Leveraging Object Linking and Embedding for Process Control Unified Architecture Standards with Smart Card Technology for Secure Applications and Services

Yuankui Wang (Matr.-Nr.: 6670785)

University of Paderborn wangyk@mail.upb.de

Abstract. Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the most recent released industry standard from OPC Foundation, which compared with his predecessors is equipped with a list of charming new features, with whose help OPC UA is capable of developing a common communication interface for devices which participate in automation system.Meanwhile, the technology of smart card is widely used in information security fields of finance, communication, personal and government identification, payment. Therefore it is meaningful and promising to develop OPC UA standard compliant application on embedded smart card secure device, for the purpose of secure remote control, enterprise resource planning and etc. Since the storage and compute capacity of chip card is limited, OPC UA product will consist of two essential parts, namely client/server application code, realized as Android or other application, and communication stack, realized as Javacard Applet based on Remote Application Management from GlobalPlatform. The implemented demonstration scenarios and corresponding analysis show the possibility of developing OPC UA standard compliant application on devices embedded with smart card to benefit customers.

1 Introduction and Motivation

According to the *Mobile Economy 2013* from *Global System for Mobile Communications Association*, at the end of year 2013 there are over 3.2 billion mobile subscribers in total, which means one half the population of the earth now enjoy the social and economic convenience brought by mobile technology. Moreover by year 2017 700 million new subscribers are expected to be added. And the number of mobile subscriber will reach 4 billion in 2018. Mobil technology opens nowadays a promising market.

Mobile products play an irreplaceable role at the heart of our daily life. With the help of mobile technology, the user's world in many domains such as, education, financial transactions, health and etc. are inter-connected. Mobile users are enjoying the advantages of mobility. Services, like 24/7 monitored home security, full control over the management of home humidity and temperature, exist not only in science fiction film but also could be realized by today's technology.

At the same time, mobility in industry and business world is also a critical assert, which can not only increase efficiency and productivity but also drive new revenue generation and competitive advantage. The most convicting example here is Machine to Machine communication, that is also referred as M2M technology. In M2M communication, machines which are usually embedded with smart cards exchange gathered date with each other to accomplish common task using wireless or wired networks. M2M technology is widely employed in different industry spheres such as factory automation, remote access control and sensor monitoring. It boosts the efficiency of corresponding processes, offers centralized service support and date management, minimizes system response time.

But in order to enjoy the aforementioned features, two tough issues must be resolved. First, how to achieve a common interface for the devices that build the system. And second how to guarantee system security under different communication environments with various date complexity and customer's requirement.

1.1 Solution Idea

In this master thesis, I am going to address solutions for questions mentioned in section *Introduction and Motivation* and design a smart home system for the purpose of demonstration. In this smart home system, home owner using smart phone is capable of experiencing 24/7 home security service, remotely managing inner home environment parameters and assigning access permissions. This system consists of smart phones with Universal Integrated Circuit Cards (UICC smart card), digital door locks, electronic devices (such as coffee maker) and environment sensors. Moreover each device is equipped with smart card, which acts not only as secure token, that saves user credentials, but also is in charge of communication management with other devices.

In particular, I will introduce the newly released industry automation standards object linking and embedding for processes control unified architecture(OPC UA standards) to build a common communication interface for devices that are mentioned above and design a OPC UA specification compliant communication stack on UICC smart card., whose duties are: creation and management communication between OPC client/server application, entity authentication and secure message exchange.

1.2 Paper Structure

At first, in the second chapter I will present the fundamental technologies which will be frequently mentioned in this paper. Secondly the state of art, for instances mobile security, home remote control technologies, Remote Application Management from GlobalPlatform will be introduced, which act together as cornerstone for my implementation scenario. In the fourth section, I am going to focus on UICC mobile security and base on UICC framework build a OPC UA standards scarifying communication stack as Javacard Applet with the help of GlobalPlatform specifications. In the next implementation chapter, I will present how my demonstration scenario can be realized. As sixth and seventh chapter, test and

performance analysis will be describe to show the reliability and security of my proposal.

2 Fundamental Technologies

In this section, i am going to give a brief introduction about technologies and terminologies that are applied in this paper.

2.1 OPC Unified Architecture Structure Overview

Object Linking and Embedding for Process Control Unified Architecture, known as OPC UA is the most recent released industry standards from OPC Foundation, acts nowadays as the most promising candidate in industry M2M automation world, whose major duty is to build a secure communication interface for machines that participate in automation system.

OPC UA Specifications The whole OPC Unified Architecture specification can be divided into three main parts, core specification part, which consists of OPC UA concepts, security model, address space model, services, information model, service mapping and profiles, access type specification part including date access, alarm and conditions, programs and historical access, at last utility specification part covering discovery together with aggregates.

Table 1. OPC UA specifications

OPC UA Part1	Overview and Concepts Specification
OPC UA Part2	Security Model Specification
OPC UA Part3	Address Space Model Specification
OPC UA Part4	Services Specification
OPC UA Part5	Information Model Specification
OPC UA Part6	Mappings Specification
OPC UA Part7	Profiles Specification
OPC UA Part8	Data Access Specification
OPC UA Part9	Alarms and Conditions Specification
OPC UA Part10	Programs Specification
OPC UA Part11	Historical Access Specification
OPC UA Part12	Discovery and Aggregates Specification

OPC UA Client Server Structure OPC UA standards apply the classic client server architecture, where server is in charge of managing functionalities provided by a machine as well as data information gathered by that device, for instance temperature data measured by a remotely allocated sensor and the make

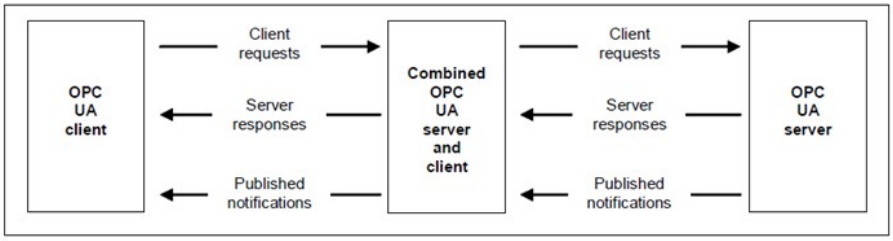

Fig. 1. OPC UA Client Server Structure[1]

coffee function offered by coffee maker. Meanwhile client possesses the ability to query information from server, submit subscription and send command to server.

Figure 1 illustrates a typical OPC UA client server architecture and also describes an internal combined server-client structure. The routine communication between client and server consists of requests from client, corresponding responses sent by server and notifications which are generated because of client's early subscription.

OPC UA Terminologies In OPC Unified Architecture on server stored information that can be visited by clients is defined as *address space*[2] and there also exits a set of services[3] which are provided by server and are introduced in order to apply operations on *address space*. Information in address space is organized as a set of in particular hierarchy structured *Objects*. *Object* here could refer to data gathered by sensor, server system parameters and etc. Clients can query and accept information provided by OPC Unified Architecture Servers in two major ways, *binary structured data* and *XML documents*, depending on the complexity of exchanged date, network quality and so on. In addition three kinds of transport protocol are already defined to support client server communication. They are: *OPC UA TCP*, *HTTP/SOAP* and *HTTP*. Also the hierarchy structure in which *Objects* are organized in *address space* is also various and not only limited to simple single hierarchy.

One of the charming features provided by OPC UA is *Event Notifications*. With the help of *Event Notification*, OPC UA servers are allowed immediately after satisfaction of conditions, which is normally predefined by a client, to publish data to particular client. In this way, clients can for instance discovery failures within client-server-communication quickly and recover that communication as soon as possible, which in return minimizes the lost to the smallest possible amount. Also clients are able to observe the subscribed data more precisely and find the pink elephant as fast as possible.

Standard OPC UA Server In figure 2, the structure of one standard OPC UA Server is described. It includes three main parts, server application, internal API

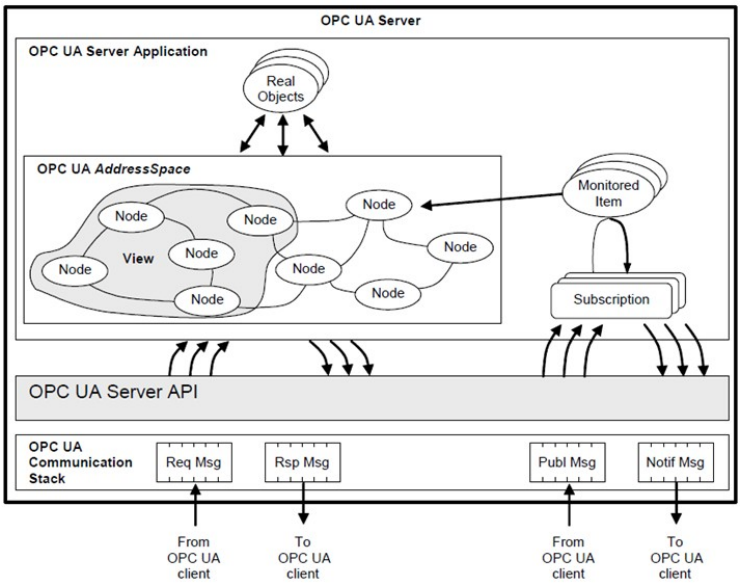

Fig. 2. OPC UA Server Structure[1]

and communication stack. In server application part, functionalities and services which are offered by OPC UA standards are realized, such as *Event Notification* , processing request from connected OPC UA client, data encryption and decryption. Moreover, *Real objects* here refers physical field devices and software applications that are maintained and managed by OPC UA server. *Nodes* in *Address Space* presents abstractly above mentioned *Real Ojbects*. *View*, which is pictured as a part of address space, presents *Objects* that can be browsed by particular clients. The main task for communication stack is to establish communication session based on secure channel between OPC UA client and server. Typically communication messages which are exchanged frequently among clients and servers are, request-, notification- message from client and corresponding response-, publish message from server. At last, an internal API connects the server application and the communication stack.

Standard OPC UA Client Figure 3 pictures one simple OPC UA client containing client application, an internal API, isolating the application code from communication stack, and a communication stack that converts API calls into messages and delivers them to OPC UA server.

Secure Channel and Session Since some data exchanged between client and server could be extreme precious and should be protected from other malicious third party, OPC UA defines a full set of *security model*, with which sytem developer can configure the security level of the application to meet the need of reality.

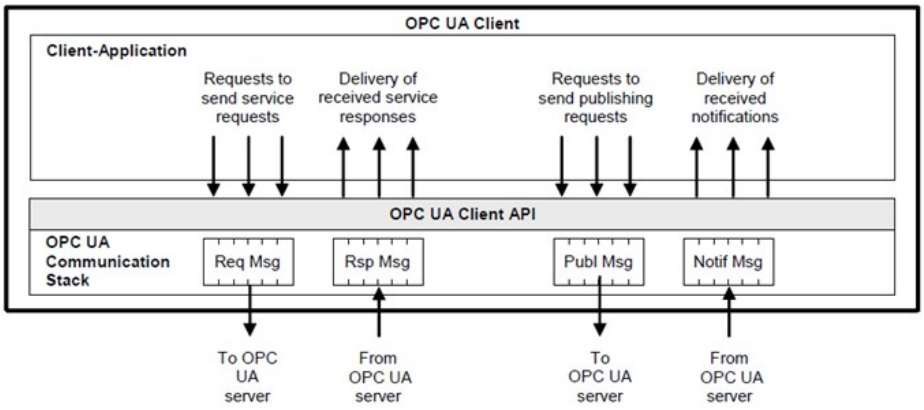

Fig. 3. OPC UA Client Structure[1]

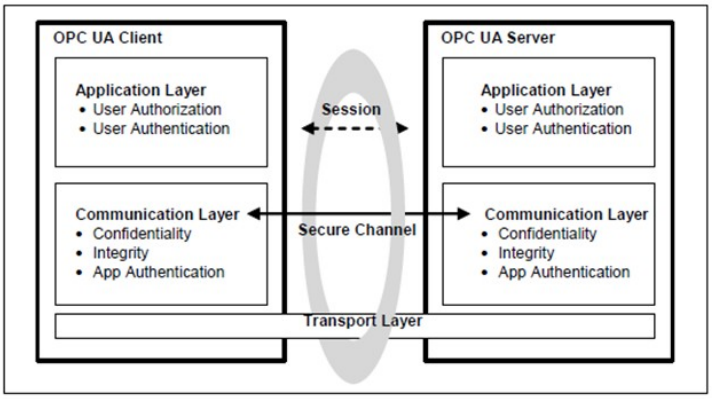

Fig. 4. OPC UA Client Server Communication[4]

In the *security model*, authentication of client and server, authorization, integrity and confidentiality of client-server-communication, auditability(also known as traceability) and availability of services are guaranteed. Also OPC UA standard provides a set of countermeasures against attacks such as message flooding, eavesdropping, message spoofing, message alteration, message reply, server profiling, session hijacking and so on[4].

Figure 4 pictures the typical communication architecture between OPC UA client and server. As shown in 4, the communication between OPC UA client and server is established above a secure channel, which is active during the whole application session and in this session, the state information, such as algorithms used for authentication, user credentials, is maintained. The secure channel is established only after successful validation of both client and server certificates and it provides necessary mechanisms to support confidentiality, message integrity and application authentication. On top of secure channel, is an application level session between OPC UA client and server, whose responsibility is to transmit data information and commands. It should be pointed out that, even a secure channel is out of work for some reasons, the session is still valid and OPC UA client and server involved in aforementioned session can still re-establish the broken secure channel. A secure transport layer is guaranteed by encryption and signatures methods provided by platform that supports OPC UA structure.

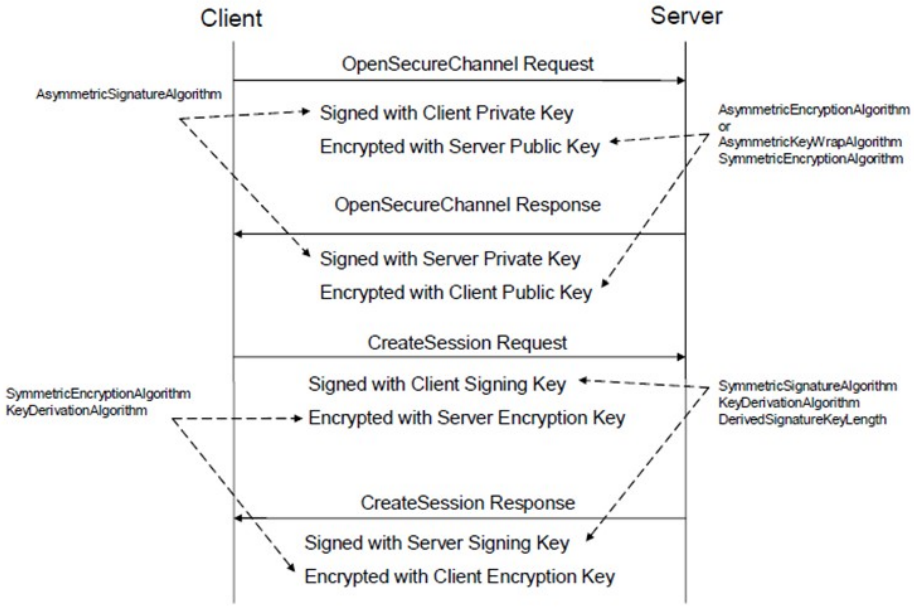

Fig. 5. OPC UA Client Server Security Handshake[4]

Security Handshake Security handshake as below explains with some details how secure channel and session are established. Normally OPC UA client initiates the first *OpenSecureChannel* request and waits the response from server. Messages exchanged during the process of construction secure channel between client and server are encrypted using asymmetric encryption and signature algorithms. But some security protocols that could be applied according to OPC UA standards, are not using an asymmetric message encryption algorithm to encrypt to request/response messages. Instead, they apply AsymmetricKeyWrapAlgorithm to encrypt symmetric keys and use symmetric encryption algorithm with encrypted keys to encrypt messages. After a successful construction of secure channel, OPC UA client sends *CreateSession* request and waits for server response. Messages transported during this procedure are encrypted with symmetric encryption algorithms and signed with client/server signing key.

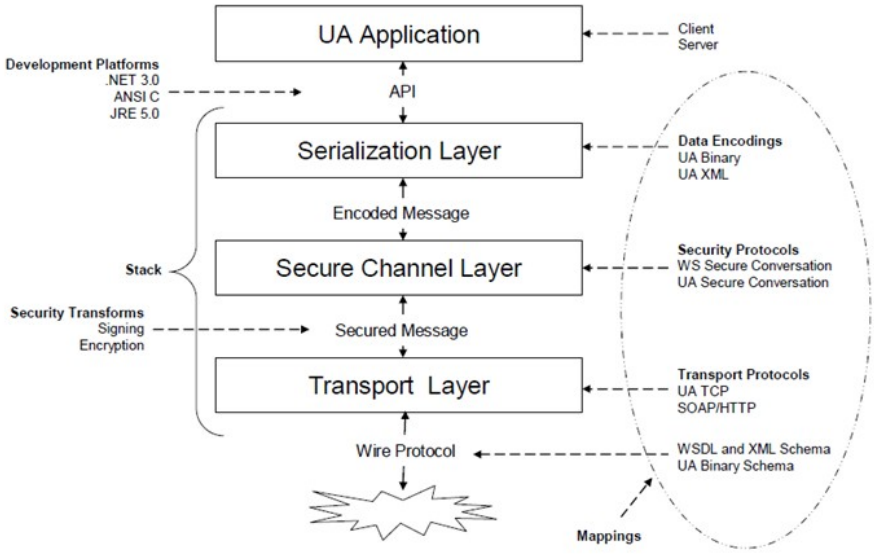

Fig. 6. OPC UA Client Server Communication Stack[4]

OPC UA Communication stack As described in figure 6, according to different responsibility, OPC UA communication stack consists of three parts: application layer, communication layer and transport layer. Even the terminologies of those layers using the same English words as the ones used in ISO model,

but they are not equal to layers in ISO model. Figure 6 also pictures a precise functionality overview of each component.

UA Application part realizes client or server application. Serialization layer together with secure channel layer build the communication layer and their job is dividing long message into pieces referred as message chunk, encrypting each individual message chunk and forwarding encrypted message chunk to transport layer. When receiving message chunk from others, OPC UA message receiver firstly verifies whether this message piece meets the security standard negotiated between OPC UA client and server. If not, message receiver will close the secure channel. After a successful verification of all message chunks, the original OPC UA message will be reconstructed and sent to UA Application Code through API. Each secure message chunk applies the following structure described in figure **??**.

Historical Data Last but not least security feature offered by OPC Unified Architecture is auditing, which supports traceability of any behaviours occur in OPC UA system. This record can be used for forensic research.

Other Competitor WebSphere Message Broker Message Queuing Telemetry Transport (MQTT)[5] is another machine to machine (M2M) communication protocol. Compared with OPC UA standard, MQTT also supports UDP protocol in the transport layer. In OPC UA, only unidirectional, client to server, communication is provided, but in MQTT server to client communication is also possible without server implements client code. Moreover the communication overhead of MQTT is in comparison with OPC UA is relative small.

Even thought MQTT protocol supports communication environment with low bandwidth and high latency, OPC UA provides complex object model and supports more features, including historical data record, alarm, notification, complete security policies and this is reason why OPC UA is more suitable for the application scenario that handles sensitive data with complex structure and needs immediate response.

Another member from Internet of Things is Constrained Application Protocol (CoAP)[6] which is designed for the extreme simple electronic devices with less memory and computing power and original CoAP only runs over UDP. Compared with OPC UA, simplicity from CoAP is the advantage, but apparently it should be considered that in the implementation scenario other transport protocol could be used, like TCP, more functions and services other than pure message exchange between client and server, are requested from users.

2.2 UICC

The Universal Integrated Circuit Card is the smart card used in mobile terminals in GSM and UMTS networks. It enables authenticated subscriber to join the network with their mobile terminals and at the same time protects essential user data. UICC acts also most time as the secure token, which stores and protects

subscriber's confidential information. Moreover, as a 32bit processor, UICC is also capable of processing necessary encryption and decryption algorithms[7].

Smart Card Smart Card, whose characteristic feature is an integrated circuit that is embedded in a chip card, which is capable of performing data process, information storage and message transmitting[8]. The most charming feature of smart card is that, sensitive user's credential data such as certificates, encryption keys, digital signature along with other precious user information can only be accessed though a serial interface, which stands under strict control of the card operation system. This characteristic provides strong protection against unauthorized data access and ensures the confidentiality of on card stored information. Therefor smart cards are widely used in applications that require strong protection.

With sophisticated communication protocol using Application Protocol Data Units(APDU), smart card and Card Accepting Device(CAD) are able to process secure message exchange. Smart card is also able to process cryptographic algorithms on hardware. Nowadays, it supports symmetric key algorithms like DES, triple DES; standard public key cryptography for instance RSA, hash functions such as commonly SHA-1[8]. More powerful microprocessor on chip card is, the speed performance is better.

Table 2. ISO/IEC 7816[8]

ISO7816 document	Description
ISO 7816-1	Physical characteristics
ISO 7816-2	Dimensions and location of the contacts
ISO 7816-3	Electronic signals and transmission protocols
ISO 7816-4	Industry commands for interchange
ISO 7816-5	Number system and registration procedure for application identifiers
ISO 7816-6	Interindustry data elements

In ISO/IEC 7816 standards family, the smart card's fundamental properties and functionalities are defined.

Smart Card Software Components As illustrated in figure 7, one typical smart card software includes Card Operation System, native services such as I/O operation and memory management, Java Card Runtime Environment(JCRE) that consists of Java card Virtual Machine and Framework who is in charge of dispatching APDU as well as applet management, installed applet and at last other optional industry specific extensions[9]

Smart card File Management One of smart card's characteristics is data storage media. But in contrast with other storage device, the most distinguishing

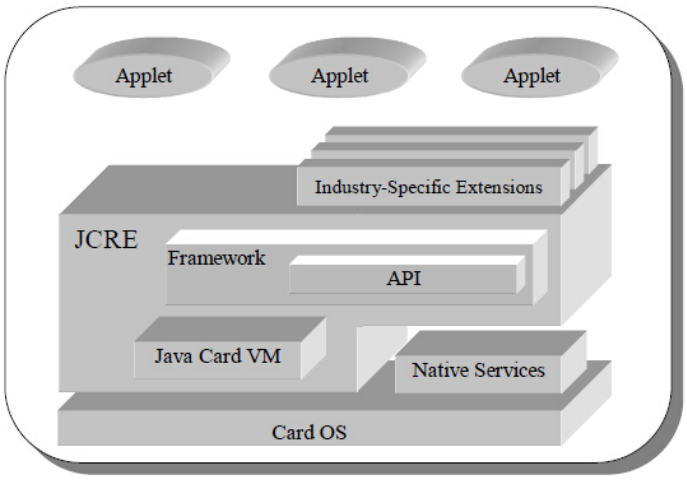

Fig. 7. Smart Card Software Components[9]

feature of smart card file system is that, there exists no man-machine interface[8], which means all files are addressed with help of hexadecimal codes and every single file process command is strictly based on this shema.

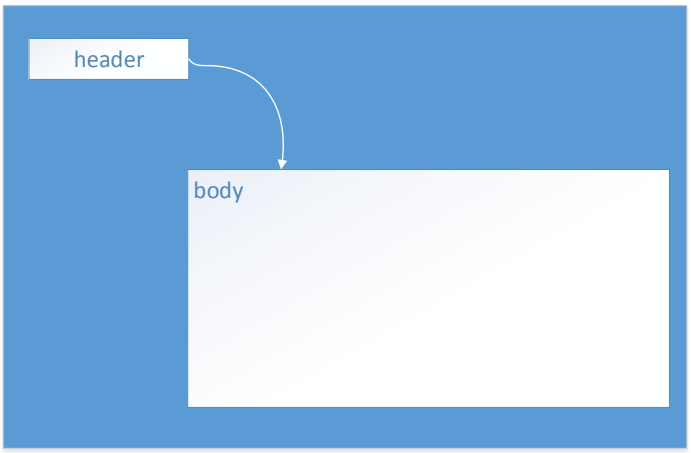

Fig. 8. the internal structure of a file in smart card file management system[8]

File Structure As pictured in figure 8, files on smart card consist of two parts, the header, which encapsulates administrative information such as, file structure and access conditions, the body, that stores real user data and is linked with file header using a pointer. This file management mechanism has its own advantage. To be more specifically, since file header and body are separately located, therefore even write/read error occurs in file body, file header, which is under normal circumstances never altered and saves essential access conditions, will not be affected, which in return provides better physical storage security.

File Types According to ISO/IEC 7816-4 specification, smart card offers two major file types, dedicated file (DF) and elementary file (EF). DF is also described as directory file, which contains lower-level DFs and EFs. And in EF, real user data is stored. Moreover there is a special DF, called master file (MF), which represents root directory of smart card file system and only selected by smart card OS. Figure 9 illustrates one possible architecture of smart card file system.

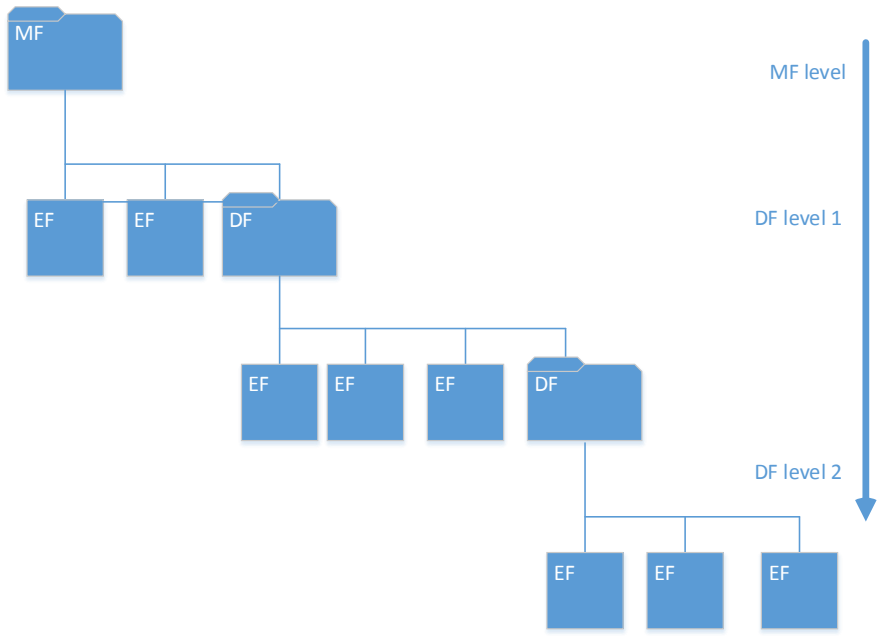

Fig. 9. File Tree on Smart Card[8]

PKCS#15 As already shown, smart card not only is used as storage media but also acts as cryptographic token which offers enhanced security and privacy functionalities for other applications. The PKCS#15 (Public key cryptography

standards) specification[10], which was proposed by RSA Inc. and is nowadays worldwide accepted, provide standards, that define how to store credential information such as cryptographic keys, certificates on smart card, and how to retrieve specified token with help from PKCS#15 interpreter.

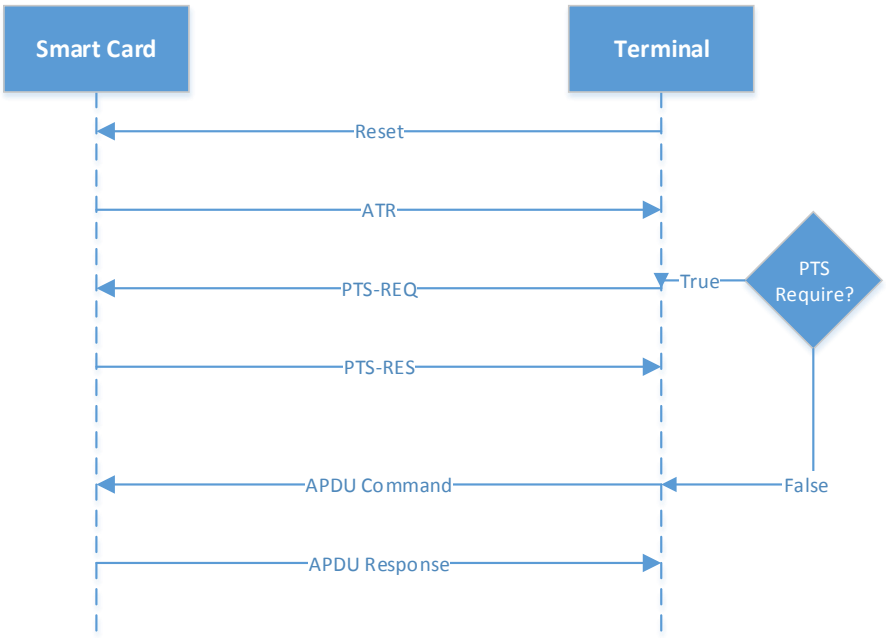

Fig. 10. Communication between Smart card and CAD[8]

Data Exchange with Smart Card The communication protocol between smart card and terminal is described as Master-Slave relationship[8], that means terminal device knowns as Master processes unidirectional control over its slave, namely smart card. Once Master-Slave relationship is established, each communication will be initialized by Master and Slave only reacts based on Master's command.

When a smart card is inserted in a terminal, Master will send its Slave a RESET command which causes smart card to perform a Power-on-Reset behavior. After this Power-on-Reset, smart card informs Master using Answer-to-Rest (ATR) message about card state and communication parameters. In the next step, if necessary terminal will generate a Protocol Type Select (PTS) command, which is used to choose communication protocol and parameters suggested by smart card. After a successful negotiation, smart card and terminal are able to exchange date using Application Protocol Data Unit.

Application Protocol Data Unit, or APDU for short, is used to perform data exchange between smart card and CAD and its structures satisfy ISO 7816-4 specification[11]. There are two categories of APDU, namely command APDU and response APDU. Command APDU structure and response APDU structure are described in Table 3 and Table 4 respectively[8].

Table 3. Command APDU Structure

CLA	class byte identifying application	mandatory
INS	instruction byte representing the actual command	mandatory
P1	parameter 1 used to provide more command information	mandatory
P2	parameter 2 used to provide more command information	mandatory
Lc field	the length of data received by card	mandatory
data field	data sent to card	optional
Le field	the length of data sent by card	

Table 4. Command APDU Structure

data field	length decided by Le of preceding command APDU	mandatory
SW1	state word 1 also called return code 1	mandatory
SW2	state word 2 also called return code 2	mandatory

Secure Messaging Since all communication between smart card and terminal is based on digital electrical pulse performed on card I/O line, attacker can easily record all communication information and and recover it. Therefore secure messaging mechanism is proposed and used to protect against aforementioned message eavesdropping, to ensure authenticity and confidentiality of exchanged information.

In secure message mechanism, both message sender and receiver must agree on to be applied message cryptographic algorithms and corresponding pre-shared keys. In telecommunication domain in accordance with ISO/IEC 7816-4[8], TLV(Type-lengthl-valuse)-formed data, which encapsulates relative user information, is used to perform secure messaging as data carrier.

2.3 Java Card

Java Card technology not only adopts the distinguishing features from Java, such as productivity, security and portability[9], it also makes Java technology available on smart card, where programmer must be faced with more harsh conditions, like limited memory resource and computer ability.

In contrast with Java VM, Java Card VM consists of two parts, the on-card part, which is in charge of bytecode execution, class as well as object management and secure data exchange, the off-card part, which is the real Java Converter. As

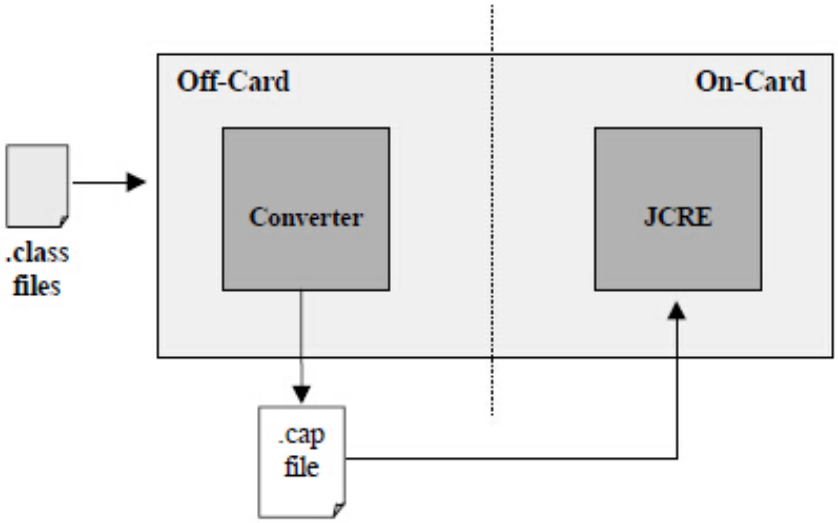

Fig. 11. Java Card VM Components[9]

illustrated in figure 11, a complied Java applet (.cap file) is generated by off-card Converter based on inputed Java class file and executed by on-card Java Card Runtime Environment (JCRE).

Language Specification Apart from above mentioned Java Card VM, compared with original Java language, Java card has many unique features. Since current smart card does not support multitasking, therefore threads are not backed up in Java Card. Also garbage collection is performed by VM, as a result function *finalize()* is not supported. Moreover because on smart card, memory space and process ability is limited, as a result programmer can only use three main primitive types, namely byte, short and boolean. Furthermore only one-dimensional array is offered. Nonetheless Java card language supports all features of inheritance and provides all Java language security features, for example, private access modifiers as well as bytecode verification[9].

Transaction Integrity One of the most import features of Java Card technology is that Java Card Runtime Environment ensures the integrity of transaction, which means even an unexpected loss of power occurs on smart card, the ongoing transactions' integrity is protected with the help of following schema[8]:

```
// Transaction Starts
JCSystem.beginTranscation();

doSomething

//Transaction Ends
JCSystem.commitTransaction();
```

Only when JCRE finishes running method *JCSystem.commitTranscation()*, the corresponding transaction will be finished and submitted. Otherwise JCRE will throw transaction exception and reset data that involved in this broken transaction.

Persistent Object and Transient Object In the realm of Java Card, all objects are preserved in nonvolatile memory, which means this persistent object exits on smart card beyond the execution time of corresponding applet, as long as there exits a reference pointing to it. But also it is allowed to develop transient object on Java card. To be precisely, for instance class array object stays in nonvolatile memory space but in contrast one actual instance of array in stored in volatile memory[8].

Java Card Applet Java Card Applet refers to the Java Card language programmed code, which extends the class *Applet* from package *javacard.framework*. Meanwhile Java Card Applet should implement following four methods:

- *install()*, this method must be implemented and be used to create an applet instance.
- *process(APDU)*, the implementation of this method is also mandatory and uses APDU as input parameter. In this method applet developer designs how to process APDU sent to this applet and which APDU response should be generated.
- *select()*, when JCRE detects a SELECT APDU command, which is applied to select one installed applet, JCRE will call this method of to be selected applet.
- *deselect()*, this method is called by JCRE to inform corresponding applet that it is no longer selected by JCRE.

After finishing programming one Applet, it comes to next phase, namely applet installation. This process takes place usually at the factory or office under the control of card issuer. When one applet is installed on smart card, it only directly communicates with JCRE and other installed applet classes. It should be pointed out, this installed Java Card applet is also belongs to *persistent object* and stored in smart card nonvolatile memory space. Every Java Card Applet is assigned one unique Application ID, which is also known as AID and be used by JCRE to *register* and *select* corresponding applet at run time.

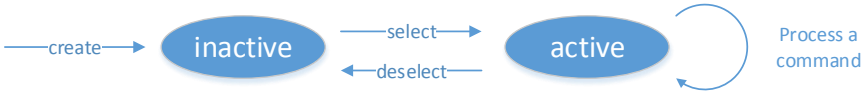

Fig. 12. Javacard Applet Execution States[8]

Figure 12 pictures execution states transaction of Javacard applet. Furthermore figure 13 illustrates how JCRE selects and deselects one applet based on input APDU commands.

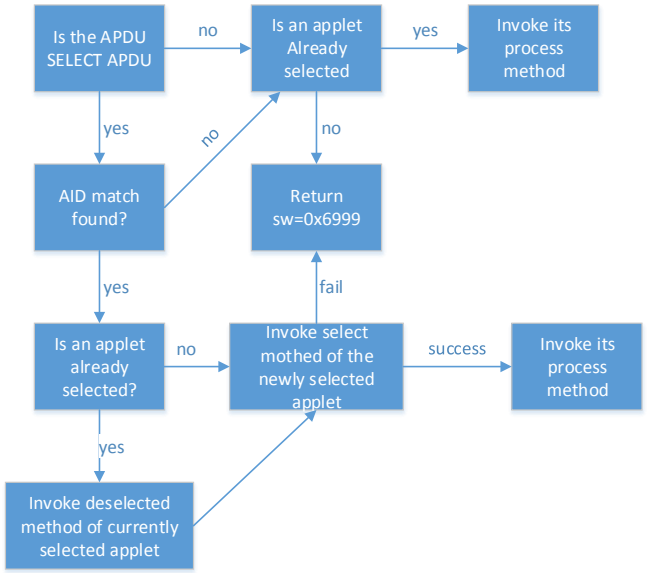

Fig. 13. APDU command processing[8]

Javacard Cryptography Javacard provides a list of APIs to support not only smart card application and data exchange security but also to reinforce other system's security by acting as essential security token. In order to ensure sure messaging between smart card and terminal following three aspects must be considered:

– *Entity Authentication:* Usually mutual authentication is applied to guarantee the authorities of both communication partners.

- *Message Confidentiality:* The transfered information is encrypted using algorithms negotiated between two communication entities to ensure data privacy and security.
- *Message Integrity:* In order to protect exchanged message from unauthorized modification and to provide authenticity assurance, message authentication code (MAC) is calculated based on to be transfered data and integrated in that message.

Packages *javacard.security* and *javacardx.crypto* support aforementioned security mechanism with following class and interfaces[11]:

Table 5. package: *javacardx.crypto*

Class or Interface	Function Description
Cipher	Abstract class offers cryptographic cipher used for encryption and decryption
KeyEncryption	Class provides implementation of keys

Table 6. package: *javacard.security*

Class or Interface	Function Description
Key	Interface for all keys
SecrectKey	Interface for symmetric algorithms' keys
DESKey	Interface for keys used for DES or two-key triple DES or three-key triple DES
PrivateKey	Interface for private keys
PublicKey	Interface for public keys
RSAPrivateKey	Interface for keys used by RSA algorithm to sign data
RSAPublicKey	Interface for keys used to verify signatures generated with RSA
DSAKey	Interface for keys used by DSA
DSAPrivateKey	Interface for keys to sign data with DSA algorithm
DSAPublicKey	Interface for keys to verify signatures generated with DSA
KeyBuilder	Factory class implemented to construct key objects
MessageDigest	Abstract class for hashing algorithm
Signature	Abstract class for signature algorithm
RandomData	Abstract class for generation of random data
CrptoException	Exception class

2.4 GlobalPlatform and Remote Application Management

GlobalPlatform is an international non-profit organization that provides standardized specifications for multiple smart card applications. It is now widely accepted and used as industry standard for managing Java Applet based application on Javacard Operation System in several domains, for instance in communication industries and payment company[12].

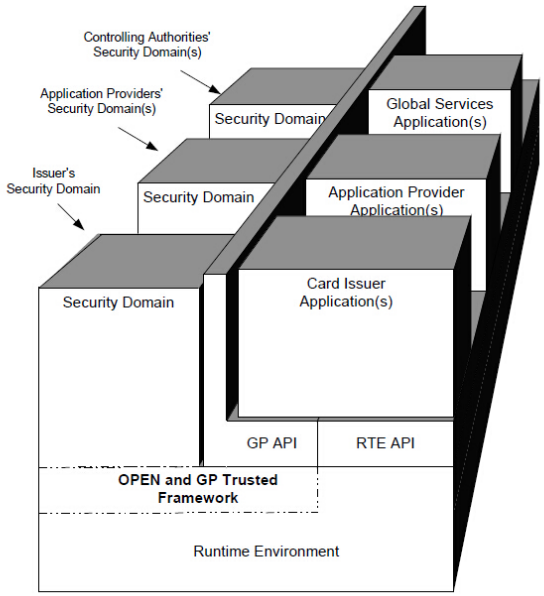

Fig. 14. Card Operation System Architecture[12]

As shown in figure 14, the GlobalPlatform card architecture contains four essential parts. The runtime environment, that provides hardware-neutral API for card application and manages card memory spaces. The on card installed applications, which offers customers various functionalities and services. The security domain (SD), that is usually associated with particular application and known as on-card representatives of off-card authorities. SD is in charge of message encryption as well as decryption, creation and validation of digital signature and handling keys used in cryptographic processes. The last component is OPEN framework[12].

OPEN - GlobalPlatform Environment OPEN provides various sets of APIs offering functionalities such as entity authentication, remote data exchange, secure channel configuration and remote application management. This framework also performs APDU dispatching as well as is application selection and logical channel management[12]. Logical channel is designed to enable the data exchange between multi applications and one terminal. Each opened logical channel will handle message regard of one application. Moreover the special logic channel named basic channel is always opened. In order to ensure system security, OPEN supports secure mechanisms such as, user authentication , resource availability guarantee and secure channel protocol .

Secure Channel Protocol In particular, GlobalPlatform has designed the secure mechanism: secure channel protocol, to guarantee a secure communication. It ensures confidentiality of exchanged information and offers date integrity check. Moreover Secure Channel Protocol also introduces a cryptographic exchange process to let smart card and off-card entities to perform entity authentication with each other.

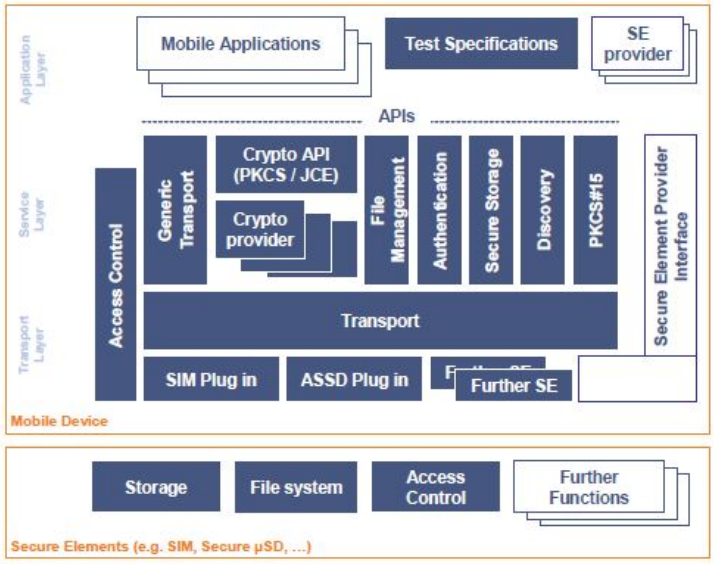

Fig. 15. OpenMobileAPI architecture overview[13]

2.5 OpenMobileAPI

OpenMobileAPI, provided by SIMalliance, constructs an interface between terminal and chip card, which can be used by on terminal installed mobile application to access recourse stored on smart card as well as call function provided by smart card applet. Moreover OpenMobileAPI offers security mechanisms such as access control, that can be applied for mutual authentication between application and secure element. Figure 15 presents an architecture overview of OpenMobileAPI, which consist of three functional layers:

- *Transport Layer:* This layer is in charge of providing secure elements access control services using APDUs and acts as cornerstone for other two layers.
- *Service Layer:* Abstract interfaces, that provide various functions such as secure storage, cryptographic services, are offered by this layer.
- *Application Layer:* Mobile applications which benefit from OpenMobileAPI lie in this layer.

2.6 Android

Overview Android is a open source platform based on Linux and modified by Google, which is designed for mobile devices. As a comprehensive platform, android manages to create a separation between hardware and software that runs on it. At the same time being a open source platform means its entire stack is open and android developer is able to deploy his android on specific hardware as well as learn the system to the fundamental level.[14] Moreover, android system provides a list of software and hardware attracting features to his customers.

- *Security:* Linux, as the cornerstone of android, has been proved to be a secure system through many harsh tests over the years. Which in return guarantee the android system security. [14]
- *Multi-Media:* A wide range of media formats are supported by Android. For instance: MPEG4, ACC, PNG, GIF and so on. [15]
- *Designed for Online:* One outstanding core feature of Android system is the ability to stay Online[16]. Under various conditions such as Wi-Fi network, GSM/CDMA and so on, android device always provide its end users qualified network connection.
- *Variety of applications :*As one of the most popular platform, Android, with the help of Android Market and great number of Android App developers, offers its customers the ability to extend functionalities of their android devices[16]. The user is capable of downloading and installing applications from Android Market, which in return enhances the user experience.

Android Software Stack Android stack is composed of four different layers as shown in figure 16.

Linux kernel: This fundamental layer separates other three layers from device hardware and provide higher layer core functions such as power and hardware driver management.

Libraries: As second layer of android stack, Libraries layer supports android runtime environment by using various C/C++ core libs which offer most of the Java-functionalities, and for android specific design virtual machine, namely Dalvik[14]. Two reasons for not using Java VM are mentioned[14]. First of all, since Java VM is general developed virtual machine, therefore some constrains from mobile systems and devices are not concerned, such as battery life and processing ability. Secondly, when android project was been carried out, Java VM belonged not to open source projects. For these reasons Dan Bornstein and his group developed the license free and mobile platform specific virtual machine, Dalvik.

Apart form android runtime, other libraries which provide services to application framework layers are also included in this Libraries layer, for instance:

- OpenGL, library that supports 2D and 3D graphics rendering.

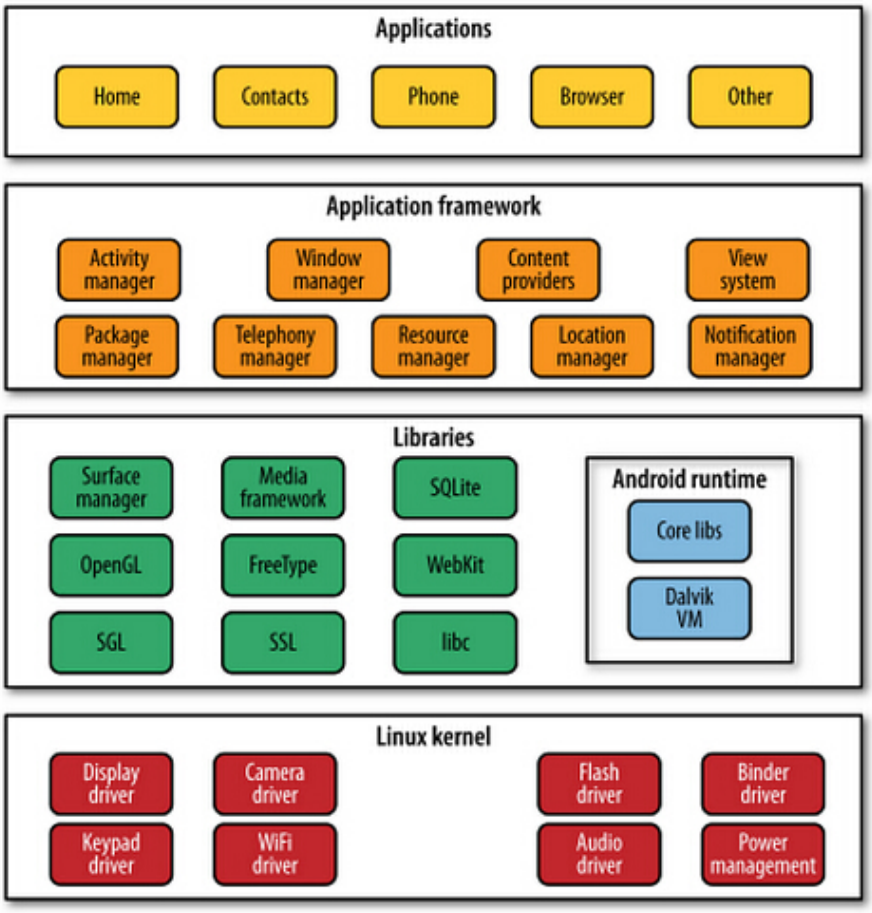

Fig. 16. Android Stack Overview [14]

– SSL, the widely applied secure socket layer library.
– SQLite, which provides SQL database services.
– WebKit, the fast web-rending engine.

Application framework Application framework layer provides a large amount of application framework components, for instance activity manager which is in charge off managing application life cycles, content providers that controls data exchange between applications.

Application Being the top layer of entire android software stack, on this lay both native and third party reside, using component from application framework layer and offering various services to end users.

2.7 Android, Dalvik and Java

As described above, Dalvik VM compared with general Java virtual machine, takes the constrains that are specific about mobile device in to account, which means this android virtual machine concerns the hardware shortcomings including less memory space, low processing power, no swap space as well as short battery lifetime. Minimum recommendations for android device[17] are list in table 7.

Table 7. Minimum Android Recommendations[17]

Feature	Minimum Requirement
Chipset	ARM-based
Memory	128 MB RAM; 256 Flash External
Storage	Mini or Micro SD
Primary Display	QVGA TFT LCD or larger, 16-bit color or better
Navigation keys	5-way navigation with 5 application keys, power, camera and volume controls
Camera	2MP CMOS
USB	Standard mini-B USB interface
Bluetooth	1.2 or 2,0

When Dalvik virtual machine is applied, the compiling process is different from what is taken by Java VM. Figure 17 clearly pictures the difference. In Android runtime environment, after the first compilation of Java source code, the newly generated Java byte code will be compiled by Dalvik Dex complier, as a result, Dalvik byte code is created, which is going to be executed by Dalvik VM.

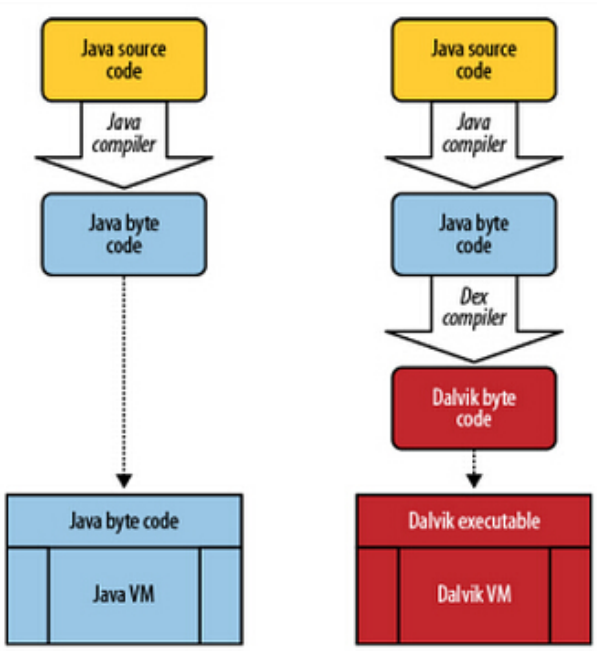

Fig. 17. Compiling process difference[14]

As pictured in figure 18, compared with *.class* Java byte code, *.dex* code adopts shared and type specific constant pools with the main purpose of conserving memory[17]. To be more specifically, the constant pool in Java byte code, which is colored blue in above-mentioned figure, is heterogeneous, which means all constant pools listed in right side *.dex* file are mixed here, as a consequence, duplication may occur.

Another obvious distinguish between Java VM and Delvik is that, the former one adopts stack-based architecture and the later one uses register-based architecture, that in comparison with stack-based architecture needs on average 32.3% less execution time[17], which is obviously the better choice for the system that runs on mobile devices with limited battery life.

3 State of Art

3.1 Smart Home Research

Overview The smart home system we discussed about, that is also described as automated home, integrated home systems or intelligent building[18], has drawn more and more industry developers' and researchers' attention over the decades, research groups such as Siemens, IBM, Cisco, Microsoft[19] has already contributed in this domain. A great number of Smart Home application, network

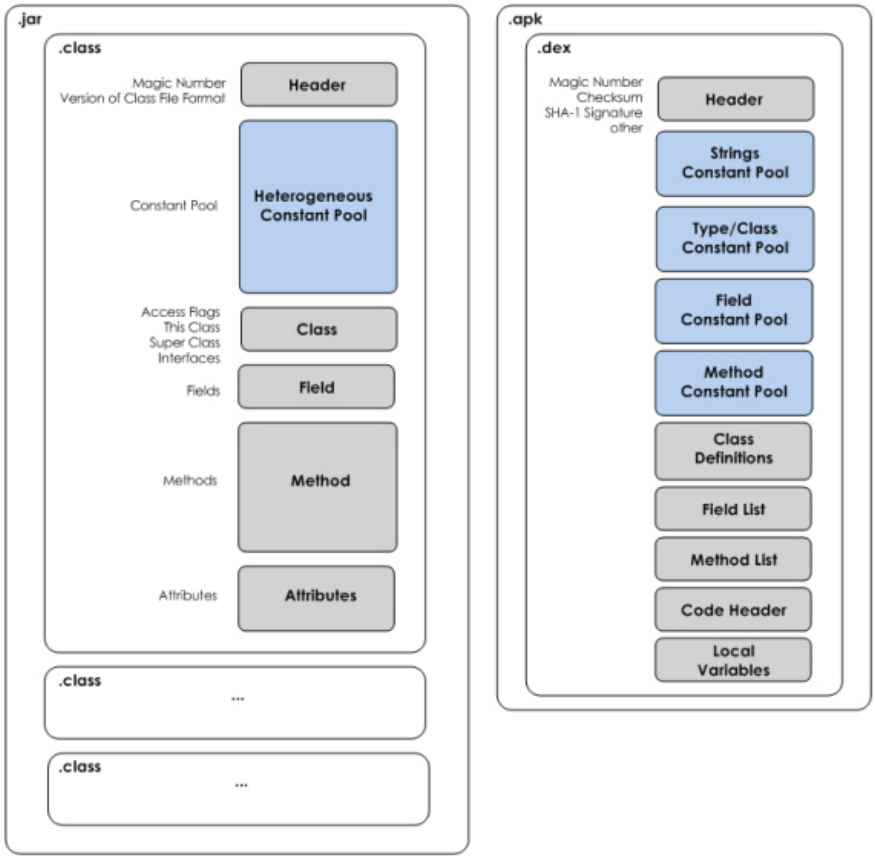

Fig. 18. Comparison between Java byte code and Dalvik executable file[17]

protocols as well as gateways[20] haven come into world and been applied to benefit their customers.

With the development of Smart Home technology, nowadays' Smart Home is not only in charge of monitoring and controlling lighting and heating inside the building, but also capable of connecting almost every electronic devices, inhabitant action prediction as well as making scheduler decisions. Functionalities provided by intelligent home are not just limited to turn device on and off, record and report senor date, but include self-adjusting the inner building environment, supporting various predefined pattern, such as energy saving pattern, especially the concept of Smart Home for elderly[21], which perfectly combines modern remote control and monitoring technologies, with senior-friendly and patient-concerning housing, is welcomed by the market.

Two kinds of Smart Home will be introduced in the following as best practice example, they are Smart Home optimized for energy services and Smart Health Home.

Smart Home Optimized for Energy Services This type of Smart Home put its main goal in the energy saving and monitoring domain, which helps householder to make wiser decisions in under the energy crisis background.

The key component in this Smart Home is decision-support tool[22], that applies a scheduling algorithm which offers house owner suggestion based on various parameters such as distributed energy resources(DER), with the prospect of energy and resource saving.

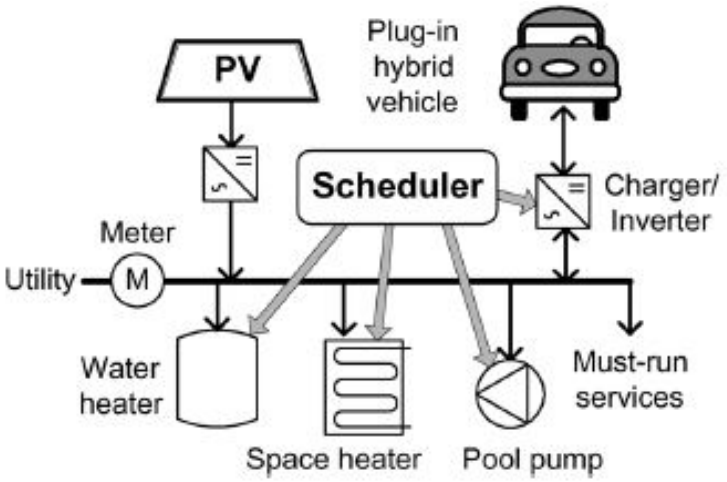

Fig. 19. DER Scheduler in Smart Home optimized for energy[22]

Decision Support Tool The decision support tool used in paper[22] consists of two components, they are energy service model which describes the energy service request and distributed energy resource scheduling algorithm, as described in figure 19 . To be more specifically, energy service model presents the demand for one particular energy resource. For instance the demand aimed at hot water means, the hourly consumption of heated water or the energy that is hourly needed by water heater. According to [22], the heat content of water is also defined as "energy equivalent" and therefore the energy service model is applied to increase "monetary benefit" from every "energy equivalent" unit.

Meanwhile the DER scheduler algorithm helps householder by reducing the unnecessary consumption of energy. This algorithm is in nature one mathematical optimization problem defined by[22].

$$\sum_{t=1}^{T} \sum_{i=1}^{S} [\lambda_{ES,i}(t) \cdot U_{ES,i}(t,x)] - Cost$$

The purpose of DER scheduler, presented as x is to maximize that above introduced fitness function, where S represents all number of services offered by Smart Home, T stands for the whole simulation time, $\lambda_{ES,i}$ and $U_{ES,i}$ describe the "energy equivalent" and energy demand of the ith service respectively. *Cost* means the total electricity consumption. Also in paper[22] the choice of DER algorithm is well discussed.

Smart Health Home Another suitable application domain for Smart Home is the Smart Health Home, which describes the intelligent housing that takes care of patients at home or elder resident.

Overview The charming features of this Smart Home system are the combination telemedical system with communication technologies[**?**]nd customizing services such as, teleconsulting, telediagnosis, real time imaging as well as distance medical education[23], which together improve the living condition of householder and at the same time build a caring system that takes care of residents' need. Nine best practice examples are provided and evaluated in paper[21], they provide a guideline for the design of Smart Home system and summarizes precious experience.

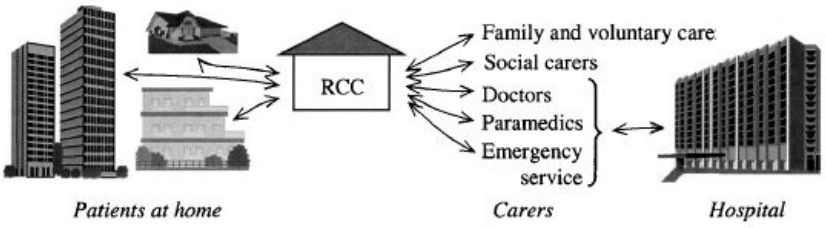

Fig. 20. Overview of Smart Health Home structure[23] (RCC: Remote Control Center)

3.2 OPC UA Security Concept

3.3 Smart Card Security Design

3.4 Secure Android Design

4 Implementation Scenario

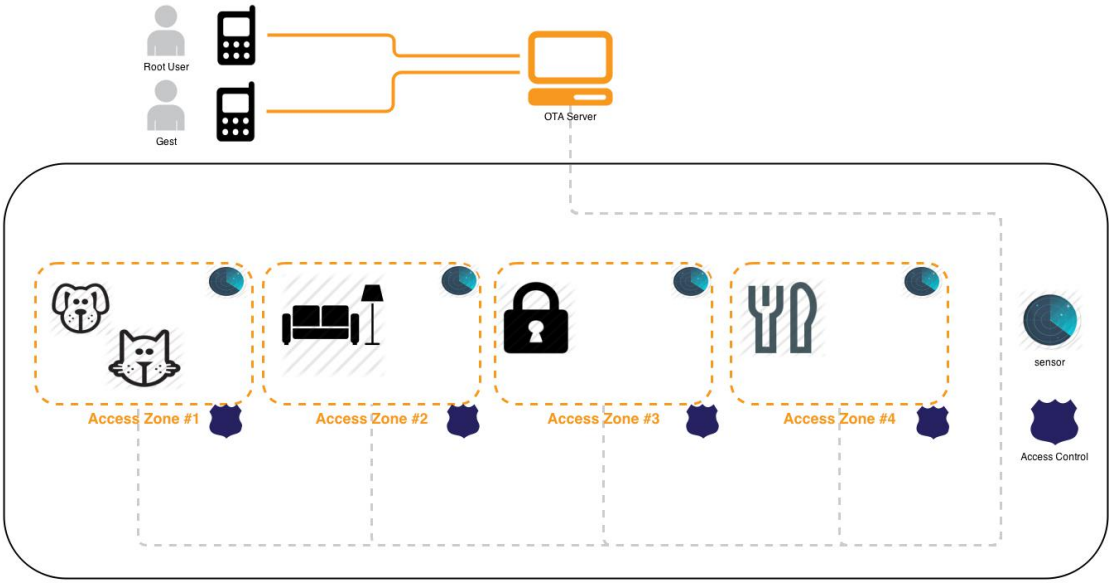

Fig. 21. Smart Home

4.1 Overview

Figure 21 describes the basic structure and functionalities provided by implementation scenario Smart Home. In each room, embedded with UICC smart card sensors which are in charge of monitoring environment variables, such as, home temperature, luminance and how much water the pet has, as well as embedded UICC smart card electronic device, for instance coffee maker, are deployed. On each this sensor and device an OPC UA server is installed, whose major responsibility is controlling that corresponding device as well as data gathered by it. Moreover each door in this scenario is equipped with a digital lock, that only allows people with enough authority to access. Also a OPC UA server exists on those locks and they are embedded with UICC smart card. Users in this scenario can be home owner or guests. Using embedded with UICC smart card cell

phone and on this mobile terminal installed OPC UA client application, subscriber is capable of configuring sensors and querying data gathered by sensors, remote controlling aforementioned secure devices, viewing historical information recorded by corresponding facilities. With the help of such services a comfortable living condition is created in an automated way. Moreover the root user, namely the owner of this house, is also able to assign the permission of accessing particular room to other guests. In case of when he/she is taking a vocation and pet cannot get necessary care.

In this implementation scenario, OPC UA clients, namely Universal Integrated Circuit Card (UICC) based phone user communicates with OPC UA server, which is deployed on other secure hard devices, via an OTA server. Smart card that is applied in this scenario acts as security token for both OPC UA client and server and it contains credential information like encryption keys. certificates and digital signature. The communication stack, that manages secure communication between OPC UA client and server application, is also developed and integrated on smart card as a Java Card applet, which means without corresponding UICC card, OPC UA client and server are not able to appropriately finish their work.

4.2 Software Structure

Figure 22 pictures aforementioned OPC UA client server structure. OP UA Client and Server application communicate with each other with the help of an OTA server. And the communication stack, is in charge of creation and managing the secure communication between OTA server and secure device. Moreover using different chip card, OPC UA client application is able to communicate with OPC UA server application using Short Message Service(SMS) or TCP/IP based web service.
server on secure device provides following services:

- processing client subscription/publishing environment data
- secure message exchange with client
- authority management
- historical data record
- execution client's command

Basic client functions as following are provided:

- submitting subscription/receiving published data
- secure message exchange with server
- sending command/configuration data
- providing user friendly interface

Communication stack is integrated in UICC smart card, whose responsibility is realizing secure channel as well as session management, transporting data to receiver using TCP/IP connections. An internal API translates OPC UA application instructions in to Application Protocol Data Unity (APDU) messagse

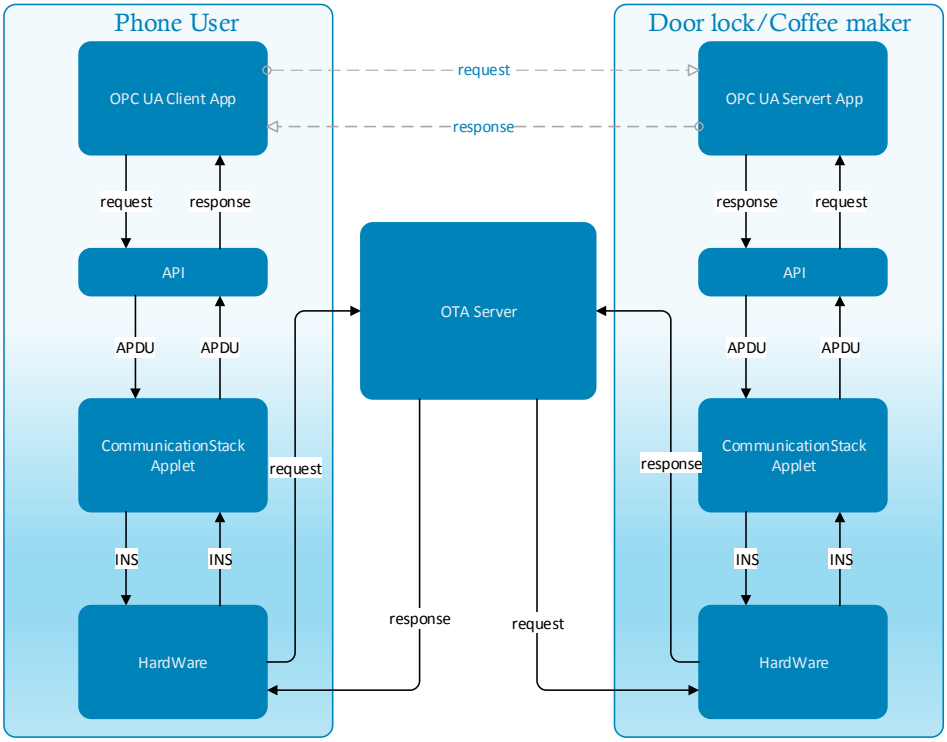

Fig. 22. OPC UA Client Server Structure Example

and forwards them to smart card OS, which is eventually in charge of user authentication and processing secure messaging between card application and chip card pair.

Moreover thanks to self-containment structure, smart card itself does not dependent on other external resources, which could be extreme vulnerable to potential secure attack, and therefore provides a better hardware security and OS security.

Client Structure As described in figure 23, the OPC UA client consists of client application code that realizes client application level functions, OPC UA client API that translates client application instructions into APDU and forwards APDU to UICC smart card as well as manages secure communication between smart card and OPC UA client application code, which is realized as Android App. The Communication stack is developed and integrated with UICC card and its main responsibilities are:

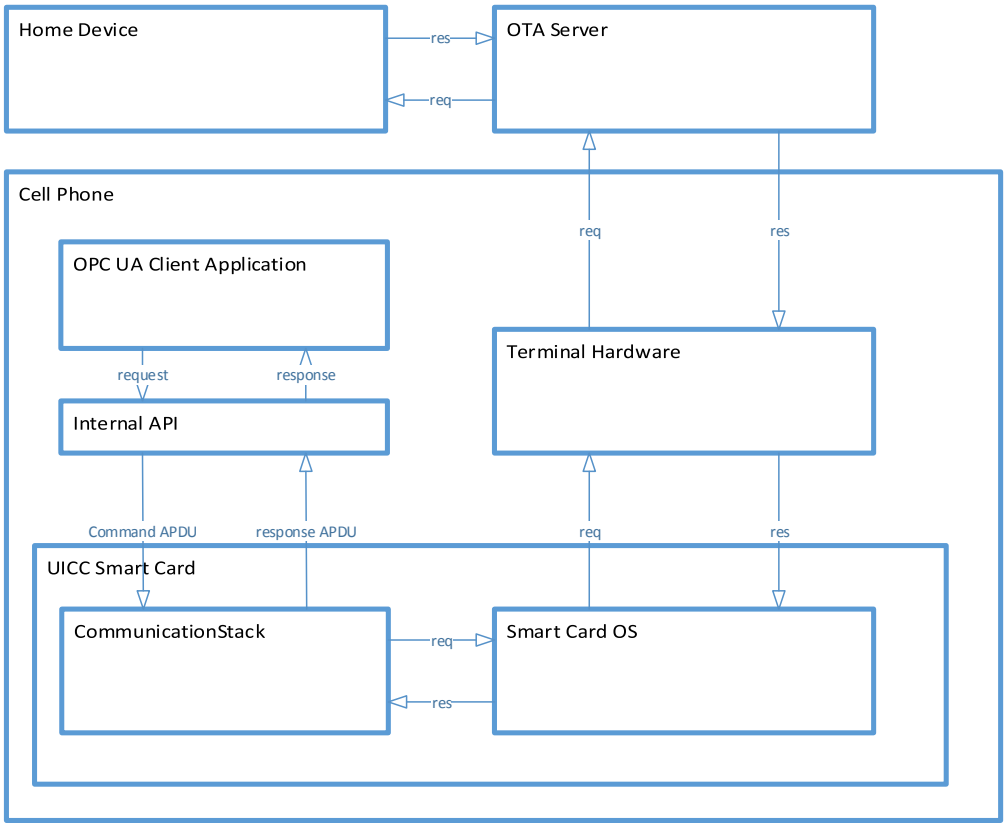

Fig. 23. Client Structure

– initiate HTTP session based on TLS(proactive)
– trigger HTTP session based on trigger SMS send by OTA server(passive)
– rebuild broken communication channel
– message encryption as well as decryption
– message transmit

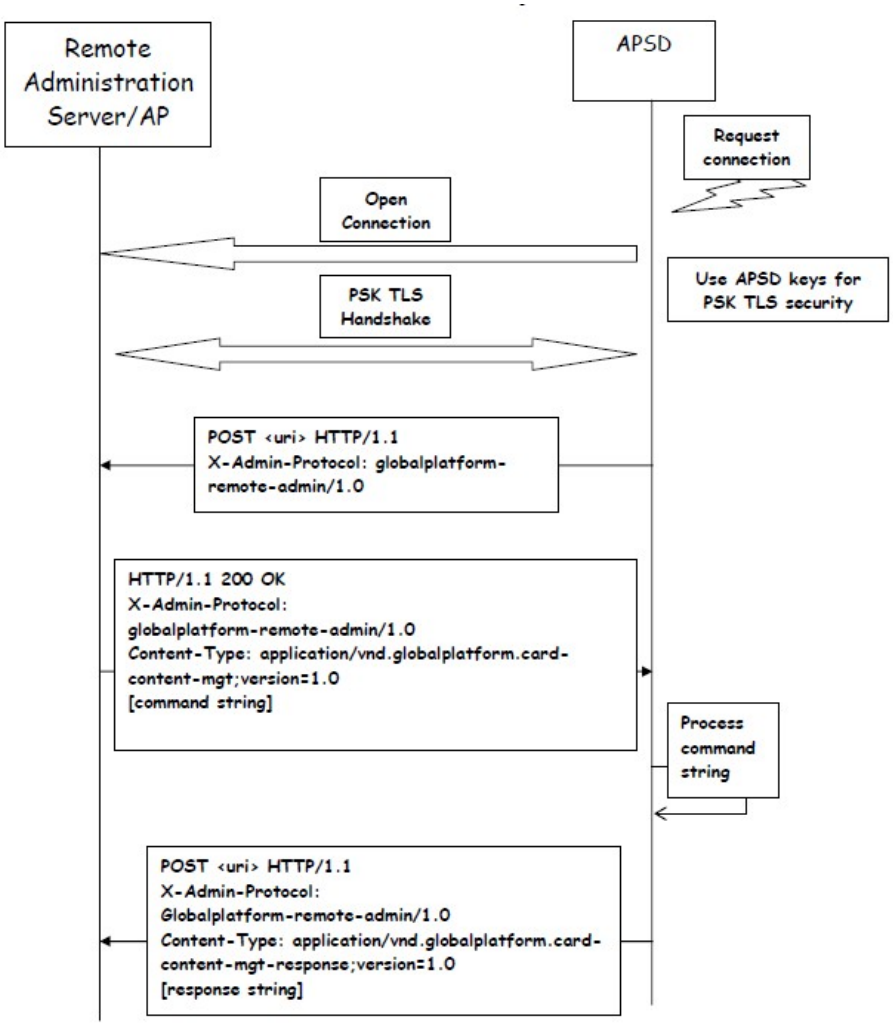

Fig. 24. Communication Flow between an AP and corresponding APSD[24]

The communication flow compliant with the one provided by GlobalPlatform, which provides mechanisms that allow secure information exchanged be-

tween a remote entity and a terminal, this process is also known as Remote Application Management(RAM) over HTTP protocol and PSK TLS security. The on card component, which is responsible for connection creation with the remote entity and user/application authentication, is called Security Domain(SD). And the aforementioned remote entity is referred as Remote Administration Server as well. With these concepts, smart card with Security Domain issued by GloablPlatform can act as HTTP client and is capable of packing APDU formate information into HTTP POST message and transmitting HTTP message to OTA server, which will then forward this HTTP message to target receiver.[24]

Figure 24 illustrates a typical communication flow between administration server and corresponding security domain (Application Security Domain) on smart card. As can be seen, the request for open communication is usually initialized by security domain, which is also the phone user. After a successful creation of secure handshake, the remote administration server and security domain is able to use HTTP connection to exchange request and response strings, which include APDU instructions. GlobalPlatform has also provided a lists of API used to initialize authentication process, to configure algorithm and keys for message encryption and decryption, to perform message exchange behavior and so on.

Server Structure Server here refers sensors, electric device as well digital locks that together build up the smart home system. Each server controls exactly one above-mentioned secure device and take subscription as well as publish corresponding notification to authenticated subscriber. The server structure is pictured as figure 25 and it consists of OPC UA server application code, which offers basic server like subscription and notification mentioned before, an internal API and a on smart card integrated communication stack.

Implementation Tool Support Morpho presents JACADE with full name, Java Card Applet Develop Environment, which is more than just a IDE but a complex selection of various class API and software modules, that can be applied to design complete Java Card Applet as well as to debug Java source. When it comes to running test with Java Card Applet, apart from using regular cell phone with UICC smart card, on where to be tested applet is installed, Morpho Card Reader (MCR) or Java virtual card together with iCardReader, Universal Test Environment (UTE) can be applied. To be more specifically, iCardReader is a tool developed by Morpho and be used in order to send APDU script to Java virtual card or to real smart card connected with test PC using smart card reader such as MCR reader, and to monitor corresponding response APDU information. Universal Test Environment provided by Morpho uses Java languages developed test cases and test scenarios[1] to simulate user cases and obverse related smart card reactions. In contrast with iCardreader, which can only send APDU command sequence to smart card, UTE integrates more software models that can

[1] Test scenario is a collection of relative test cases.

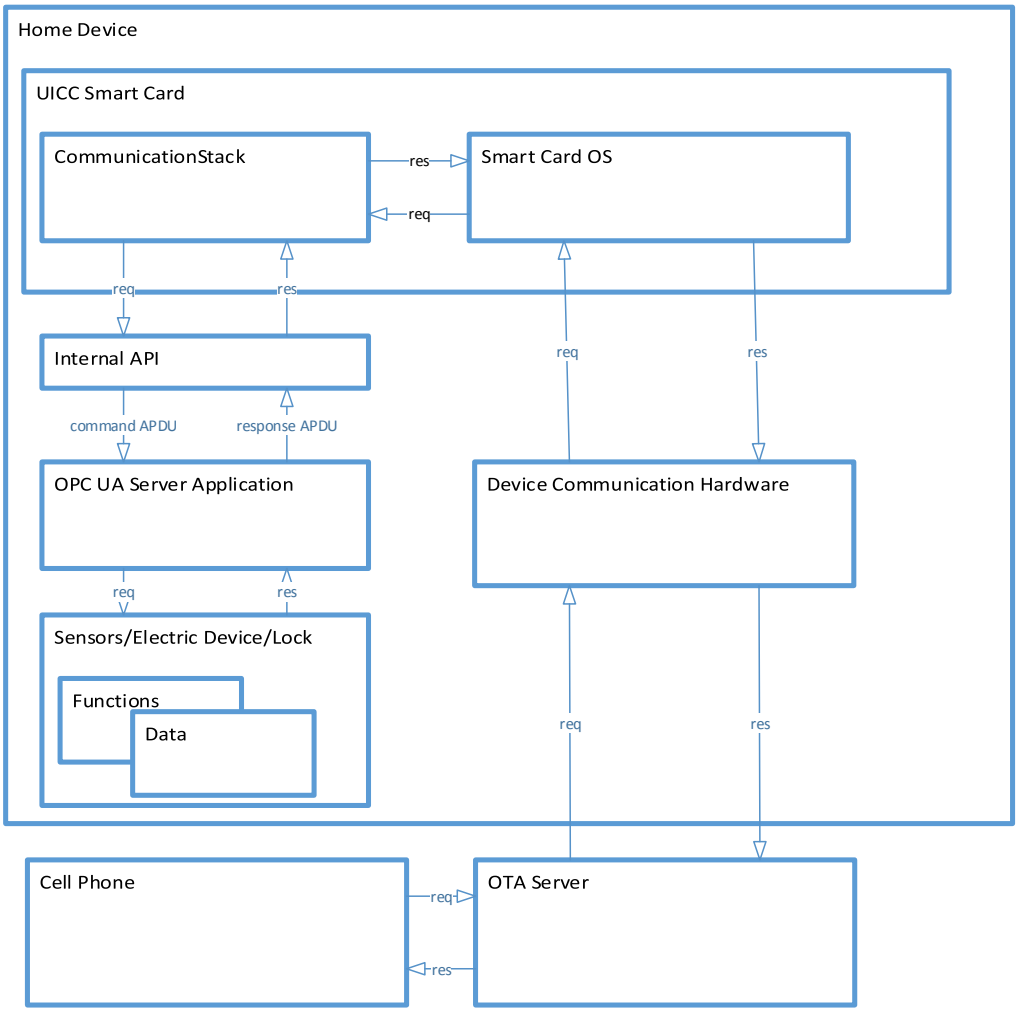

Fig. 25. Server Structure

be used to simulate such as security domain offered by other card application provider, and therefore can provide more sophisticate test environment.

5 System Design

As described in previous sections, my demonstration system consists of two main parts, namely the on card integrated communication stack and android application. Figure 26 illustrates the request and corresponding response message exchange process that occurs in the demonstration system.

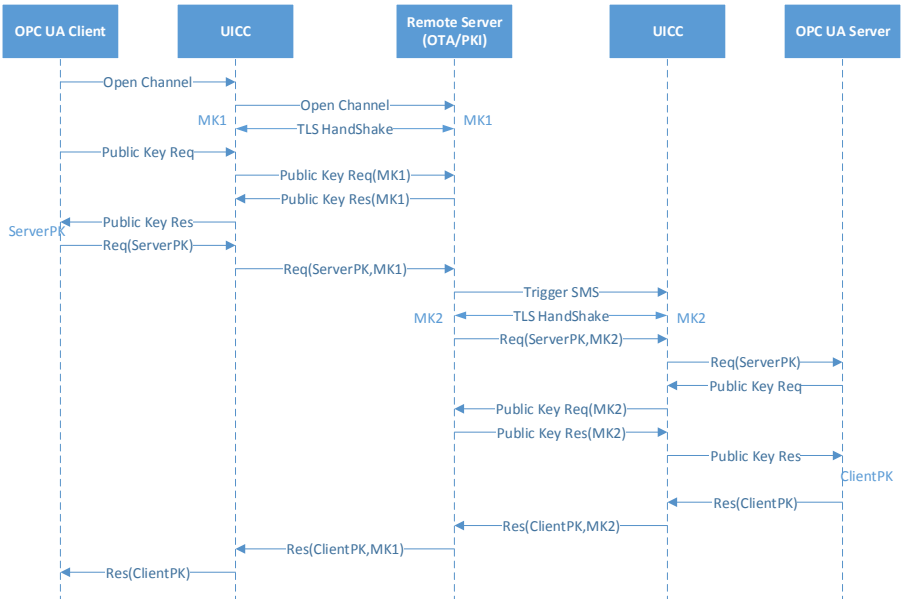

Fig. 26. Message Exchange

5.1 UICC Applet

Classes Communication stack includes following classes:

- *CommunicationStack*, which is the main class of my applet, that implements *install*, *select*, *deselect* as well as *process* methods provided by *javacard.framework.Applet*. This applet is also designed as one UICC system applet.
- *AdminTrigger*, which implements Globalplatform and toolkitframework interfaces and offers functions for proactive and passive communication session creation with OTA server, cipher suit negotiation and mutual authentication.

– *ProcessData*, which is in charge of processing incoming APDUs and transmitting output APDUs. This class also provides methods that are necessary for cryptographic operations.

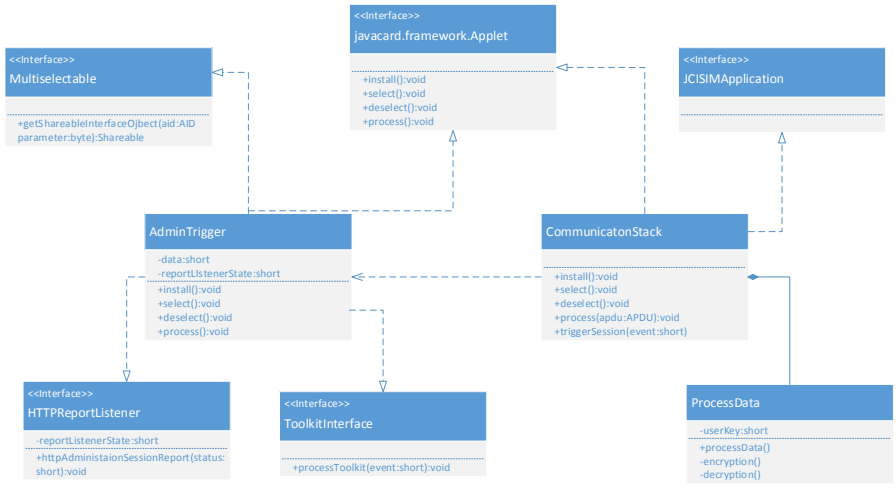

Fig. 27. Class Diagram

Communication Flow The components involved in this scenario are:

– CommunicationStack applet and associated Security Domain from Globalplatform (APSD for short)
– OTA server which is also known as Remote Administration Server (RAS for short)

The communication between CommunicationStack and Remote Administration Server involves following steps:

– Open Communication Channel and Create Communication Session
– Secure Message Exchange
– Close Communication Session

Communication Session Creation and Message Exchange This process could be either initiated by Remote Administration Server by sending target applet a trigger SMS or be sponsored by applet itself. In both cases, applet sends the first *OpenChannel Request* message. During the PSK TLS Handshake phase, APSD and remote server will agree on to be used cipher suit and authenticate each

other. After a successful PSK TLS Handshake, CommunicationStack and remote OTA server will be able to exchange HTTP message,that encapsulates APDU strings as body, with HTTP header that is in compliance with GlobalPlatform standards.

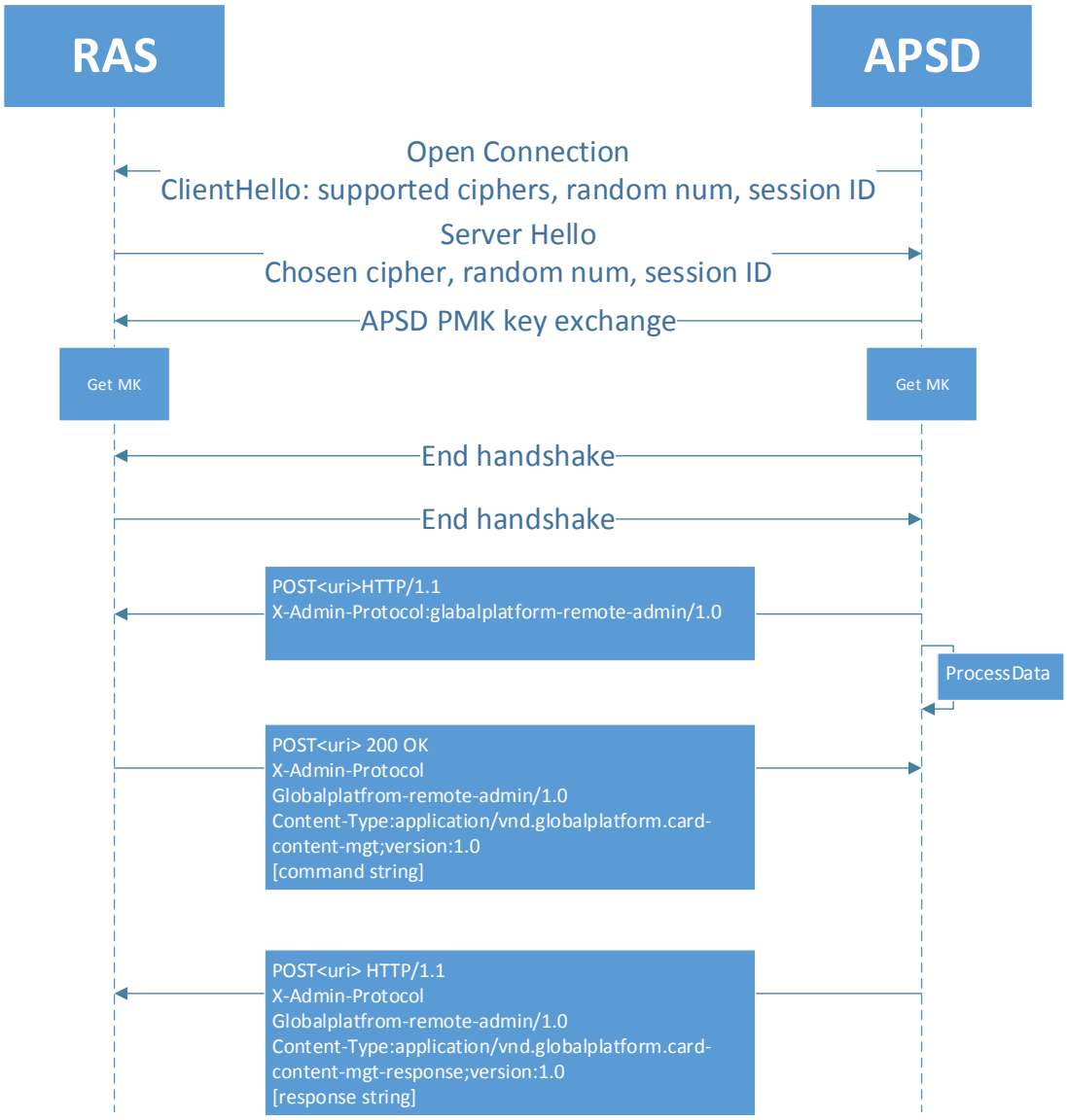

Fig. 28. Class Diagram

HTTP Header Format The HTTP message sent from remote sever to targeted applet uses following schema[12]:

```
HTTP/1.1 200 OK [or HTTP/1.1 204 No Content CRLF]
X-Admin-Protocol: globalplatform-remote-admin/1.0 CRLF
[X-Admin-Next-URI:  <next-URI> CRLF]
[Content-Type: application/vnd.globalplatform.card-content-mgt
-response;version=1.0 CRLF]
[X-Admin-Targeted-Application: <security-domain-AID> CRLF]
[Content-Length: xxxx CRLF] or [Transfer-Encoding: chunked CRLF]
CRLF
[body]
```

In the field *security-domain-AID* could be filled the AID of targeted applet.

The HTTP response message sent from applet to remote server uses following schema[12]:

```
POST<URI>HTTP/1.1 CRLF
Host: <Administration Host> CRLF
X-Admin-Protocol: globalplatform-remote-admin/1.0 CRLF
X-Admin-From: <Agent ID> CRLF
[Content-Type: application/vnd.globalplatform.card-content-mgt
-response;version=1.0 CRLF]
[Content-Length: xxxx CRLF] or [Transfer-Encoding: chunked CRLF]
[X-Admin-Script-Status: <script-status> CRLF]
[X-Admin-Resume: true]
CRLF
[body]
```

The filed *X-Admin-Script-Status* could contain following values:

- *ok*, which means that the previous message is successfully received by applet.
- *unknown-application*, which stands for the error, that the targeted applet for previous message can not be found.
- *not-a-security-domain*, this errors occurs when targeted applet is not a Security Domain.
- *security-error*, as its name indicates, this values is returned if the security of previous message can not be checked.

Close Communication Session Whenever the communication channel is about to be closed, either because of session security issue, or due to successful finish of communication, remote server will send target applet HTTP message:

- No *X-Admin-Next-URI* field is present in this HTTP message and message body is empty, which will be recognized as final message from remote server and then the session will be closed.
- No *X-Admin-Next-URI* filed is present but in this HTTP message body is not empty. The receiver will process the date in body and close the communication session appropriately. But no response message will be generated.

Commands: Interface between Applet and CAD Before concrete implementation of Javacard Applet code, the interface, which in essence is a set of commond APDUs and corresponding response APDUs, between applet and CAD must be well defined. The CommunicationStack support two categories commond APDU:

- The *SELECT* Command APDU, which is used by JCRE to select CommunicationStack applet.
- Other command APDUs, which are introduced in order to provide functionalists such as: trigger communication session, process input APDU and etc. To be more specifically:
  - Verify PIN APDU sets
  - Reset PIN APDU sets
  - Trigger Communication Session APDU sets
  - Close Communication Session APDU sets
  - Exchange Data APDU sets

SELECT APDU The header of this command APDU is fixed and *Lc* indicates the length of CommunicationStack AID. In*Data filed* real AID is saved. Two

Table 8. SELECT command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0x00	0xA4	0x04	0x00	Length of AID	AID	N/A

categories of response APDUs are expected, one represents successful processing of *SELECT* command APDU and the other stands for failure.

Table 9. SELECT response APDU

Optional data	Status word	Description
No data	0x9000	Successful processing
	0x6999	Failed to select CommunicationStack applet

Verify PIN APDU This APDU command and response set is used to let CommunicationStack applet verify identity of terminal user .

Two categories of response APDUs are expected, success response APDU and fail response APDU.

Proactive Communication Session Creation This APDU command and response set is used to let CommunicationStack applet initiate the request to establish communication channel and create communication session above it.

Following return codes are expected in response APDU:

.

Table 10. Verify PIN command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x10	0x00	0x00	N/A	N/A	N/A

Table 11. Verify PIN response APDU

Optional data	Status word	Description
No data	0x9000	Successful processing
	0x6300	Verification failed

Table 12. Trigger Session command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x40	0x00	0x00	0x00	N/A	N/A

Table 13. Trigger Session Return Code

Status word	Description
0x9000	Successful processing
0x66AB	Array Index out of bounds exception
0x665E	Security exception
0x6600	Nullpointer exception
0x6C00	UnKnown exception

Close Communication Session This APDU command and response set is used to correctly close communication channel.

Table 14. Close Session command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x50	0x00	0x00	0x00	N/A	N/A

Following return codes are expected in response APDU:
.

Table 15. Close Session Return Code

Status word	Description
0x9000	Successful processing
0x6031	Failed to close session

Process Communication Data This APDU command and response set is used to perform secure data exchange.

Table 16. Process data command APDU

CLA	INS	P1	P2	Lc	Data field	Le
0xA0	0x50	0x00	0x00	0x00	N/A	N/A

Following return codes are expected in response APDU:
.

Table 17. Process data Return Code

Status word	Description
0x9000	Successful processing
0x6031	Failed to close session

References

1. OPC Foundation: Opc unified architecture specification part1 overview and concepts 1.02. (July 10.2012)

2. OPC Foundation: Opc unified architecture specification part3 address space model 1.01. (February 6.2009)
3. OPC Foundation: Opc unified architecture specification part4 services 1.01. (February 6.2009)
4. OPC Foundation: Opc unified architecture specification part2 security model 1.01. (February 6.2009)
5. SID: Mqtt vs opc-ua: Das http der industrie 4.0? `http://dennisseidel.de/the-http-for-industrie-4-0-mqtt-vs-opc-ua-german` [update;2013].
6. Wikipedia contributors: Constrained Application Protocol. `http://en.wikipedia.org/wiki/Constrained_Application_Protocol` [update;Februray 8,2014].
7. Jean-Louis Carrara,Herv Ganem,Jean-Franois Rubon,Jacques Seif: The role of the uicc in long term evolution all ip networks. (January 2009)
8. Wolfgang Rankl und Wolfgang Effing: Handbuch der chipkarten - 5. deutsche auflage. (2008)
9. Sun Microsystems: Java card applet developers' guide. (July 1998)
10. RSA Laboratories: Pkcs #15 v1.1: Cryptographic token information syntax standard. (December 1999)
11. Zhiqun Chen: Java card technology for smart card. (June 2000)
12. GlobalPlatform: Globalplattform card specification. (January 2011)
13. simalliance: Open mobile api specification. (February 2014)
14. Marko Gargenta: Learning android. (March 2011)
15. Franke Feinbube from HPI University Postdam: Android. (November 2011)
16. Andrew Hoog: Android forensics: Investigation, analysis and mobile security for google android. (July 2011)
17. David Ehringer: The dalvik virtual machine architecture. (March 2010)
18. Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delhoche, Christophe Loge: The smart home concept: our immediate future
19. Li Jiang, Da-you Liu, Bo Yang: Smart home research. (August 2004)
20. Dimitar Valtchev, Ivailo Frankov: Service gateway architecture for a smart home. (April 2002)
21. Sibylle Meyer, Eva Schulze: Smart home fr ltere menschen. (February 2008)
22. Michael Angelo A. Pedrasa, Ted D. Spooner ,Iain F. MacGill: Coordinated scheduling of residential distributed energy resource to optimize smart home energy services. (September 2010)
23. Vincent Rialle, Florence Duchene, Norbert Noury, Lionel Bajolle, Jacques Demongeot: Health ´smart´ home: Information technology for patients at home. (Number 2002)
24. GlobalPlatform Card: Remote application management over http card specification v2.2 amendment b. (March 2012)