CMA and to receive its responses. With these messages the bundle is able to obtain configurations stored in the card and to make the required changes in the environment. Besides, it is possible to create a new configuration from the current status of an environment and to store it.

The current version of this bundle is limited and thus it applies first configuration found. It is planned to develop a more complex bundle that includes a Graphic User Interface (GUI). Once the GUI is finished, can a user select among different configurations that apply.

4.1.3 LONWORK Bundle

The LONWORK bundle’s purpose is to act as an interface between the Configuration Bundle and the LONWORK network, in such a way, that the Configuration Bundle is able to control these devices. Currently, this bundle is in a conceptual status but it is

expected to launch the development soon in order to test the whole system connect to an industrial network. Then, other bundles for different automation networks can be created, and this will enable to control more complex environments within different automation networks.

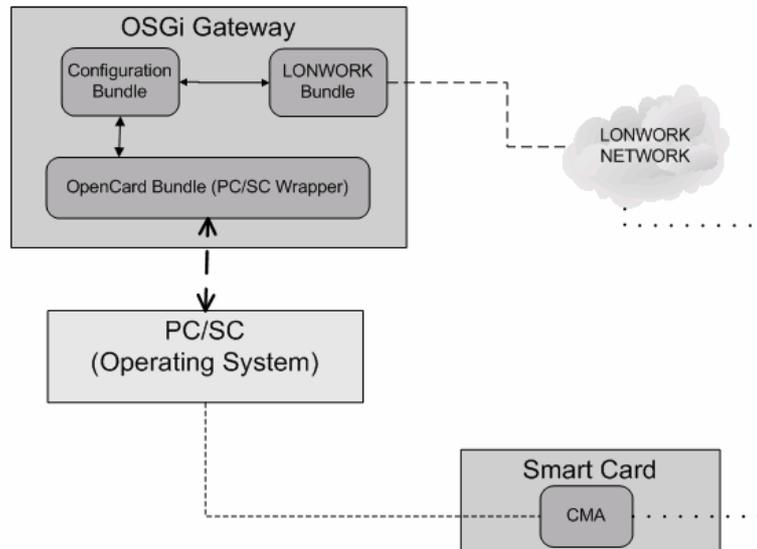


Fig. 7. Current CMA available operations

5 Conclusion and Future Work

This article presented a proposal for an integration of smart cards into an OSGi environment. Besides the description of the design and the implementation, several applications have been described. Furthermore, the OCFBundle feasibility has been successfully proven and additional applications have been proposed.

It is planned to offer this bundle to the Oscar Initiative for further discussion and possibly to include it as a regular bundle in the distribution.

Currently, CMA functionalities and Configuration bundle improvements are almost finished. While the implementation of the LONWORK connection is still under development. Since both projects (TRUST-eS and Planets) are running, it is planned to perform the remaining tasks with the help of the regular work flow.

Finally, the convenience of abstracting the CMA application in the OSGi framework as service will be considered. With this service a regular Java interface that hides the implementation details will be provided, in such a way, that any other CMA implementation could be used.

References

1. D. Ladson: Building a next-generation residential gateway means making tough Odesign choices. Texas Instruments, Wireless Net DesignLine July 2005, at www.embedded.com/showArticle.jhtml?articleID=166402757
2. P. O. Bobbie, S. H. Ramisetty, A. Yussiff and S. Pujari: Designing an Embedded Electronic-Prescription Application for Home-Based Telemedicine Using OSGi Framework. Embedded Systems and Applications, Proceedings of the International Conference on Embedded Systems and Applications, ESA '03, June 23 - 26, 2003, Las Vegas, Nevada, USA.
3. A. Herzog and N. Shahmehri: Towards Secure E-Services: Risk Analysis of a Home Automation Service. Proceedings of the 6th Nordic Workshop on Secure IT Systems (NordSec). Copenhagen, Denmark. Pages: 18-26. November, 2001.
4. J. Rossebo, J. Ronan and K. Walsh: Authentication Issues in Multi-Service Residential Access Networks. 2003. Proceedings of 6th International Conference on Management of Multimedia Networks and Services MMNS'2003, LNCS 2839, pages 381-395, Belfast, UK.
5. D. Naccache and D. M'Raihi: Cryptographic Smart Cards. IEEE Micro, pp. 14-24, 1996.
6. J. Domingo-Ferrer, A. Martínez-Ballesté and F. Sebé, Francesc: MICROCAST, Smart Card Based (Micro)Pay-per-View for Multicast Services. CARDIS 2002: 125-134
7. Vigo Segura, J.A., J.J. Sánchez Sánchez, N. Martínez Madrid and R. Seepold: Profile-based configuration of residential networks. ISBN 8-489315-43-4. EUNICE 2005, Networked Applications, 11th Open European Summer School, Colmenarejo, Madrid (Spain).
8. International Organisation for Standardisation (ISO): ISO/IEC 7816. Identification cards - Integrated circuit(s) cards with contacts. Available at www.cardwerk.com/smartcards/smartcard_standard_ISO7816.aspx
9. STMicroelectronics: ST19XT34 Brief Data, Smartcard MCU with MAP, USB/ISO Interface & 34 Kbytes High Density EEPROM. September 2003.
10. International Organisation for Standardisation (ISO): ISO 7816-4: Interindustry Commands for Interchange. Available at www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4_7_transmission_interindustry_commands.aspx
11. Sun Microsystems, Inc: Java Card™ Specifications Version 2.2.1. October, 2003. Available at http://java.sun.com/products/javacard/RELEASENOTES_jcspecs.html
12. Axalto: Cyberflex Access Cards Programmer's Guide. Axalto 2004
13. The Open Services Gateway Initiative Alliance: OSGi Service Platform, Release 3. OSGi 2003. This specification can be downloaded from the OSGi web site: www.osgi.org
14. The Open Services Gateway Initiative Alliance: About the OSGi Service Platform Technical Whitepaper, Revision 3.0. 2004. Available at www.osgi.org/documents/osgi_technology/osgi-sp-overview.pdf
15. Oscar, An OSGi framework implementation. Available at oscar.objectweb.org
16. P.O. Bobbie, A.L. Yussiff, S. Ramisetty and S. Pujari: Designing an Embedded Electronic-Prescription Application for Home-based Telemedicine Using OSGi Framework. Proceedings of the 2003 International Conference on Embedded Systems and Applications (ESA'03), Eds. H. R. Arabnia and L. T. Yang, Las Vegas, June 23-26, 2003, pp. 16-21.
17. D. Corcoran and T. Cucinotta: MUSCLE Cryptographic Card Edge Definition for Java Enable Smartcards. MUSCLE 2001. Available at www.linuxnet.com/musclecard/files/mcardprot-1.2.1.pdf
18. MUSCLE - Movement for the Use of Smart Cards in a Linux Environment Home-Page: linuxnet.com

19. D. Corcoran and T. Cucinotta: Musclec card Framework Application Programming Interface. version 1.3.0, MUSCLE 2002. Available at www.linuxnet.com/musclec card/files/muscle-api-1.3.0.pdf
20. O. Karsten: MuscleCard. Available at www.inf.tu-dresden.de/~ko189283/MuscleCard/MuscleCardArticle.html
21. Sun Microsystems, Inc.: Java™ Cryptography Extension (JCE) Reference Guide for the Java™ 2 Platform Standard Edition Development Kit (JDK) 5.0. 2004. Available at java.sun.com/j2se/1.5.0/docs/guide/security/jce/JCERefGuide.html
22. Cryptix JCE Home-Page: www.ntua.gr/cryptix/products/jce
23. PC/SC Wrapper CardTerminal Home-Page. www.gemplus.com/techno/opencard/cardterminals/pcsc/doc/README-PCSCWrapper.html
24. OpenCard Home-Page. www.opencard.org
25. J.A. Vigo Segura, J.J. Sánchez Sánchez, N. Martínez Madrid and R. Seepold: Integration of Smart Cards into Automation Networks. IEEE Catalog Number 05EX1101, ISBN 3-902463-03-1. WISES 2005. Hamburg (Germany).
26. Echelon Corporation. Introduction to the LonWorks System, 1999. Available at www.echelon.com/support/documentation/manuals/078-0183-01A.pdf
27. LonMark Association. LONMARK Application-Layer Interoperability Guidelines version 3.3, 2002. Available at <http://www.lonmark.org/products/guides.htm>