Aplicación Web para la administración de tarjetas inteligentes con GlobalPlatform.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas.

Autor(es): Rita Milena Hernández Díaz.
Yasiel Conde Bernal.

Tutor(es): MSc. Adonis Cesar Legón Campos.
Ing. Ander Sánchez Jardines.

Universidad de las Ciencias Informáticas
La Habana Junio de 2013

“Todos y cada uno de nosotros paga puntualmente su cuota de sacrificio consciente de recibir el premio en la satisfacción del deber cumplido, conscientes de avanzar con todos hacia el Hombre Nuevo que se vislumbra en el horizonte.”

Ernesto Guevara.

Declaración de autoría

Declaramos que somos los únicos autores del presente trabajo titulado: Aplicación *Web* para la administración de tarjetas inteligentes con *GlobalPlatform* y autorizamos a la Universidad de las Ciencias Informáticas y al Centro de Identificación y Seguridad Digital a usarlo en su beneficio.

Para que así conste firmamos la presente a los ___ días del mes de___ del año___

Autores

_____	_____
Rita Milena Hernández Díaz	Yasiel Conde Bernal

Tutores

_____	_____
Adonis César Legón Campos	Ander Sánchez Jardines

Agradecimientos

Rita Milena

A mis abuelitos Magali y Candelario, por cuidar de mí desde que era pequeña, por consentirme y mimarme en todo momento. Por ser mis guías y mis dos grandes amores.
A mi tata Ilena por todo su cariño y enseñanzas, por sus regaños y buenos consejos, por tener esa inmensa fe en mí.
A mis 3 hermanitos Tania Milena, Lázaro Israel y Wimian, a pesar de que no se los diga muy seguido los quiero muchísimo.
A mi papá por su incondicionalidad.
A mi primo Pabel por estar pendiente de mí durante mis años de la universidad.
A toda mi familia por ser tan especial y unida.
A Maidi y Arlety por estar a mi lado en los momentos más difíciles de mi vida y enseñarme el verdadero valor de la amistad.
A Nelson por acompañarme a lo largo de estos cuatro años, por ser mi amigo, mi cómplice e inspirarme a ser una mejor persona.
A Susej por sus enseñanzas y paciencia.
A mis tutores Adonis y Ander por guiarme durante la realización de este trabajo con paciencia y maestría.
A todos los compañeros del departamento de Tarjetas Inteligentes.
A los educadores que han contribuido con mi formación como universitaria.
A todos los que han estado a mi lado de una forma u otra en algún momento de mi vida, legándome algún consejo o algún gesto de apoyo.

Yasiel

Quisiera darle las gracias primeramente a mi Señor Jesucristo y al Espíritu Santo por estar siempre a mi lado guiándome y por levantar mi ánimo en los momentos difíciles.

A mi madre, por estar siempre a mi lado apoyándome, por ser mi inspiración, por regañarme y aconsejarme siempre.

A mi abuela por ser tan especial conmigo y darme todos mis gustos.

A mi hermana y mis sobrinas por ser tan especiales para mi vida.

A mi padre y mi madrastra por ayudarme siempre que lo necesité.

A mi amiga Grettel, por su amistad incondicional durante estos 5 años de carrera.

A mi amiga Heidi por estar presente en mis momentos difíciles.

A mi hermanito Yordany por ayudarme a acercarme cada día más a la luz que es Jesucristo.

A Yasmany por sus enseñanzas tan poderosas y por ser tan comprensivo y paciente conmigo.

A Asiel por apoyarme siempre, por introducirme acá en el pueblo cristiano de la UCI y por sus tiempos de oración conmigo que me fortalecieron grandemente y que además fueron de gran bendición.

A mi hermanito Roberto Cárdenas por dedicarme tiempo de su tiempo para compartir la palabra de Dios.

A la comunidad cristiana de la UCI por su unidad y amor tan grandes.

A Hugo Chávez Frías por su entrañable amistad y solidaridad con la Revolución Cubana.

A nuestro querido Comandante en Jefe Fidel castro Ruz por hacer nuestros sueños realidad.

A nuestro presidente Raúl Castro Ruz por continuar reafirmando su apoyo a nuestra universidad.

A todas aquellas personas que de cierta manera contribuyen a la predicación o distribución de la palabra de Dios.

Dedicatoria

Rita Milena

Ante todo a mi abuela Magali por ser mi razón de ser, la persona que me impulsa cada día a querer ser mejor, por su amor incondicional y ternura desmedida.

A mi tía Ilena por apoyarme en todas mis decisiones y ser mi eterna confidente.

A mí mamá, por ser mi ángel de la guarda. A pesar de no estar físicamente llevo conmigo sus palabras y su cálida sonrisa.

A mi familia en general por ser la mejor del universo.

Yasiel

A mi madre, por estar siempre a mi lado apoyándome y por ser mi inspiración.

A mi abuela por ser tan especial conmigo y darme todos mis gustos.

A mi hermana y mis sobrinas por alegrar mi vida.

A mi padre y mi madrastra por ayudarme siempre que lo necesité.

Resumen

En el presente trabajo se hace un profundo estudio acerca de las tarjetas inteligentes y las tecnologías asociadas, ya que actualmente la mayoría de las aplicaciones que ofrecen servicios a través de la *Web* haciendo uso de tarjetas inteligentes, necesitan de la instalación de *middlewares* del lado del cliente. Esto supone un elevado riesgo para su seguridad, debido a que el uso de claves simétricas en el cliente posibilita que puedan ser accedidas por atacantes permitiendo acceso a operaciones que requieren seguridad sobre la tarjeta y los sistemas que la utilizan.

Con el objetivo de eliminar los riesgos antes mencionados, en el Centro de Identificación y Seguridad Digital (CISED) de la Universidad de las Ciencias Informáticas (UCI), se desarrolla una aplicación *web* para la administración de tarjetas inteligentes con *GlobalPlatform* 2.1.1 que cuenta con un *middleware* del lado del servidor, aprovechando las ventajas que ofrece este estándar en cuanto a seguridad.

El documento recoge los resultados de la investigación realizada. Se describen de forma detallada las principales características de los sistemas analizados, la arquitectura, el diseño de sistema propuesto; así como las herramientas, tecnologías utilizadas y los artefactos generados en el proceso de desarrollo de la aplicación.

Palabras claves: administración, *GlobalPlatform*, *Middleware*, Tarjetas inteligentes.

Índice de contenido

Índice de figuras

Índice de tablas

Índice de tablas

Introducción

En la actualidad el mundo se encuentra inmerso en un complejo proceso de construcción, generación y perfeccionamiento de la sociedad tecnológica, al querer transformar y sustituir los viejos sistemas de orden político y social; por completas y modernizadas instituciones públicas que basan el flujo de sus procesos en los más novedosos adelantos tecnológicos. Con el avance de las Tecnologías de la Información y las Comunicaciones se registra una tendencia al uso de dispositivos inteligentes que facilitan y aseguran el proceso de identificación, tal es el caso de las tarjetas inteligentes. (1)

Son muchos los escenarios en los que se utilizan estos dispositivos, entre los que se puede mencionar el comercio electrónico con el uso de los monederos electrónicos. También son utilizadas en el control de acceso, este tipo de aplicación suele estar ligada a puertas automatizadas que permiten o impiden el paso físico de una persona a un área determinada. Otro de sus importantes usos en la actualidad, es su vinculación a la firma digital de documentos, o sea, el almacenamiento del certificado digital dentro de la tarjeta. Las tarjetas inteligentes también se han expandido a la esfera de la salud, ya que el uso de la tarjeta anula la necesidad de tener que compartir una inmensa base de datos y tener que hacer réplicas periódicas. Otro de sus usos es en el sector público para el transporte, ya que permite pagar la cuota de autobús sin necesidad de usar efectivo o monedas. En los últimos años las tarjetas inteligentes han ido evolucionando en la tecnología móvil, ofreciendo a los clientes servicios con un mayor beneficio y facilitándoles diferentes aplicaciones y tecnologías desde el móvil. (1)

Países como España, Finlandia y Venezuela, son ejemplo de las tendencias e incremento del uso de estas tecnologías, para erradicar los problemas de seguridad y suplantación de identidad. (1) Formando parte de los procesos enfocados al progreso de la informática y las comunicaciones en Cuba, la Universidad de las Ciencias Informáticas (UCI) en el Centro de Identificación y Seguridad Digital (CISED), específicamente en el departamento de Tarjetas Inteligentes ha venido desarrollando soluciones referentes a esta tecnología, con el fin de obtener productos y servicios basados en tarjetas inteligentes, entre los que se encuentran el *SmartCardFramework* que permite el desarrollo para aplicaciones de pasaportes electrónicos, licencia de conducción e Infraestructura de Clave Pública (PKI por sus siglas en Inglés), la aplicación *SmartCardTool* que forma parte del *SmartCardFramework*, el *SmartCoP* que es una plataforma para el desarrollo de servicios en línea, entre otras.

El desarrollo de sistemas operativos multiaplicaciones como *Java Card*[1], posibilita la implementación de aplicaciones *applets* que se ejecutan en la tarjeta, de modo que esta permita su uso práctico en un dominio de aplicación específico. Para la gestión segura de las aplicaciones en el chip de la tarjeta (descarga, instalación, personalización, supresión) muchos de los productos *Java Card* se basan en las especificaciones *GlobalPlatform* [2] que permite administrar sus aplicaciones. (2)

La manera tradicional hasta la actualidad de administrar tarjetas inteligentes es a través de *middlewares* que usan clave simétrica, lo que pone en riesgo su seguridad. Para establecer canales seguros de comunicación *GlobalPlatform* establece la forma de realizar la comunicación (protocolo) y las posibles operaciones a realizar con la tarjeta (comandos). (2)

Partiendo de todo lo anteriormente abordado se manifiesta la siguiente **situación problemática**:

Para realizar las operaciones de administración en las tarjetas inteligentes que implementan *Java Card* y *GlobalPlatform* es necesario instalar un *middleware* del lado del cliente. Por tanto se hace complejo ejecutar las operaciones de administración, debido a la necesidad de dominar su uso por parte del cliente. Unido a esto para realizar operaciones sobre los dominios de seguridad es necesario que el *middleware* cuente con las llaves para tener acceso a los mismos. Por cada terminal donde se llevan a cabo las operaciones es necesario contar con las llaves, constituyendo una vulnerabilidad, pues estas al ser simétricas corren el riesgo de ser obtenidas por personal no autorizado, permitiendo el acceso a operaciones que requieren seguridad sobre la tarjeta.

Derivado de la situación anteriormente expuesta se formula el siguiente **problema científico**:

¿Cómo mejorar la administración de tarjetas inteligentes que implementen el estándar *GlobalPlatform* 2.1.1, en cuanto a seguridad?

Por tanto, el presente problema está contenido en el **objeto de estudio**: administración de tarjetas inteligentes con *GlobalPlatform*.

Para dar solución al problema existente se ha tomado como **objetivo general**: desarrollar una aplicación *web* para la administración de tarjetas inteligentes mediante la implementación del estándar *GlobalPlatform* 2.1.1, utilizando la plataforma para el desarrollo de servicios en línea con tarjetas

[1] *Java Card*: es una tecnología que permite ejecutar de forma segura pequeñas aplicaciones *Java* (*applets*) en tarjetas inteligentes y similares dispositivos empotrados.

[2] *GlobalPlatform*: organización internacional que desarrolla y publica las especificaciones que facilitan el despliegue de múltiples aplicaciones integradas en la tecnología de chip seguro.

inteligentes, desarrollada en el departamento Tarjetas Inteligentes del CISED.

Entre los objetivos específicos se encuentran:

- Conformar el marco teórico de la investigación.
- Realizar el análisis y diseño de la solución para la administración de tarjetas inteligentes con *GlobalPlatform* 2.1.1.
- Implementar la aplicación *web* utilizando la plataforma para el desarrollo de servicios en línea (*SmartCoP*) con tarjetas inteligentes.
- Probar la solución *web* desarrollada.

Para dar respuesta a la interrogante presentada en este trabajo y con los objetivos trazados se plantea el cumplimiento de las siguientes tareas de la investigación.

- Análisis de la situación actual de la administración de tarjetas inteligentes en aplicaciones *web* para la sustentación teórica del problema.
- Análisis de los diferentes estándares, herramientas y tecnologías a utilizar para el desarrollo de la solución propuesta.
- Análisis de la aplicación *SmartCardTool* y del *SmartCardFramework* para determinar las características esenciales de su funcionamiento.
- Análisis y diseño de las extensiones (*PlugIn*) que actúan como eje del *middleware* del lado del servidor *JWebSocket.*
- Realización del diseño de interfaces correspondientes a las extensiones generadas en el servidor.
- Implementación de las extensiones del lado del servidor *JWebSocket*.
- Implementación de las interfaces correspondientes a esas extensiones del lado del cliente.
- Realización de las pruebas unitarias y de aceptación a la aplicación *web* para verificar el cumplimiento de los requisitos definidos.

Los métodos científicos de investigación se clasifican en **métodos teóricos** y **métodos empíricos**.

Los **métodos teóricos** permiten estudiar particularidades del objeto de investigación, además de proporcionar la construcción de modelos. Por estas razones los métodos teóricos que se evidencian en la presente investigación son: Analítico-Sintético e Histórico-Lógico.

- **Analítico–Sintético**: mediante este método se realizará un estudio de diferentes soluciones existentes en la actualidad y los principales estándares relacionados con las tarjetas inteligentes.

Además, se consultará toda la bibliografía necesaria para dar cumplimiento a las tareas de la investigación.

- **Histórico-Lógico**: permitirá analizar la trayectoria completa acerca del proceso de administración de tarjetas inteligentes en la *web*, así como el estudio histórico del mismo que determina deficiencias y propone soluciones de acuerdo a las necesidades.

Los **métodos empíricos** permiten, como parte del procedimiento trazado para la investigación, determinar el método de recolección de datos y tipo de instrumento que será utilizado, por tanto el método empírico que se utiliza en el presente trabajo de diploma es: Entrevista.

- **Entrevista:** posibilita obtener información, experiencias, ideas, puntos de vistas, que contribuyan al desarrollo de la investigación y aporten conocimientos específicos del tema.

El fundamento de la presente investigación está dado por la importancia que posee el desarrollo de un *middleware* del lado del servidor que permita administrar tarjetas inteligentes, debido a que se elimina el riesgo de que las llaves simétricas sean obtenidas por atacantes, permitiendo una mayor seguridad para la administración. Además al ser estas tarjetas implementadas con el estándar *GlobalPlatform* son aún más fiables debido al canal de comunicación seguro que este estándar brinda, siendo de gran utilidad para el departamento de Tarjetas Inteligentes del CISED.

El trabajo se encuentra estructurado de la siguiente forma:

Capítulo 1. Fundamentación teórica: se realiza una fundamentación teórica de la investigación. Se expone un análisis del estado del arte del proceso de administración de tarjetas inteligentes para la *Web* actualmente, tanto a nivel nacional como internacional, así como las herramientas, tecnologías y metodología a utilizar en el desarrollo de la aplicación.

Capítulo 2. Características, análisis y diseño del sistema: brinda una descripción de la solución propuesta, a partir de la cual se describen las actividades de análisis de la solución, seguidas por la descripción de los procesos del sistema.

Capítulo 3. Implementación y pruebas: se describe la etapa de implementación. Se elaboran y documentan las pruebas realizadas a la solución propuesta para demostrar el correcto funcionamiento de la solución de la misma. Se realiza un análisis de los resultados de la aplicación en un entorno real.

Capítulo 1: Fundamentación teórica

1.1. Introducción

Este capítulo tiene como objetivo, abordar todo lo referente al estado del arte que presenta el tema de administración de tarjetas inteligentes en la *Web*, tratando los principales conceptos y aspectos más relevantes relacionados con las herramientas abordadas en diferentes fuentes bibliográficas. Se realiza un estudio de las metodologías de desarrollo de *software*, los lenguajes de programación, las herramientas y tecnologías que se utilizarán para el desarrollo de la solución. Además de ofrecer un panorama general acerca de la gestión de tarjetas inteligentes para aplicaciones *web* desarrolladas con *JWebSocket* que utilicen el estándar *GlobalPlatform* 2.1.1.

1.2. Tarjetas Inteligentes

Las tarjetas inteligentes fueron concebidas y patentadas por alemanes, japoneses y franceses en 1970. Estos dispositivos son tarjetas de plástico que contienen un circuito integrado, son semejantes en tamaño y otros estándares físicos a las tarjetas de crédito. Dicho circuito puede ser de solo memoria o contener un microprocesador con un sistema operativo que le permite un grupo de tareas tales como:

1. Almacenar información
2. Encriptar información
3. Leer y escribir datos, como una computadora

Las tarjetas inteligentes con su seguridad hacen que los datos que se consideran personales solo sean accesibles por los usuarios apropiados. Aseguran la portabilidad, seguridad y confiabilidad en los datos. Son las tarjetas inteligentes basadas en estándares las utilizadas para autenticar la identidad de una persona o entidad. Es importante destacar el nivel de seguridad que poseen para el almacenamiento de la información. Pueden ser multiaplicaciones, ya que pueden contener información de una o varias aplicaciones al mismo tiempo, esto puede ocurrir debido a que poseen una jerarquía de almacenamiento de la información. Cada una de las aplicaciones contenidas en la tarjeta se protege independientemente. (1)

Con el surgimiento de las tarjetas inteligentes se han desarrollado cuantiosas aplicaciones, que hasta hace unos años eran consideradas prácticamente imposibles. En la actualidad se pueden apreciar en diferentes ámbitos, siendo los más comunes: (3)

- Telefonía Móvil Digital: donde la identificación del titular del número telefónico, la herramienta de cifrado y la base de datos del propietario constituyen las funciones de la tarjeta inteligente.
- Banca: su desarrollo está dado a través de los monederos electrónicos, permitiéndole al usuario cargarlo con una determinada cantidad de dinero y realizar compras teniendo en cuenta el saldo disponible en la tarjeta.

Las tarjetas inteligentes son clasificadas de acuerdo a sus capacidades, estructura del sistema operativo, formato e interfaz de comunicación. Esta última está compuesta por otras dos clasificaciones: tarjetas de contacto y tarjetas sin contacto. Las tarjetas de contacto disponen de unos contactos metálicos visibles y debidamente estandarizados (parte 2 de la Organización Internacional de Normalización (ISO por sus siglas en Inglés) y la Comisión Electrónica Internacional (IEC por sus siglas en Inglés) 7816), no siendo así en las tarjetas sin contacto. Las tarjetas de contacto, por su parte, deben ser insertadas en una ranura de un lector para llevar a cabo operaciones en las mismas. Luego el lector alimenta eléctricamente a la tarjeta y transmite los datos oportunos para operar con ella conforme al estándar. Por su parte las tarjetas sin contacto utilizan la tecnología de identificación por radiofrecuencia (RFID por sus siglas en Inglés) para leer y escribir desde el chip incrustado en la tarjeta. (1)

1.3. Comunicación de la tarjeta

Cuando dos computadoras se comunican mutuamente, ellas intercambian paquetes de datos, los cuales son construidos siguiendo un conjunto de protocolos. En forma similar, las tarjetas inteligentes se comunican al mundo exterior usando sus propios paquetes de datos llamados APDU (*Application Protocol Data Units*)[3]. Los APDU pueden contener un mensaje de comando o un mensaje de respuesta. Por lo cual una tarjeta inteligente siempre juega el rol pasivo. En otras palabras, una tarjeta inteligente siempre espera por un comando APDU desde una terminal, ejecutando la acción especificada en el APDU y respondiendo a la terminal con un APDU respuesta. APDUs comandos y APDUs respuestas son intercambiados alternativamente entre una tarjeta y una terminal. (3) (Ver anexo 2)

[3] El *Application Protocol Data Unit (*APDU*)* es la unidad de comunicación entre un lector de tarjetas inteligentes y una tarjeta inteligente.

1.4. Organización para el intercambio de información

Para organizar el intercambio de información entre la tarjeta inteligente y la interfaz de comunicación, por ejemplo, el lector de tarjetas inteligentes, se especifican los siguientes rasgos básicos: (4)

Parejas de comando-respuesta

Los comandos y respuestas se envían o se reciben de la tarjeta inteligente en forma de APDU, que es la unidad de comunicación entre un lector de tarjetas inteligentes y la propia tarjeta. Su estructura es la siguiente:

- **Cabecera:**

1 byte para denotar la clase (CLA).

1 byte para denotar la instrucción (INS).

2 bytes para denotar los parámetros (P1-P2).

- **Campo Lc:** se compone 0, 1 o 3 bytes en el caso de los APDUs extendidos. En caso contrario estará ausente.
- **Campo de datos:** bytes correspondientes a la longitud de la cadena de datos a enviar. Si no hay datos a enviar, no se especificará ningún valor en este campo.
- **Campo Le:** número de bytes esperados en la respuesta. Si no se espera ninguna cadena de bytes en la respuesta, no se especificará ningún valor en este campo.

1.5. Estándares utilizados en tarjetas inteligentes

ISO 7816

Estándares internacionales para tarjetas con circuito integrado (tarjetas inteligentes). El objetivo de estos estándares es lograr la interoperabilidad entre distintos fabricantes de tarjetas inteligentes y lectores de las mismas, en lo que respecta a características físicas, comunicación de datos y seguridad. Estos estándares son basados en los ISO 7810 e ISO 7811, los cuales definen características físicas de tarjetas de identificación. (5)

La apariencia física de las tarjetas inteligentes está especificada por la Organización Internacional de Normalización y Comisión Electrotécnica Internacional.

ISO/IEC 14443

El estándar ISO 14443 está relacionado con las tarjetas de identificación electrónicas, en especial con las tarjetas inteligentes, gestionado conjuntamente por la ISO y la IEC. Este estándar define una tarjeta de proximidad, utilizada para identificación y pagos. (6)

PC/SC

PC/SC es un grupo de desarrollo cuyo objetivo es el de promover una especificación estándar, que asegure la interoperabilidad entre tarjetas inteligentes, lectores de tarjetas inteligentes y computadoras. PC/SC desarrolló una especificación independiente de la plataforma, que puede ser implementada sobre cualquier sistema operativo. Fue construido sobre los estándares actuales de Tarjetas Inteligentes, definiendo interfaces de bajo nivel para dispositivos y APIs independientes del dispositivo. La especificación actual es la PC/SC *Specification* 1.0. (4)

GlobalPlatform

Este estándar lidera mundialmente el desarrollo en temas de infraestructura de tarjetas inteligentes. Sus descripciones técnicas probadas para las tarjetas, dispositivos y sistemas son consideradas como las normas de la industria para lograr implementaciones interoperables, flexibles y sostenibles por tarjetas que soportan multi-aplicación y multi-actor e implementaciones multi-modelo de negocio. (2)

GlobalPlatform es un estándar para la administración de la estructura de las tarjetas inteligentes, comprende la instalación y borrado de las aplicaciones y la administración de otras tareas de las tarjetas. El emisor de la tarjeta tiene el control total sobre el contenido de esta, pero puede permitir a otras instituciones administrar sus propias aplicaciones, esto se logra aplicando protocolos criptográficos, permitiendo a cada institución tener su propia área segura en la tarjeta. (2)

La figura 1 representa un diagrama que muestra la arquitectura definida por *GlobalPlatform*.

Runtime Environment: comprende la máquina virtual de *Java Card* (JCVM) junto a clases y servicios definidos en la *Java Card Application Programming Interface* (API).

Security Domain: es un área protegida en la tarjeta inteligente, a estos dominios de seguridad se le asignan aplicaciones que pueden utilizar los servicios criptográficos que el dominio ofrece.

Card Manager: es el componente central de una tarjeta con arquitectura *GlobalPlatform*. Todos los servicios son ejecutados por él y ofrece interfaces para utilizar los servicios, internamente a través de las APIs de *GlobalPlatform* y externamente a través de los comandos APDU.

***GlobalPlatform* API**: permite el acceso a las funciones del CardManager y autenticar el terminal con la tarjeta, utilizando un canal seguro.

A continuación se muestra la arquitectura de las tarjetas inteligentes según *GlobalPlatform*

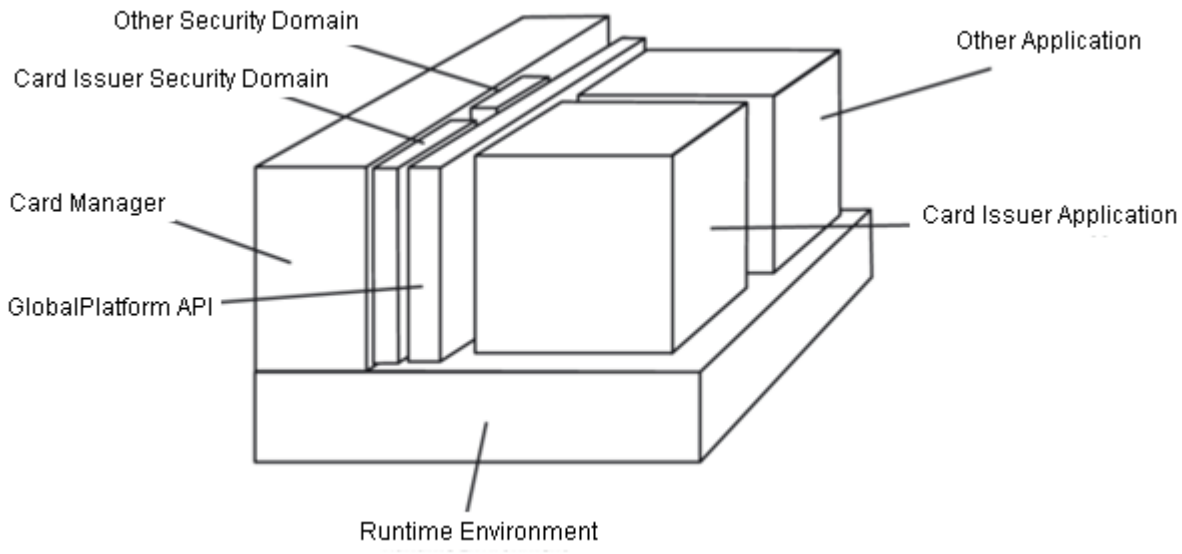

Figura 1: Arquitectura de las tarjetas según *GlobalPlatform*. Fuente: (2).

1.6. Tecnología en tarjetas inteligentes

1.6.1. Java Card

Esta tecnología permite en tarjetas inteligentes y similares dispositivos incrustados, ejecutar aplicaciones *Java*. Ofrece la posibilidad y capacidad al usuario de implementar soluciones aplicadas en distintas esferas de la sociedad y la economía, con elevados niveles de seguridad, ejemplo de ello son las aplicaciones que permiten la firma digital, la autenticación de usuarios, tarjetas de monedero electrónico y además en las tarjetas módulo de identificación del suscriptor (*SIM* por sus siglas en Inglés) utilizadas en la telefonía celular. (7)

Una tarjeta inteligente posee una estructura lógica además de su estructura física. El microprocesador es el encargado de realizar todas las operaciones, mientras el sistema operativo se encarga de traducir las acciones de las capas superiores en instrucciones que el microprocesador pueda entender. La máquina virtual de *Java Card* permite la ejecución de aplicaciones *Java Card* dentro de la tarjeta, mientras que las interfaces de programación de aplicación (*APIs* por sus siglas en Inglés) de *Java Card* poseen funcionalidades utilizadas por los *applets.* (7)

1.6.2.*Java Card Applet*

Un *applet* de *Java Card* es un programa de *Java* que cumple un conjunto de convenciones que permiten ejecutarlo en el entorno de ejecución de *Java Card*. Un *applet* de *Java Card* no está pensado para ejecutarse en un explorador. Estos se pueden cargar en el entorno de ejecución después de que la tarjeta haya sido fabricada, es decir a diferencia de las aplicaciones de muchos sistemas embebidos, los applets no necesitan ser “quemados” en la ROM durante el proceso de fabricación, más bien los applets pueden ser bajados dinámicamente a la tarjeta en un instante posterior a la fabricación. Un *applet* que se ejecuta en la tarjeta es una instancia de un *applet*, o sea un objeto de dicho *applet*, como ocurre con cualquier objeto persistente. El entorno de ejecución de *Java Card* soporta un entorno multi-aplicación, por lo que pueden coexistir múltiples *applets* en una sola tarjeta inteligente y un *applet* puede tener múltiples instancias. (8)

Los *applets* son las aplicaciones que corren dentro de una tarjeta *Java Card*. Dichas aplicaciones interactúan en todo momento con el *Java Card Runtime–Environment* (*JCRE)* utilizando los servicios que este brinda, e implementan la interfaz definida en la clase abstracta *javacard.framework.Applet.*

Una vez registrada en el *JCRE* (con el método register ()), entonces el *applet* está listo para ser ejecutado. La clase *javacard.framework.Applet* define cuatro métodos públicos que son utilizados por el *JCRE* para hacer funcionar las aplicaciones.

- Método install(byte(), short, byte)

El *JCRE* antes de crear la instancia del *applet* en la tarjeta, invoca a este método, cuya implementación habitual es llamar al constructor de la clase (generalmente este constructor es privado), crear todos los objetos necesarios para ejecutar el *applet* y por último registrar el *applet* con el método register ().

- Método select()

Como resultado de la recepción de un APDU, el *JCRE* solicita el método *SELECT APDU*. El *APDU* contiene el identificador de aplicación (*AID* por sus siglas en Inglés) del *applet* a seleccionar.

El *AID* es una secuencia de entre 5 y 16 bytes, donde la *ISO* asigna los 5 primeros bytes (*RID*) y los proveedores de aplicaciones definen los siguientes 11 bytes (*PIX*), esto hace que la aplicación sea identificada de forma única, de acuerdo a la *ISO/IEC 7816*, y es la propia *ISO* quien le otorga los *AIDs*. (8)

- Método process(APDU)

Cuando llega un *APDU* el *JCRE* invoca este método del *applet* seleccionado, pasándole como parámetro el *COMMAND APDU* recibido. Dentro de este método, el *applet* identifica el comando asociado al *APDU* y los parámetros, si los hay, los procesa de acuerdo al protocolo que se haya definido para la interacción entre el *applet* y la aplicación terminal.

- Método deselect()

El *JCRE* invoca a este método para avisar al *applet* que se encuentra en ese momento seleccionado que va a dejar de estarlo.

La tecnología *Java Card* tiene como principal objetivo llevar los beneficios del desarrollo del *software* orientado a objetos al mundo de las tarjetas inteligentes.

Finalmente, luego de haber realizado un estudio previo sobre *Java Card* se ha llegado a la conclusión de que esta es la única tecnología que permitirá desarrollar aplicaciones *Java* que corran dentro de las tarjetas inteligentes, siempre y cuando estas tarjetas tengan como sistema operativo *Java Card*.

1.7. Soluciones desarrolladas con tarjetas inteligentes

Varias son las empresas que se dedican hoy a desarrollar soluciones haciendo uso de los distintos tipos de tarjetas inteligentes. Una de las empresas líderes es Gemalto, esta ofreció una solución denominada SConnet, la cual tiene como objetivo principal, proporcionar un puente de conexión entre el *JavaScript*, que corre en la página *web* de un navegador y la tarjeta inteligente. Permitiendo la conectividad entre estas últimas y los servicios web. (9)

SConnect incluye un sistema de medidas de seguridad para mitigar ciertos riesgos, proteger a los usuarios y sitios web conectados. Dichas medidas incluyen una extensión con firma digital de *SConnect*, un HTTPS reforzado, llave de conexión, validación de servidor y alarma a usuario. (9)

Otra de las soluciones de Gemalto es Coesys eGov 2.0, producto que tiene como objetivo autenticar a los usuarios a través de la *Web* para que tengan acceso a los servicios en línea de gobierno electrónico. Permite un servicio de identificación electrónica mediante tarjetas inteligentes basado en la *Web*, en vez de un *software* basado en un cliente de autenticación de instalación local. (10)

Por su parte la empresa *I-Card Software* ofrece la solución Control Paterno, producto que hace referencia a ciertas funcionalidades de un determinado dispositivo o *software* que permite a los padres filtrar o aplicar restricciones sobre toda la navegación que hacen sus hijos en la *Web*. (11) Por otro lado la empresa Firma Profesional ofrece la solución *WebLogon*, producto que hace de la sencillez y velocidad de instalación su punto fuerte. Esta funciona como una pasarela intermedia, interceptando las peticiones de autenticación tradicional y modificándolas para realizar una verificación del certificado del usuario, que se encuentra en su tarjeta inteligente. Permite integrar en la red corporativa el acceso a cualquier página, propia o externa, con tarjeta inteligente y certificado, reforzando la seguridad. (12)

Otra de las empresas dedicadas a dar soporte al trabajo con las tarjetas inteligentes es MacroSeguridad, esta da a conocer la solución *Payment Card* que ofrece transacciones online seguras, que ayuda a disminuir los peligros de fraude y *phishing*[4] bancarios, por la incorporación de funciones lógicas de identificación y autenticación de usuarios. (13)

Por su parte la empresa *Athena SmartCard* brinda el producto *Payment and PKI*, esta herramienta permite a una audiencia global ver y comprar desde cualquier lugar bienes y servicios. Por el auge del comercio minorista en línea, *Athena* proporciona una generación de herramientas para proteger a los emisores y continuar el crecimiento minorista en línea. Ofreciendo una solución de pago que cumpla con todos los estándares relevantes de la industria. (14)

No obstante todas estas soluciones por su forma tradicional de interactuar con las tarjetas traen consigo algunas restricciones como son: el dominio que deben tener los usuarios para efectuar las actualizaciones

[4] *Phishing*: Es un término informático que denomina un tipo de delito encuadrado dentro del ámbito de las estafas cibernéticas, y que se comete mediante el uso de un tipo de ingeniería social caracterizado por intentar adquirir información confidencial de forma fraudulenta (como puede ser una contraseña o información detallada sobre tarjetas de crédito u otra información bancaria)

en la tarjeta, así como la necesidad de poseer una serie de permisos en el manejo de los recursos de la computadora para poder instalarlas. Además las llaves simétricas necesarias para trabajar dentro de la tarjeta, se encuentran en el cliente.

En la Universidad de las Ciencias Informáticas en el Centro de Identificación y Seguridad Digital en el departamento Tarjetas Inteligentes fue desarrollada una plataforma llamada *SmartCop* que integra diferentes servicios en línea de las aplicaciones de las tarjetas inteligentes, permitiendo la viabilidad y comodidad de uso. Utiliza el protocolo de comunicación *WebSocket*, aprovechando ventajas como el tiempo real de comunicación y los altos niveles de escalabilidad. La figura 2 muestra la arquitectura de la plataforma *SmartCoP*.

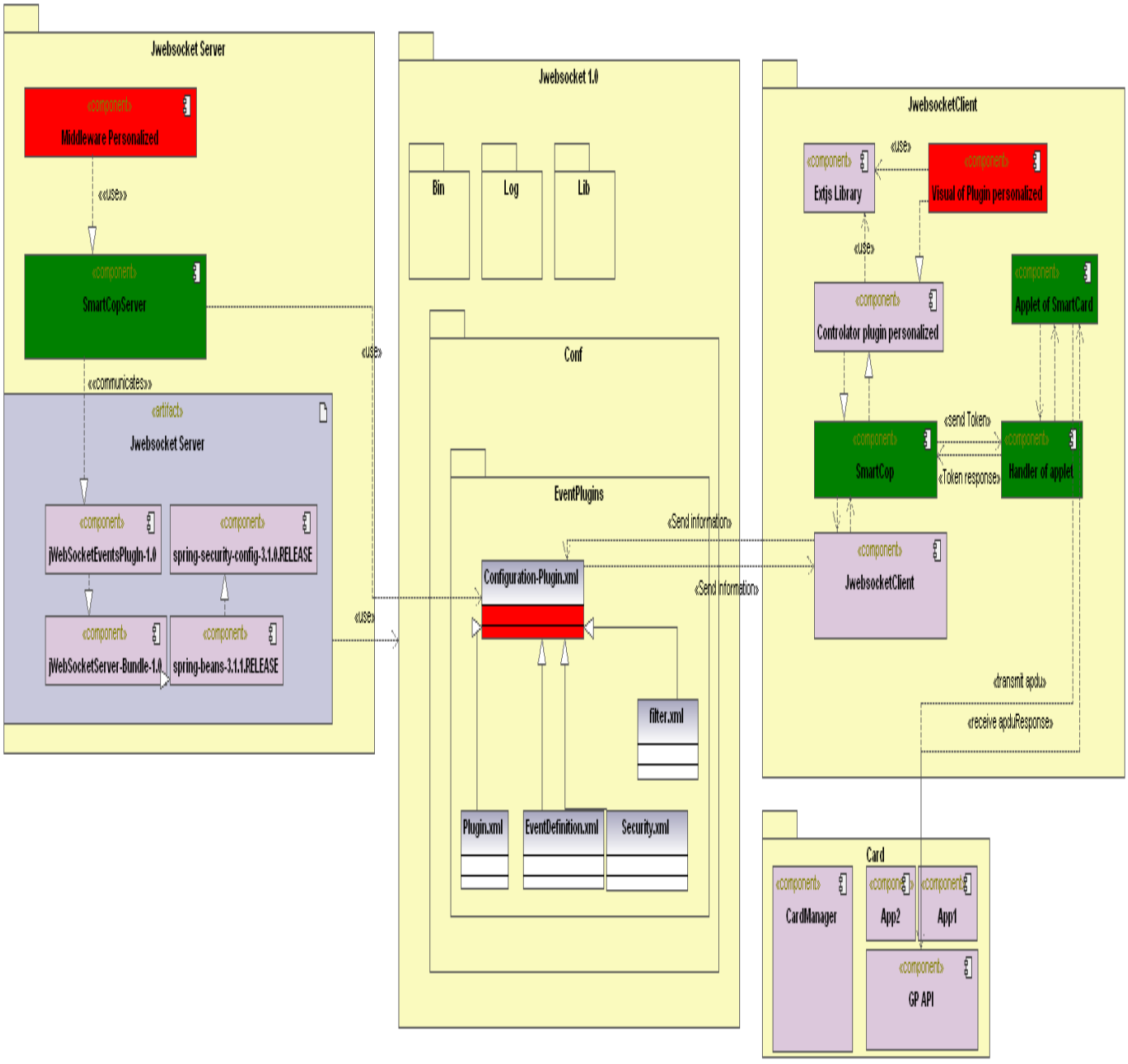

Figura 2: Arquitectura del *SmartCoP*. Fuente: Elaboración propia.

En el mismo departamento también se realizó el *framework SmartCardFramework*, que no es más que una herramienta desarrollada en .Net que permite la administración de tarjetas inteligentes y el envío de comandos *APDU* personalizables para aplicaciones específicas. Se desarrolló además la aplicación *SmartCardTool* que forma parte del *SmartCardFramework*, pero se centra específicamente en la administración de tarjetas con *GlobalPlatform*.

1.8. Middleware

Los *middlewares* han aparecido de manera relativamente reciente en el mundo de la informática. Funcionan como una capa de abstracción, situada entre la de aplicación y las inferiores (sistema operativo y red) haciendo posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. En general son usados para relacionar sistemas que necesitan intercambio de información, permitiendo realizar la conexión a través de interfaces de alto nivel. (15)

Entre sus funciones se encuentran:

- Transparencia de la heterogeneidad de los componentes de hardware, sistemas operativos y protocolos de comunicación.
- Proporcionan un estándar de alto nivel de interfaces para los desarrolladores e integradores de aplicaciones, para que estas puedan ser fácilmente integradas, reutilizadas, adaptadas y hechas para interoperar.
- Hacen más fácil el desarrollo de aplicaciones, ofreciendo abstracciones de programación común, mediante el enmascaramiento de la heterogeneidad, la distribución del hardware subyacente y sistemas operativos; además oculta los detalles de la programación de bajo nivel. Actualmente son muy usados para interactuar con las tarjetas inteligentes en los computadores personales ya que hacen función de intermediarios entre diversas aplicaciones y los lectores de tarjetas.
- Suministran un conjunto de servicios comunes a diversas funciones de propósito general, a fin de evitar la duplicación de esfuerzos y facilitar la colaboración entre las aplicaciones. (15)

1.9. Navegador *Web*

Un navegador es un programa que permite visualizar la información que contiene una página *web*, interpreta el código de la página y lo presenta en el monitor de la computadora permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos. Los documentos pueden estar ubicados en la computadora en donde está el usuario, pero también

pueden estar en cualquier otro dispositivo que esté conectado a la computadora del usuario o a través de Internet y que tenga los recursos necesarios para la transmisión de estos, como por ejemplo un servidor *web*. Tales documentos, comúnmente denominados páginas *web*, poseen hipervínculos que enlazan una porción de texto o una imagen a otro, normalmente relacionado. La comunicación entre el servidor *web* y el navegador se realiza generalmente mediante el protocolo HTTP. (16)

Entre los navegadores más utilizados a nivel mundial se encuentran *Internet Explorer (IE)* desarrollado por *Windows*, siendo el más utilizado desde 1999, lo que ha ido disminuyendo paulatinamente por la renovada competencia de navegadores como *Mozilla Firefox*. Desarrollado por la Fundación *Mozilla* y un gran número de voluntarios externos, es un navegador multiplataforma y de código fuente abierto (*open source*). El 2 de septiembre de 2008 *Google Chrome* vio la luz. Es el navegador *web* desarrollado por *Google* y compilado con base en componentes de código abierto y su estructura de desarrollo de aplicaciones (*Framework*). *Google Chrome* es el navegador más utilizado en Internet y actualmente posee una cuota de mercado del 27,20%. Está disponible gratuitamente bajo condiciones de servicio específicas. (16)

1.10. WebSocket

Desde sus inicios la comunicación en la *Web* fue soportada por el protocolo HTTP (*Hypertext Transfer Protocol*). Inicialmente diseñado como protocolo de solicitud-respuesta para el intercambio de documentos, HTTP no logra satisfacer las nuevas necesidades de la red de redes. A lo largo de los últimos años, se han implementado varias técnicas y tecnologías en aras de lograr una comunicación *web* en tiempo real y bidireccional. Técnicas como el *polling*, *longpolling*, *streaming* y tecnologías como *AJAX* y *Comet* han formado parte de este esfuerzo, logrando algunas de ellas resultados favorables pero muy costosos. (18)

Sin embargo el protocolo *WebSocket* define los procedimientos para actualizar la conexión a través de HTTP a una conexión mediante *WebSocket* totalmente bidireccional usando TCP (*Transmission Control Protocol*), logrando el tiempo real con altos niveles de escalabilidad. *WebSocket* es una tecnología que proporciona un canal de comunicación bidireccional y *full-duplex* sobre un único *socket* TCP. Soluciona las limitaciones del protocolo HTTP, al establecer una comunicación *full-duplex* (TCP) entre el cliente y el servidor, sustituyendo la comunicación *half-duplex*[5] (HTTP). Se reduce, en grandes proporciones, el tráfico

[5] *Half-duplex*: método o protocolo de envío de información es bidireccional pero no simultáneo

en la red teniendo en cuenta que al establecer la comunicación *WebSockets* entre el cliente y el servidor solo hay un envío de 2 bits, eliminando las cabeceras HTTP. (18)

Los principales servidores que soportan WebSockets para el desarrollo de aplicaciones hoy día son, la pasarela *WebSockets* de *Kaazing*, *JettyWebSocketServlet*, *Windows Server*, *Socket.IO*, *django-websocket* del proyecto *Python* y *JWebSocket*.

1.11. Metodologías a emplear para el desarrollo de la solución

Las metodologías orientadas al desarrollo de *software* no son más que el conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo *software*.

En los inicios los desarrolladores creaban *software* de manera artesanal sin ningún tipo de guía o patrones y esto provocaba que la mayoría de los grandes proyectos fueran abandonados o estuvieran colmados de errores, muchas veces con carácter irreversible.

Con el pasar de los años y la experiencia de las empresas dedicadas al desarrollo de *software* se fueron creando métodos y patrones que conducían a resultados eficientes y con calidad. Estos métodos y patrones evolucionaron para convertirse en las metodologías de desarrollo de *software* que se conocen en la actualidad. (19)

1.11.1. Metodologías Tradicionales

Las metodologías tradicionales ofrecen cierta resistencia a cambios que sean necesarios durante el ciclo de vida del proyecto. En este tipo de metodología el proceso es muy controlado, con mucha documentación, plantillas, normas, etcétera. Se realiza un contrato prefijado y el cliente puede interactuar con el equipo de desarrollo del proyecto. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada, además la arquitectura de *software* es esencial y se expresa mediante modelos.

***Rational Unified Process* (RUP)**

Esta metodología constituye un proceso formal, cuyo objetivo fundamental se centra en la producción de *software* con alto nivel de calidad. Satisfacer los requerimientos de los usuarios finales también es premisa para RUP, todos estos requisitos deben cumplirse sin violar el cronograma ni el presupuesto propuesto.

RUP es un proceso iterativo e incremental, centrado en la arquitectura y guiado por los caso de uso, utiliza UML como lenguaje de notación. Incluye artefactos y roles.

1.11.2. Metodologías ágiles

Las metodologías ágiles intentan evitar los tortuosos y burocráticos caminos de las metodologías tradicionales enfocándose en los clientes y los resultados. Se basan en promover iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto. Estas logran que se minimicen los riesgos desarrollando aplicaciones en corto tiempo. Entre las metodologías ágiles más destacadas se encuentra SCRUM y XP. (19)

Programación Extrema – XP

XP es una metodología ligera de desarrollo de *software* que se basa en la simplicidad, la comunicación y la realimentación o reutilización del código desarrollado. Esta consiste en una programación rápida o extrema y es utilizada para proyectos de corto plazo, con requisitos imprecisos y muy cambiantes, donde existe un alto riesgo técnico. Esta metodología es basada en pruebas unitarias, la re-fabricación y la programación en pares. El ciclo de vida ideal de esta metodología consta de 6 fases: Exploración, Planificación de la Entrega, Iteraciones, Producción, Mantenimiento y Muerte del Proyecto. (19)

En el ciclo de vida de un proyecto los cambios siempre van a aparecer y a veces el equipo de desarrollo no está preparado para enfrentarlos. XP cuenta con los siguientes valores para garantizar el éxito de un proyecto de desarrollo de *software*:

- Comunicación
- Coraje
- Simplicidad
- Retroalimentación

La mayoría de las siguientes prácticas no son nuevas, sino que han sido reconocidas por la industria como las mejores durante años. (19)

- Planificación Incremental
- Pruebas
- Programación en parejas
- Refactorización

- Diseño simple
- Propiedad colectiva del código
- Integración continua
- Cliente en el equipo
- Entregas pequeñas
- Semanas de 40 horas
- Estándares de codificación
- Uso de Metáforas

XP define cuatro variables para proyectos de *software*: coste, tiempo, calidad y ámbito. Además de estas cuatro variables, Beck el creador esta metodología, propone que sólo tres puedan ser establecidas por las fuerzas externas (jefes de proyecto y clientes), mientras que el valor de la cuarta variable debe ser establecido por los programadores en función de las otras tres. (19)

Fundamentación de XP como metodología a utilizar

Aspectos esenciales que incidieron para la elección de esta metodología:

- Necesidad de obtener un producto a corto plazo.
- Cantidad de miembros del equipo de desarrollo (dos personas).
- Presencia del cliente en el grupo de desarrollo.
- Asumir una metodología robusta implica una cantidad excesiva de roles y gran volumen de información generada durante todo el ciclo de vida del proyecto. Es por ello que se hace difícil la utilización de esta metodología por un equipo pequeño.

1.12. Lenguajes de programación

1.12.1. *Java*

Es un lenguaje de programación orientado a objetos, moderno y de alto nivel, desarrollado por *Sun Microsystems* a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir muchos errores, como la manipulación directa de punteros o memoria.

Sun Microsystems liberó la mayor parte de sus tecnologías *Java* bajo la licencia GNU GPL. Actualmente este lenguaje es el más utilizado para el desarrollo de aplicaciones por las ventajas que brindan sus características. *Java* es el lenguaje utilizado para programar las aplicaciones en el lado del servidor con el marco de trabajo *JWebSocket*. (20)

1.12.2. *JavaScript*

JavaScript es un lenguaje de *scripting*[6] orientado a objetos, utilizado para crear páginas *web* dinámicas y aplicar cierta lógica programada del lado del cliente. Principalmente se utiliza integrado en un navegador *web* permitiendo el desarrollo de interfaces de usuario mejoradas y páginas *web* dinámicas, todos los navegadores modernos interpretan código *JavaScript*. Ha tenido influencia de múltiples lenguajes y se diseñó con una sintaxis similar al lenguaje de programación *Java*, aunque solo comparten algunos conceptos básicos.

JavaScript se ejecuta en el navegador (cliente) y no requiere ningún *software* de servidor. Dado que toda la ejecución tiene lugar en el navegador, *JavaScript* es responsable de la mayor parte de la interactividad en una página *web*. A través de él se pueden hacer acciones como cambiar la imagen, el color del texto cuando se pasa el ratón. El lenguaje también ha sido ampliamente utilizado para la validación de datos, ya que es mejor para validar un formulario en el lado del cliente que hacer varias solicitudes al servidor. (21)

1.13. Marcos de trabajo

1.13.1. *JWebSocket*

JWebSocket es un marco de trabajo de código abierto para el desarrollo de aplicaciones *web* estacionarias y móviles basado en *Java* en el lado del servidor y en *JavaScript* del lado del cliente. *JWebSocket* establece un modelo de comunicación basado en *tokens*, estos son datos abstractos que a través de una estructura jerárquica y una API proporcionan métodos de acceso a los contenidos. (22)

Con el objetivo de realizar una abstracción en la manipulación de los diferentes formatos, el marco de trabajo convierte los paquetes de datos entrantes y salientes en *tokens*.

Ventajas de *JWebSocket* (22)

[6]Un *script* o archivo de procesamiento por lotes es un programa usualmente simple, que por lo general se almacena en un archivo de texto plano.

- El servidor *JWebSocket* está diseñado para funcionar como servidor de comunicaciones o como servidor *web*, brindando total flexibilidad.
- Se ejecuta fácilmente desde una línea de comandos o se integra a la biblioteca de una aplicación existente de *Java*.
- *JWebSocket* como servidor *web* proporciona un conjunto importante de funcionalidades y su arquitectura extensible permite añadir fácilmente características adicionales a un sistema independiente.
- Los administradores pueden configurar el servidor exactamente como sea necesario y dejar a un lado todos los módulos que no sean imprescindibles.
- En un clúster las extensiones se pueden utilizar como servicios, por lo que *JWebSocket* perfectamente es compatible con SOA (*Service Oriented Architectures*) en un entorno totalmente basado en eventos.
- Es el marco de trabajo y a la vez el servidor de aplicaciones con licencia libre más robusto y flexible para la plataforma *Java*.
- Está respaldado por excelentes resultados en cuanto a rendimiento y escalabilidad.
- Su pequeña curva de aprendizaje le permite a los desarrolladores crear y desplegar nuevas aplicaciones en corto tiempo.

Estas ventajas muestran la fortaleza y flexibilidad del marco de trabajo para el desarrollo de aplicaciones *Web* estacionarias y móviles, multiplataformas, multisectoriales y compatibles con todos los navegadores. (22)

1.13.2. *AJAX*

Es una técnica de desarrollo *web* para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones. (23)

Ajax es un conjunto de tecnologías asíncronas, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página. *JavaScript* es el lenguaje interpretado en el que normalmente se efectúan las funciones de llamada de *Ajax* mientras que el acceso a los datos se realiza mediante *XMLHttpRequest*, objeto disponible en los

navegadores actuales. En cualquier caso, no es necesario que el contenido asíncrono esté formateado en XML. (23)

Ajax es una técnica válida para múltiples plataformas y utilizable en muchos sistemas operativos y navegadores, dado que está basado en estándares abiertos como *JavaScript* y *DocumentObjectModel* (DOM).

1.14. UML (*Unified Modeling Language*)

De los lenguajes de modelado de sistemas de *software* el más conocido y utilizado actualmente es el Lenguaje Unificado de Modelado. Es un lenguaje de modelado visual que se usa para visualizar, especificar, construir y documentar un sistema de *software*.
Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios. Está concebido para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de *UML* no define un proceso estándar pero está definido para ser útil en un proceso de desarrollo iterativo. (24)

1.15. Altova UModel

Herramienta utilizada para crear e interpretar diseños de *software* mediante la potencia del estándar UML 2.1. Dibuja el diseño de la aplicación y puede generar código para *Java* o C# a partir de diagramas, además permite realizar ingeniería inversa de programas existentes a diagramas UML claros y precisos para abarcar rápidamente su arquitectura.

Con su utilización se puede corregir el código generado o los modelos y completar la ronda produciendo automáticamente diagramas nuevos o regenerando el código. Se utilizó esta herramienta de modelado por todas las características anteriormente expuestas y por ser la herramienta definida por el departamento de tarjetas inteligentes. (25)

1.16. NetBeans IDE

Es un proyecto de código abierto fundado por *Sun Microsystems* especialmente diseñado para el desarrollo de aplicaciones en *Java*, pero acepta otros lenguajes de programación. Consta de una gran base de usuarios y una comunidad en constante crecimiento, lo que le ha permitido, al igual que muchos

otros sistemas libres, el progreso paulatino de sus prestaciones y la eliminación de *Bugs* que pudiesen existir.

Ventajas: la plataforma *NetBeans* permite que las aplicaciones se desarrollen a partir de un conjunto de módulos o componentes de *software*. Un módulo contiene clases de *java* escritas para interactuar con las APIs de *NetBeans* y un archivo especial que lo identifica como módulo. *NetBeans* IDE (Entorno de desarrollo integrado) es fácil de instalar y de uso instantáneo y se ejecuta en varias plataformas incluyendo Windows, Linux, Mac OS y Solaris. Además del soporte completo para todas las plataformas *Java* (*Java* SE, *Java* EE, *Java* ME, y *Java* FX), *NetBeans* IDE es además la herramienta ideal para el desarrollo de *software* con PHP, *Ajax* y *Javascript* y C/C++. (26) Por todas las razones antes expuestas se escogió como herramienta de implementación en el servidor.

1.17. Eclipse

Eclipse fue desarrollado originalmente por IBM. Es ahora desarrollado por la Fundación Eclipse, una organización independiente que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Es un entorno de desarrollo integrado, de código abierto y multiplataforma. (26)

Mayoritariamente se utiliza para desarrollar lo que se conoce como "aplicaciones de cliente enriquecido", opuesto a las aplicaciones "cliente liviano" basadas en navegadores, es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios *web*, programas en C++ o aplicaciones *Java.* (27)

1.18. JCardManager

JCardManager es una solución de Gemalto para permitir a cualquier desarrollador de *Java Card* administrar y probar nuevas aplicaciones con *Java Card* y *GlobalPlatform*. Sus herramientas ofrecen la mejor asistencia en el proceso de desarrollo de la aplicación e incluyen todos los elementos necesarios para ejecutar y probar una aplicación. Posibilita que los desarrolladores puedan escribir *applets* y ejecutarlos en cualquier tarjeta *Java*, independientemente del sistema operativo o del chip usado. (27)

Presenta una interfaz gráfica de usuario amigable. Permite enviar comandos a diferentes plataformas globales o APDU propiedad de la tarjeta inteligente. Ofrece la posibilidad de cargar aplicaciones, para gestionar la tarjeta inteligente, para jugar o para registrar la escritura.

Permite grabar, editar y reproducir scripts. Recalcula automáticamente criptogramas y MAC, lo que permite la repetición de comandos avanzados sean cuales sean las claves de sesión. Es la herramienta perfecta para la producción de volúmenes pequeños. Todos los elementos configurables del *Java Card* para abrir un canal seguro se pueden especificar con esta herramienta. (27)

En el sistema desarrollado se utiliza el *JCardManager* específicamente para la comunicación con la tarjeta, ya que permite abrir el canal seguro de la misma.

1.19. Developer Suite

Developer Suite proporciona un conjunto de herramientas de desarrollo para crear y depurar *applets Java Card*. Permite trabajar desde dentro del IDE de desarrollo para implementar *applets*. Este además facilita el desarrollo de soluciones inalámbricas para los desarrolladores de *Java*. Por lo tanto, es un desarrollador sin problemas que comprende las particularidades de *Java Card*. Además brinda un ambiente favorable para el diseño y la implementación de *applets* y posibilita simular las funcionalidades de los *applets* antes de ser instalados en las tarjetas inteligentes. (27)

Developer Suite viene con una simulación de extremo a extremo *Suite*:

- Tarjeta de Simuladores de tarjetas SIM, tarjetas USIM y tarjetas R-UIM, y las tarjetas NFC SCWS
- Simuladores de móviles 2G, 3G y CDMA
- Simuladores Server para 2G, 3G y SCWS OMA

En este caso no se hace uso directo de la herramienta *Developer Suite,* pero esta es necesaria para el trabajo con tarjetas inteligentes que implementen *Java Card puesto* que permite la elaboración de aplicaciones *Java Card.*

1.20. Conclusiones del capítulo

- El análisis de las soluciones existentes sobre el uso de tarjetas inteligentes en aplicaciones *web* permitió identificar los principales elementos de seguridad y los estándares asociados a este tipo de tecnologías, así como las deficiencias que estas presentan. Se seleccionó la plataforma *SmartCoP* como tecnología a utilizar en la implementación de la solución propuesta.

- Partiendo del análisis realizado se pudo identificar como metodología a utilizar XP, las herramientas *Altova UModel* y *NetBeans* IDE, por ser las más completas y ágiles para el desarrollo

de la solución que se pretende implementar; además *Java* como lenguaje en el servidor y JavaScript en el lado del cliente.

Capítulo 2. Características, análisis y diseño del sistema

2.1. Introducción

En el presente capítulo se interpretan las necesidades del sistema especificándolas mediante los requerimientos funcionales y no funcionales. Se realiza el modelo del dominio con el objetivo de comprender el contexto, identificando para esto las entidades principales y las relaciones entre ellas. Además se detallan brevemente los artefactos de la metodología de desarrollo XP, propuesta en el capítulo anterior. Describiéndose las historias de usuario y las tareas de ingeniería asociadas a las mismas.

2.2. Modelo de dominio

Con el estudio de las tecnologías y estado del arte de la administración de tarjetas inteligentes se identificaron varios conceptos que definen la solución del problema a resolver, con lo que se determinó el dominio de solución. En la figura 3 se muestra el diagrama del modelo de dominio.

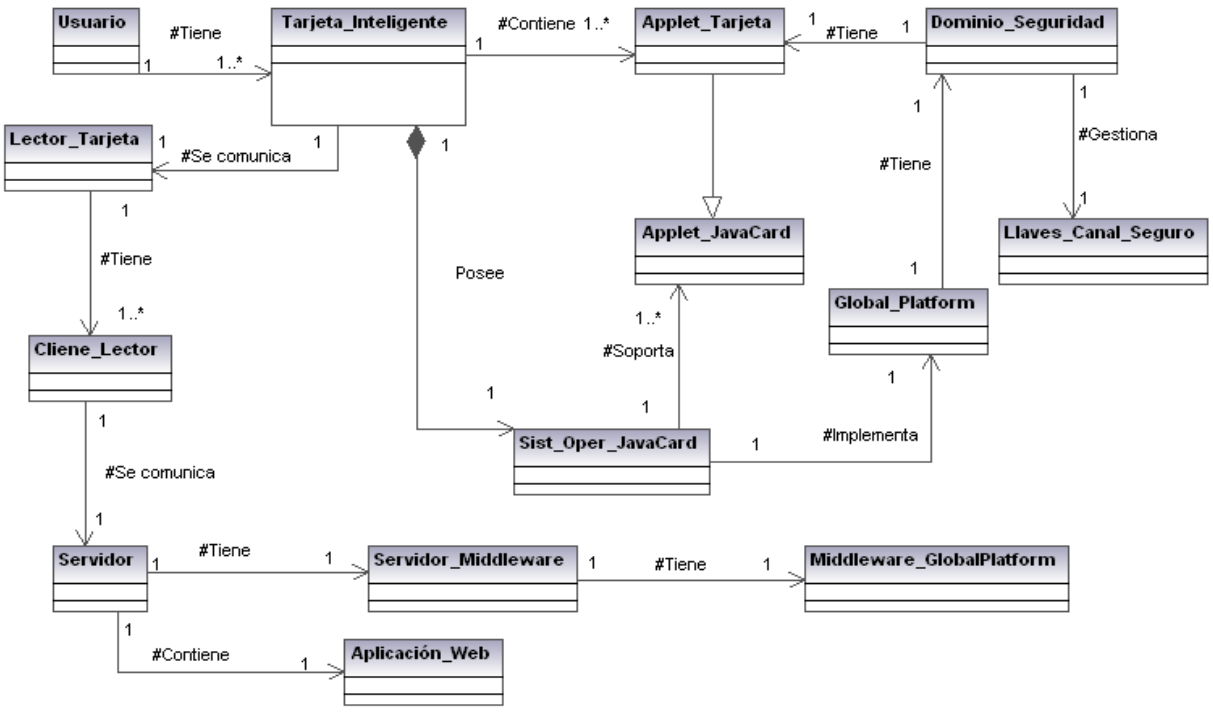

Figura 3: Modelo de dominio. Fuente: Elaboración propia.

A continuación se definirán los conceptos tratados en el modelo de dominio dentro del contexto del problema a resolver.

Usuario: tiene y usa la tarjeta inteligente.

Tarjeta_Inteligente: es una tarjeta plástica del tamaño de una tarjeta de crédito convencional, que contiene un pequeño microprocesador, que es capaz de hacer diferentes cálculos, guardar información y manejar programas, que están protegidos a través de mecanismos avanzados de seguridad.

Sist_Oper_*JavaCard*: es una tecnología que permite ejecutar de forma segura pequeñas aplicaciones *Java* (*applets*) en tarjetas inteligentes y similares dispositivos empotrados.

***Applet_JavaCard*:** los *applets* son las aplicaciones que corren embebidas en una tarjeta *Java Card*. Dichas aplicaciones interactúan en todo momento con el *JCRE* utilizando los servicios que este brinda.

***Applet*_Tarjeta:** aplicación *Java Card* para tarjetas inteligentes, que cumple con los principales estándares internacionales asociados, y con la capacidad de realizar operaciones criptográficas y de gestión de certificados digitales, en una Infraestructura de Clave Pública. Estos son creados en el *Developer Suite* respondiendo a determinados requerimientos del usuario.

Llaves_Canal_Seguro: los datos intercambiados entre los dispositivos son encriptados para mayor seguridad. Esto lo hacen con la utilización de llaves.

***GlobalPlatform*:** estándar para la administración de la estructura de las tarjetas inteligentes, comprende la instalación, borrado de las aplicaciones y la administración de otras tareas de las tarjetas.

Lector_Tarjeta: dispositivo que permite la comunicación con la tarjeta.

Cliente_Lector: cliente que tiene el lector para comunicarse con las tarjetas y que a su vez se comunica con el servidor.

Servidor: aplicación informática o programa donde varios componentes se relacionan entre sí para resolver las solicitudes que vienen desde el cliente, se encuentra asociado a la plataforma *SmartCoP*.

Servidor_*Middleware*: aplicación informática o programa donde varios componentes se relacionan entre sí para resolver las solicitudes que vienen desde el cliente, además de gestionar los *middlewares* que se

encuentran asociados a la plataforma *SmartCoP* y las funcionalidades contenidas en ellos accediendo a sus librerías.

***Middleware_GlobalPlatform*:** es un *software* que asiste en este caso a *GlobalPlatform*, para comunicarse con la tarjeta.

Dominio_Seguridad: constituye un contexto de seguridad por defecto que puede contener varias llaves, que pueden ser simétricas o asimétricas. Estos conjuntos de llaves son utilizados en los protocolos de autenticación mutua y canal seguro, permitiendo garantizar la autenticidad, integridad, confidencialidad en cualquier sesión de comunicación entre el terminal y la tarjeta.

Aplicación_*Web*: representa a la aplicación que interactúa con el cliente de forma directa y que a su vez está contenida en el servidor.

2.3. Historias de Usuario

Las historias de usuarios se utilizan en la metodología XP para describir los requisitos de una forma menos detallada y desde la perspectiva del cliente. Una historia de usuario debe responder a las preguntas: ¿Quién se beneficia?, ¿Qué se quiere? y ¿Cuál es el beneficio?

La tabla 1 muestra la historia de usuario *Authentication* y la tabla 2 muestra la historia de usuario *Get Status*.

HU_1 *Authentication*

Historia de Usuario	
Número: HU_1	**Nombre de Historia de Usuario:** *Authentication*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 1
Prioridad en negocio: Alta	**Puntos estimados:** 1.0
Riesgo en desarrollo: Alto	**Puntos reales:** 0.9
Descripción: Permite la comunicación con la tarjeta. Para ello inicialmente se conecta la tarjeta y se envía el identificador del *applet* del *javaCardManager* y el nivel de seguridad, lo que posibilita que se abra la comunicación entre él mismo y el *middleware*.	
Observaciones: Responderá con un error en caso que:	

No se encuentren las llaves necesarias para la autenticación. La información brindada por los dispositivos que aceptan tarjetas inteligentes no coincida con la generada por la tarjeta.

Tabla 1 - HU *Authentication*

HU_2 *Get Status*

Historia de Usuario	
Número: HU_2	**Nombre de Historia de Usuario:** *Get Status*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 1
Prioridad en negocio: Alta	**Puntos estimados:** 1.0
Riesgo en desarrollo: Medio	**Puntos reales:** 1.1
Descripción: Permite obtener el estado de una aplicación. Para ello se muestran los identificadores de cada *applet*, al igual que el estado de ciclo de vida de la tarjeta (cargada, personalizada o seleccionable)	
Observaciones: Responderá con una advertencia en caso que: • El usuario no seleccione todos los identificadores de los *applets*. Responderá con una error en caso que: • El identificador del que se desea obtener el estado del *applet* no coincida con el identificador de los *applets* de la tarjeta. • Se introduzca un caracter extraño o una letra que no pertenezca al lenguaje numérico hexadecimal.	

Tabla 2 - HU *Get Status*

El resto de las historias de usuario se encuentran en el anexo 3 en las tablas 22, 23, 24, 25.

2.4. Requerimientos no funcionales

Un requisito no funcional es una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo. Constituyen todas las exigencias de cualidades que se imponen al proyecto, ya sean exigencias de usar un cierto lenguaje de programación o una plataforma tecnológica específica. (29)

RNF1.*Software*

➢ Navegadores *web* a partir de la versión que soporta el protocolo *WebScoket*, entre los más populares se pueden citar *Google Chrome* versión 16.0, *Mozilla Firefox* versión 10.0, *Internet Explorer* 9.0.

➢ *Java Runtime Environment (JRE*) permite ejecutar el *applet* en el navegador.

➢ Controladores que cumplan con el estándar PC/SC.

RNF2.Apariencia o interfaz externa

➢ La aplicación deberá tener una interfaz externa amigable, que sea sencilla y fácil de entender por el desarrollador que la emplee.

RNF3. Restricciones en el diseño y la implementación

➢ Se empleará el protocolo *WebSocket*, debido a que el *middleware* a desarrollar estará integrado a la plataforma *SmartCoP* que lo utiliza para la comunicación.

RNF4. Seguridad

➢ **Confidencialidad:** la comunicación entre la tarjeta y el navegador se corresponderá con el nivel de seguridad seleccionado por el desarrollador al autenticarse (*No Security*, *MAC* o *MACEncryption*) y según los mecanismos definidos en el estándar *GlobalPlatform* 2.1.1 puesto que el mismo provee un canal de comunicación seguro.

➢ **Disponibilidad**: los mecanismos utilizados para lograr la seguridad (definir tipo de seguridad que puede ser *No Security*, *MAC* o *MACEncryption*), no ocultarán o retrasarán a los desarrolladores para obtener los datos deseados en un momento dado.

RNF5. Hardware

➢ Lector de tarjetas incorporado a la PC que cumpla con el estándar PC/SC versión 1.0 o superior.

RNF6. Conectividad

➢ Si se accede a una red que requiere el uso de *proxy*, esta debe permitir *WebSocket*, en caso contrario no debe usarse *proxy*.

2.5. Diseño de la solución

Después de tener divididas las historias de usuarios en tareas, estas deben ser transformadas a código, para esto se llevan a cabo encuentros donde se presentan las tarjetas Clase Responsabilidad Colaboración (CRC), las mismas dan una idea de la cantidad de clases a implementar y sus responsabilidades, todo esto siempre a un alto nivel. Las tarjetas CRC deben ser simples, describiendo la responsabilidad de una clase en pocas frases así como las clases con las que interactúa.

La metodología XP propone algunos principios para agilizar el proceso de desarrollo, simplicidad en el diseño, ya que es mucho más rápido de realizar y mucho menos complejo de entender. El uso de tarjetas CRC logra la contribución de todo el equipo al diseñar la solución. Se recomienda mantener las funcionalidades extras alejadas del trabajo primario, ya que las mismas no son tan necesarias y hacen perder mucho tiempo.

El diseño de la solución se realizó por iteraciones sustituyéndose las tarjetas CRC por diagramas de clases de UML (Ver figuras 7, 13 y 14), lo que no es aconsejable según varios autores, pero fue posible dado que la complejidad del problema a resolver no implicaba la elaboración de diagramas complejos, cumpliendo con los principios antes mencionados.

2.6. Metáfora

La forma tradicional de administrar tarjetas inteligentes hasta el momento es a través de *middlewares* instalados en el cliente que usan claves simétricas posibilitando que estas puedan ser obtenidas por atacantes. La realización de una aplicación *web* para administrar tarjetas inteligentes elimina el riesgo antes mencionado, pues el *middleware* desarrollado se encuentra en el servidor. Además la utilización del estándar *GlobalPlatform* 2.1.1, permite aumentar el nivel de seguridad en las tarjetas, ya que con el empleo de dicho estándar el emisor de la tarjeta tiene el control total sobre el contenido de esta, pero puede hacer que otras instituciones administren sus propias aplicaciones, garantizando que cada institución tenga su propia área segura en la tarjeta.

2.7. Arquitectura

La figura 4 muestra la arquitectura del sistema, la cual es de tipo cliente-servidor y está estructurada por componentes.

Cliente-servidor: la utilización de este patrón posibilita que se mantenga una clara separación entre la lógica de negocio, la presentación y el acceso a datos. Permitiendo flexibilidad y facilidad a la hora de realizar posibles modificaciones.

Basada en componentes: cada componente debe describir de forma completa las interfaces que ofrece, así como las interfaces que requiere para su operación, su correcto funcionamiento con independencia de los mecanismos internos que utilice para soportar la funcionalidad de la interfaz. Características muy relevantes de esta arquitectura son la modularidad y la reusabilidad.
En el lado del cliente se encuentran los siguientes componentes:

-InterfazGP

-*SmartCoP*

-*JWebSocketClient*

El cliente se comunica con la tarjeta inteligente a través de comandos APDU y a su vez con el servidor mediante el protocolo *WebSocket*.

El servidor se divide en dos componentes con el objetivo de facilitar el trabajo: Un servidor de configuraciones que posee un componente *xml* denominado *ConfigurationPlugin* y un servidor *JWebSocket* que incluye los componentes que a continuación se muestran:

-*JWebSocketServer*

-*SmartCoPServer*

-*Middleware*GP

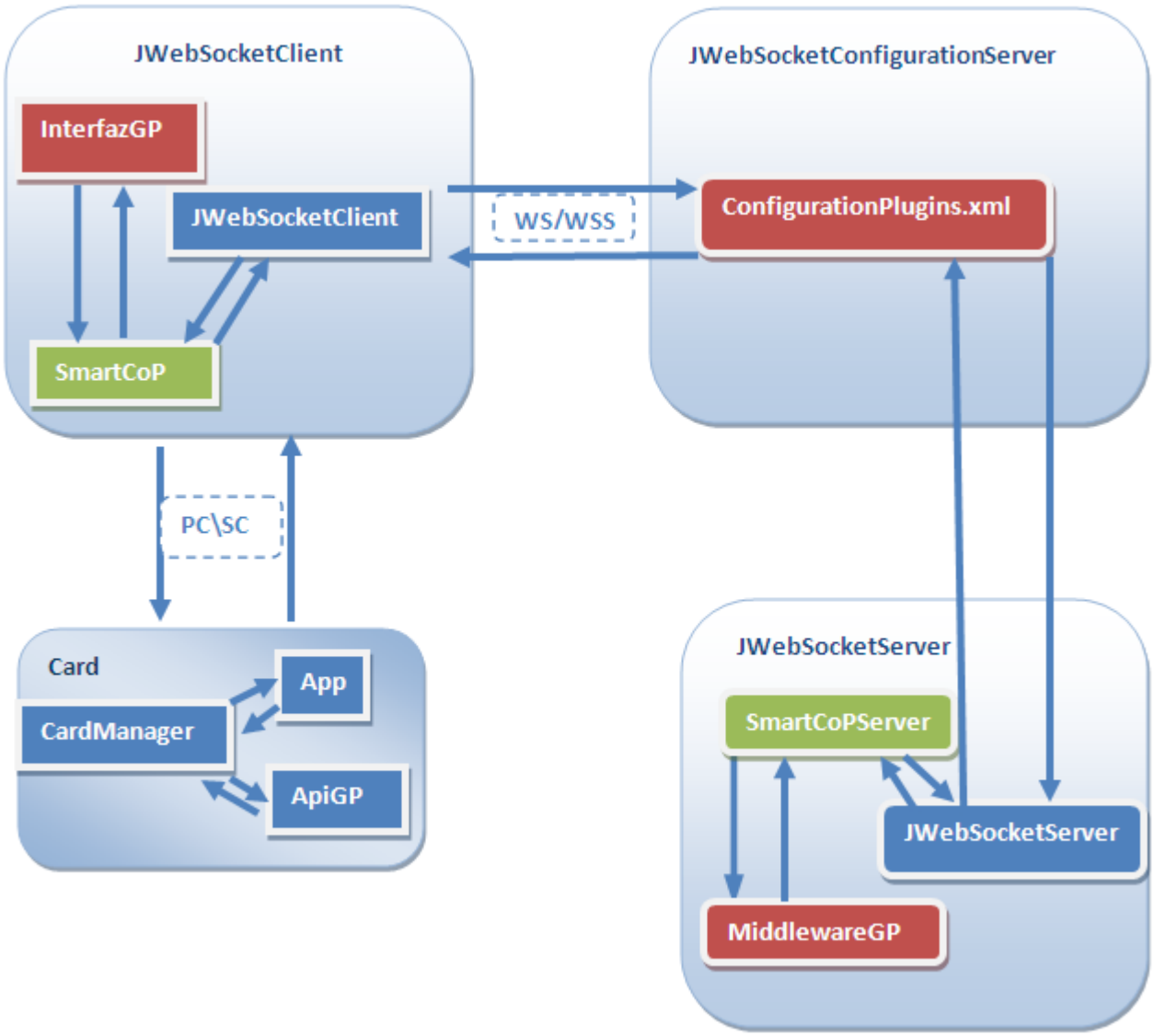

Figura 4: Arquitectura del sistema. Fuente: Elaboración propia.

Arquitectura del cliente:

El cliente posee los siguientes componentes:

-Principal

-Vistas

-Controladora

El componente principal es el encargado de vincular los restantes componentes. El componentes Vistas posee las interfaces que se muestran al usuario y el componente Controladora tiene las funcionalidades para la gestión de las transacciones efectuadas por el usuario y la validación de los datos introducidos.

El cliente utiliza las librerías de *EXTJS* 4.1 para la visualización de la página *web*, también el complemento *JCManager*, que maneja la comunicación con la tarjeta y el complemento *JWSPlugin* empleado para establecer los vínculos entre el Cliente y el Servidor.

La figura 5 permite visualizar la arquitectura del cliente.

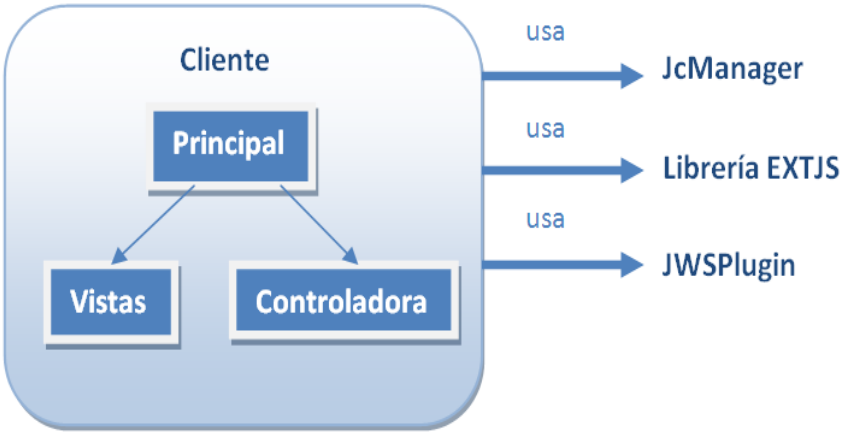

Figura 5: Arquitectura del cliente. Fuente: Elaboración propia.

Arquitectura del servidor:

El servidor posee los siguientes componentes:

-*GlobalPlatformPlugIn*

-Eventos

-Comandos

-Clases auxiliares

El componente principal del servidor es el *GlobalPlatformPlugIn*, el mismo gestiona un grupo de eventos, utiliza clases auxiliares y comandos para la gestión de la información. Cada evento puede tener implícito para su realización uno o varios comandos.

La figura 6 permite visualizar de forma detallada la arquitectura del servidor.

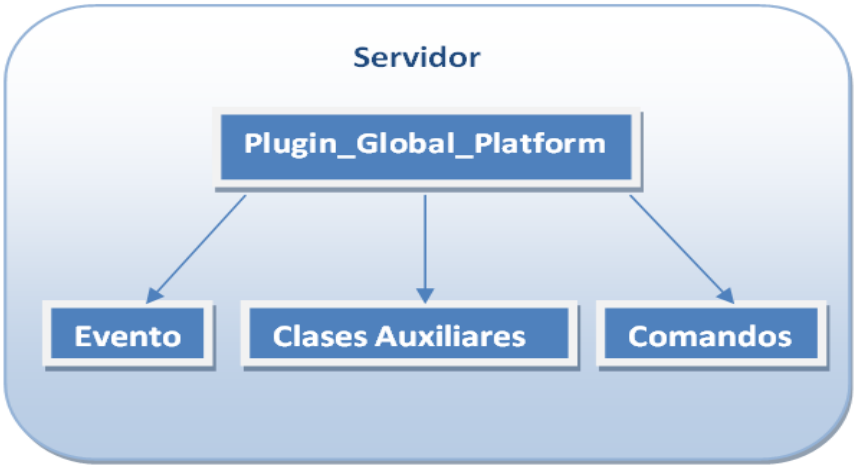

Figura 6: Arquitectura del servidor. Fuente: Elaboración propia.

2.8. Patrones de diseño utilizados

Durante el diseño de la aplicación *web* para administrar tarjetas inteligentes con *GlobalPlatform* se decidió hacer uso de patrones de *software* para la Asignación General de Responsabilidad (o patrones GRASP por sus siglas en Inglés), los mismos constituyen principios básicos a tener en cuenta cuando se quiere construir eficazmente un *software* orientado a objetos. Entre los más evidentes en el diseño propuesto se encuentran los patrones: controlador, alta cohesión y bajo acoplamiento. (30)

Controlador: es el intermediario entre la interfaz y el algoritmo que la implementa. En esta aplicación se evidencia el cumplimiento de este patrón en la clase *GlobalPlatformPlugIn*, la cual funciona de la forma antes descrita.

Alta cohesión: establece que la información almacenada en una clase debe estar relacionada con la misma. En la aplicación se pone de manifiesto este concepto con la clase *GlobalPlatformPlugIn*, ya que esta gestiona todos los eventos relacionados con el estándar *GlobalPlatform*; otro ejemplo del cumplimiento de este patrón es la clase CryptoUtil que posee solo algoritmos criptográficos.

Bajo Acoplamiento: cada clase se relaciona lo menos posible con el resto, tratando de que sea flexible a cambios. En la aplicación esto se manifiesta en la mayoría de los casos ya que las clases se relacionan entre sí solo lo necesario para en caso de que se realicen modificaciones en alguna de ellas las afectaciones a las demás sean mínimas.

2.9. Plan de entregas

Las historias de usuario servirán para crear el plan estimado de entrega. El plan de entregas se usará para crear los planes de iteración para cada iteración. Es en este momento cuando clientes y desarrolladores toman las decisiones comerciales y técnicas respectivamente. Con cada historia de usuario previamente evaluada en tiempo de desarrollo ideal, el cliente las agrupará en orden de importancia. De esta forma se puede trazar el plan de entregas en función de estos dos parámetros: tiempo de desarrollo ideal y grado de importancia para el cliente.

Este artefacto se elabora con la intención de fijar qué período de tiempo puede tardar la implementación de cada una de las historias, definiéndose las fechas en que serán liberadas las versiones funcionales del producto. (Ver anexo 4)

2.10. Diagrama de clases

El diagrama de clases captura la estructura lógica del sistema y las clases que constituyen el modelo. Es un modelo estático, describiendo lo que existe y qué atributos y comportamiento tiene, más que cómo se hace algo. Los diagramas de clases son los más útiles para ilustrar las relaciones entre las clases e interfaces. Las generalizaciones, las agregaciones y las asociaciones son todas valiosas para reflejar la herencia, la composición o el uso y las conexiones respectivamente. (31)

Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

A continuación se muestran los diagramas de clases realizados para llevar a cabo la aplicación. En este caso se dividieron por funciones, la figura 7 presenta todas las clases relacionadas con *GlobalPlatform* y las figuras 13 y 14 que se encuentran en el anexo 7 muestran las de Canal Seguro.

Diagrama de clases del *GlobalPlatform*

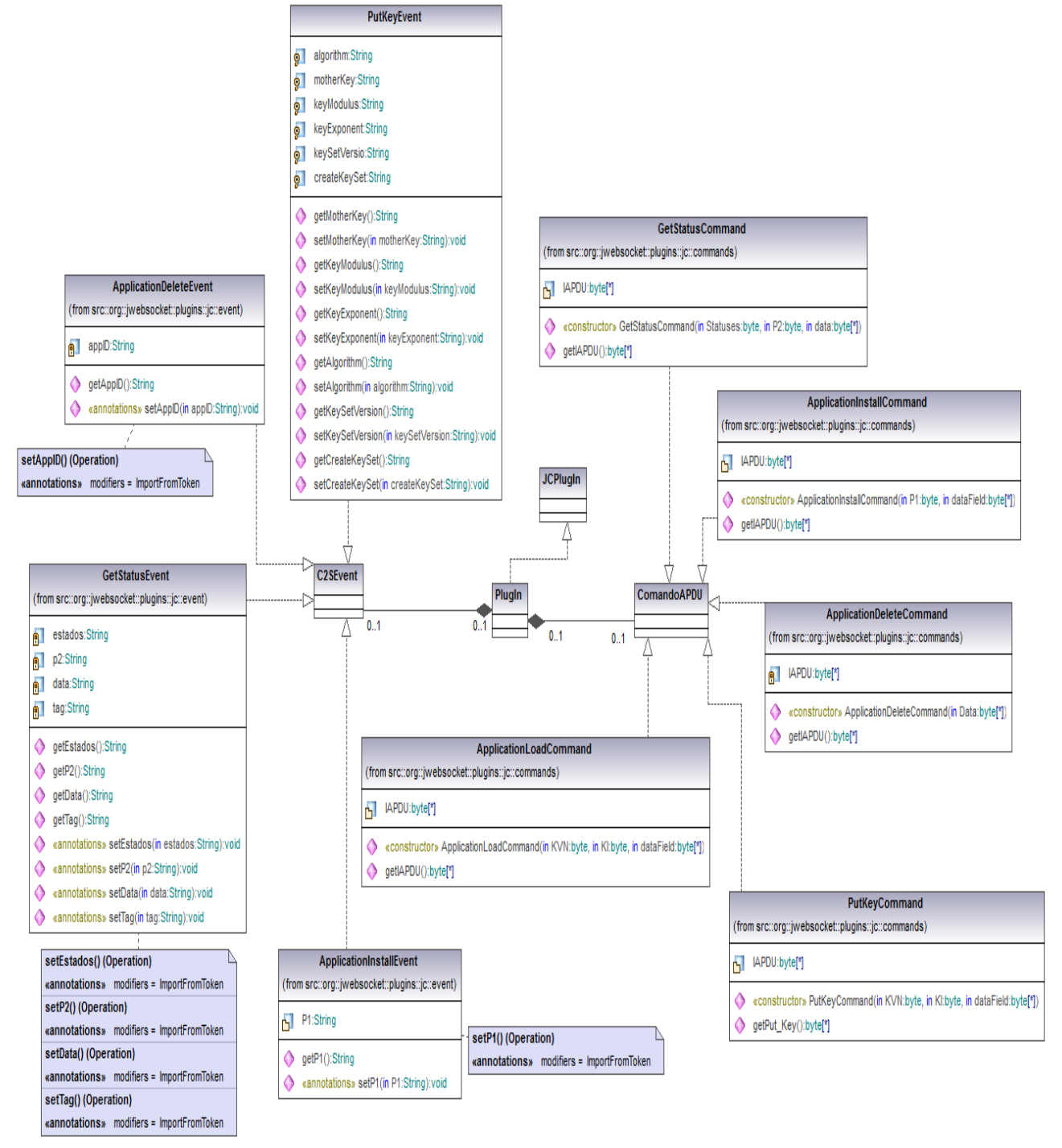

Figura 7: Diagrama de clases *GlobalPlatform*. Fuente: Elaboración propia.

Descripción de las clases

Dentro del diagrama de clases de *GlobalPlatform* se encuentra la clase ***SelectComand*** que será descrita a través de la tabla 3.

Nombre: *SelectComand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
mAppName	Byte[]
Para cada responsabilidad:	
Nombre:	SelectComand(byte[] aAppName)
Descripción:	Se inicializa un valor igualándolo al arreglo de byte pasado por parámetro.
Nombre:	getBytes()
Descripción:	Coloca en cada posición del arreglo el valor especificado por el estándar *GlobalPlatform*.
Nombre:	Getdata()
Descripción:	Devuelve el identificador del *applet* a seleccionar.

Tabla 3 - Descripción de la clase *SelectComand*

Dentro de las clases de Canal Seguro se encuentra **CanalSeguroPlugIn**, que será descrita en la tabla 4.

Nombre: *GlobalPlatform*PlugIn	
Tipo de clase: Controladora	
Atributo	**Tipo**
objeto	ClasUtil
mLog	Logger
nivelSeguridad;	Security
cstate	StateCS
apletseleccionado	boolean
con	Control
cont	int
comandoAPDU	byte []
FastMap<>	lresponse
utiles	ClasUtil

Para cada responsabilidad:	
Nombre:	processEvent(Canal_SeguroEvent aEvent, C2SResponseEvent aResponseEvent)
Descripción:	Establece el canal de comunicación seguro.
Nombre:	processEvent(GetStatusEvent aEvent, C2SResponseEvent aResponseEvent)
Descripción:	Obtiene el estado de los *applets* de la tarjeta.
Nombre:	processEvent(ApplicationDeleteEvent aEvent, C2SResponseEvent aResponseEvent)
Descripción:	Elimina un *applet* determinado de la tarjeta.
Nombre:	processEvent(ApplicationInstallEvent aEvent, C2SResponseEvent aResponseEvent)
Descripción:	Instala un *applet* en la tarjeta.
Nombre:	processEvent(PutKeyEvent aEvent, C2SResponseEvent aResponseEvent)
Descripción:	Adiciona una nueva llave a la tarjeta.

Tabla 4 - Descripción de la clase CanalSeguroPlugIn

El resto de las clases se encuentran en el Anexos.

2.11. Conclusiones

- En este capítulo se determinó el modelo de dominio que agrupa los conceptos asociados a la solución, así como las relaciones entre estos conceptos, que se determinaron con el estudio de las herramientas, tecnologías y elementos asociados al dominio de la solución.
- Se definieron las historias de usuarios que caracterizarán la solución, los requerimientos no funcionales y la arquitectura a tener en cuenta, así como los diagramas de clases pertinentes. Permitiendo sentar las bases para el desarrollo del diagrama de estados del *middleware*.

Capítulo 3: Implementación y pruebas

3.1. Introducción

En este capítulo se mostrará cómo fue implementada la aplicación, centrándose en los diagramas de componentes del cliente y del servidor, el diagrama de estados de la solución desarrollada y el diagrama de despliegue. Se realizará una validación y verificación del cumplimiento de los requerimientos planteados en el capítulo anterior con la aplicación de métodos de pruebas.

3.2. Diagrama de estados del *middleware*

El diagrama de estado se utiliza con el objetivo de modelar los diferentes estados por los que transita la aplicación durante su ciclo de vida, así como los comandos que disparan una transición o hacen que vaya de un estado a otro. Los estados marcados como persistentes (*Persistent_ State*) son los estados a los cuales regresa el *applet* en caso de que la tarjeta sea retirada del lector o el *applet* deje de estar seleccionado, regresando siempre al último estado persistente alcanzado.

El diagrama de estados del *middleware* se muestra en la figura 8.

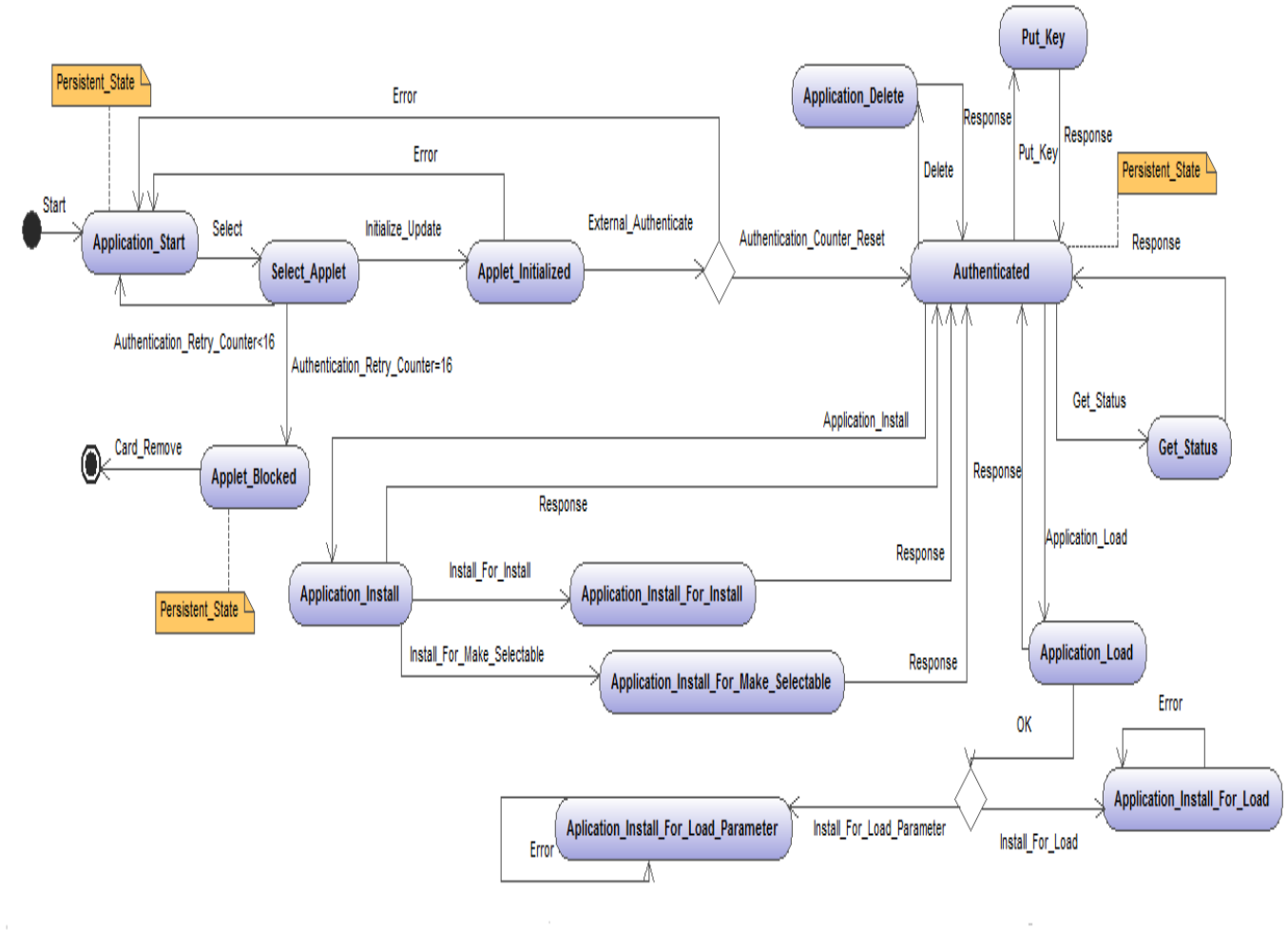

Figura 8: Diagrama de estados del *middleware*. Fuente: Elaboración propia.

A continuación se muestran los conceptos tratados en el diagrama de estados del *middleware* para facilitar su comprensión.

Application_Start: representa el estado inicial del *middleware*, ya que este funciona como una máquina de estados.

Select_Applet: es el segundo estado, el mismo permite al desarrollador seleccionar el *applet* de la tarjeta, si realiza esta operación de forma incorrecta menos de 16 veces tiene la posibilidad de volver al estado inicial para repetir la acción, pero si falla en 16 intentos pasa al estado *Applet_Blocked*. En caso que la selección haya sido satisfactoria pasa al estado *Applet_Initialized*.

Applet_Blocked: este estado representa el bloqueo de la aplicación. En caso de que la tarjeta pase a este estado, el desarrollador debe retirarla, pues ya esta se inhabilita.

Applet_Initialized: representa la inicialización del *applet*, la cual en caso de no efectuarse correctamente regresa al estado inicial, de lo contrario procede a realizar la autenticación externa.

Authenticated: este estado representa la autenticación de la tarjeta. En caso que la autenticación externa no se ejecute correctamente vuelve al estado inicial el *applet*, de lo contrario se reinicia el contador y se autentica de forma satisfactoria la tarjeta y se puede realizar cualquier operación requerida por el desarrollador.

Application_Delete: permite eliminar cualquier aplicación o paquete. Luego de realizada esta operación regresa al estado Authenticated.

Put_Key: permite adicionar o reemplazar una llave. Luego de realizada esta operación regresa al estado *Authenticated*.

Get_Status: permite obtener el estado de una aplicación. Luego de realizada esta operación regresa al estado *Authenticated*.

Application_Load: permite cargar una aplicación, lo cual se puede realizar de dos formas. Luego de realizada esta operación regresa al estado Authenticated.

Application_Install_For_Load: una de las formas de cargar aplicaciones se representa con este estado, el cual en caso de error, vuelve a intentar ejecutarse.

Application_Install_For_Load_Parameter: representa otra de las formas de cargar aplicaciones. En caso de error, vuelve a intentar ejecutarse.

Application_Install: permite instalar una aplicación, esta operación puede realizarse de dos formas. Luego de realizada esta operación regresa al estado *Authenticated*.

Application_Install_For_Install: representa una de las formas de instalar aplicaciones. Envía un mensaje de respuesta indicando si se instaló de forma correcta o incorrecta.

Application_Install_For_Make_Selectable: representa otra de las formas de instalar aplicaciones, en este caso especifica si es una aplicación previamente seleccionada. Envía un mensaje de respuesta indicando si se instaló de forma correcta o incorrecta.

3.3. Diagramas de componentes

Los diagramas de componentes describen los elementos físicos del sistema y sus relaciones. Muestran las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes representan todos los tipos de elementos de *software* que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente o de otro tipo. (32)

En la figura 9 se muestra el diagrama de componentes de la aplicación.

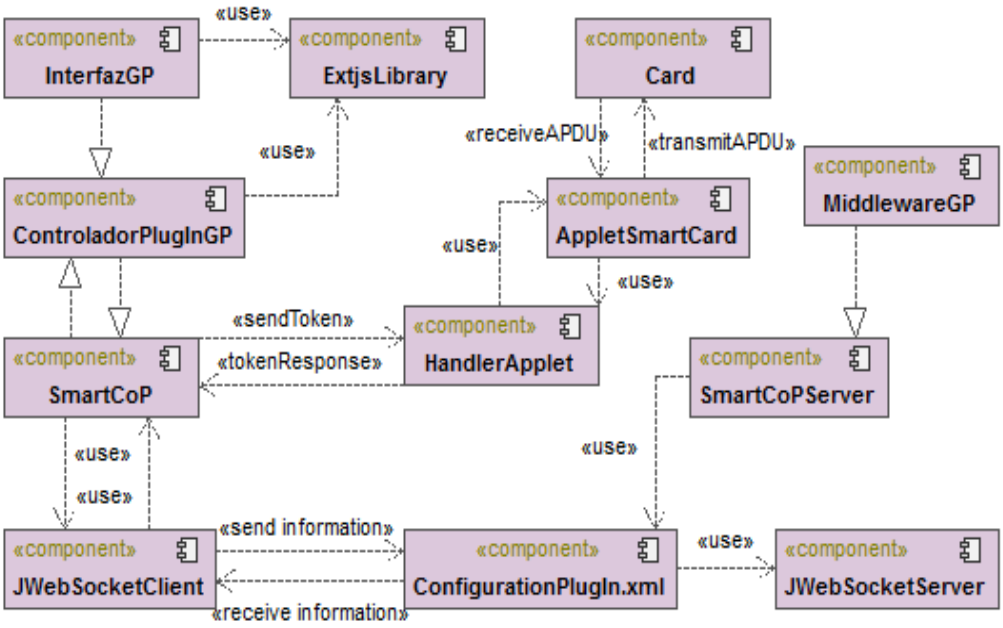

Figura 9: Diagrama de componentes. Fuente: Elaboración propia.

A continuación se muestran los conceptos tratados en el diagrama de componentes para facilitar su comprensión.

InterfazGP: interfaz de la aplicación desarrollada.

ControladorPlugInGP: plugin controlador de la aplicación desarrollada.

SmartCoP: interfaz de la plataforma *SmartCoP*

JWebSocketClient: cliente *JWebSocket*.

HandlerApplet: manejador de *applet*.

AppletSmartCard: *applet* de la tarjeta.

Card: tarjeta con la que se interactúa.

MiddlewareGP: *software* desarrollado del lado del servidor.

SmartCoPServer: servidor *SmartCoP*.

***ConfigurationPlugIn*.xml**: *plugin* de configuración.

JWebSocketServer: servidor *JWebSocket*.

3.4. Diagrama de despliegue

Un diagrama de despliegue muestra la disposición física de los distintos nodos que componen un sistema y se indican cuáles son los enlaces de comunicación existentes entre los distintos componentes en tiempo de ejecución. En la figura 10 se muestra el diagrama de despliegue de la aplicación.

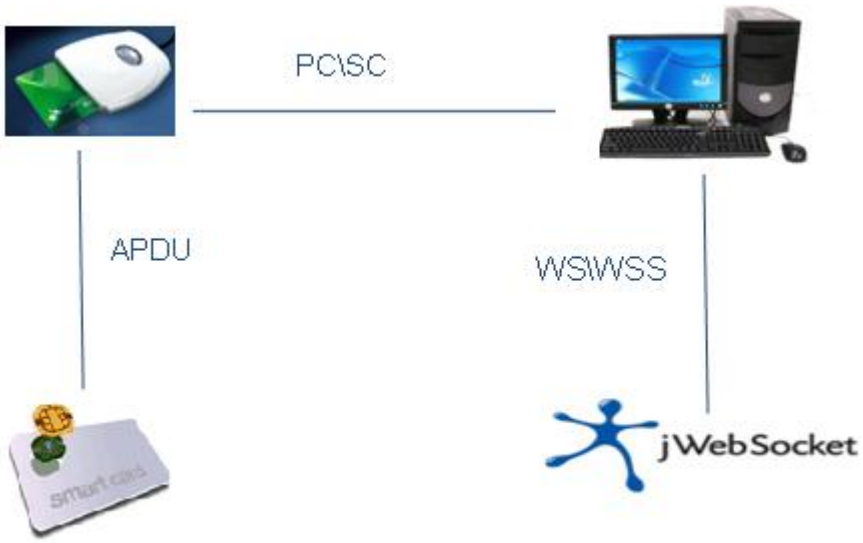

Figura 10: Diagrama de despliegue. Fuente: Elaboración propia.

A continuación se muestran los conceptos tratados en el diagrama de componentes para facilitar su comprensión.

Tarjeta inteligente: es una tarjeta plástica del tamaño de una tarjeta de crédito convencional, que contiene un pequeño microprocesador, que es capaz de hacer diferentes cálculos, guardar información y manejar programas.

Lector de tarjetas: dispositivo que se encarga de interpretar la información de la tarjeta y transmitirla hacia la máquina.

PC_Servidor: máquina en la que se gestiona y realizan todas las acciones de administración.

JWebSocket: *JWebSocket* es un marco de trabajo de código abierto para el desarrollo de aplicaciones *Web* estacionarias y móviles basado en *Java* en el lado del servidor y en *JavaScript* del lado del cliente.

3.5. Fase de producción

Para la fase de producción el equipo de desarrollo se centró en los pedidos del cliente, refinando el diseño y el código continuamente. El perfeccionamiento de código sólo fue posible a través de un grupo de pruebas automatizadas que aseguraron la ejecución correcta del sistema en todo el período de desarrollo. Frecuentemente cuando se desarrolla, siempre se realizan una gran cantidad de pruebas para verificar que el código esté correcto. Estas pruebas normalmente tienen que ser realizadas varias veces y se ven afectadas por los cambios que se introducen conforme se va desarrollando.

Con el objetivo de evaluar la calidad de la aplicación se realizó un proceso de pruebas para validar la implementación de las funcionalidades definidas, efectuando pruebas de caja blanca y pruebas de caja negra.

3.5.1. Pruebas de cajas blancas o estructurales

La puesta en práctica de este método requiere del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones del diseño o el código. Se basa en la comprobación de los caminos lógicos del *software* dado un código específico. (33)

Pasos para el diseño de pruebas utilizando el camino básico.

- Obtener el grafo de flujo, a partir del diseño o del código del módulo.
- Obtener la complejidad ciclomática del grafo de flujo. (*Application Delete*)
- Definir el conjunto básico de caminos independientes.
- Determinar los casos de prueba que permitan la ejecución de cada uno de los caminos anteriores.
- Ejecutar cada caso de prueba y comprobar que los resultados son los esperados.

Casos de prueba

De acuerdo al siguiente segmento de código correspondiente a la historia de usuario ***Application Delete***, se realiza la prueba de caja blanca.

```
public void processEvent(ApplicationDeleteEvent aEvent, C2SResponseEvent aResponseEvent) throws
Exception
    {
    String Datos = utiles.QuitarEspacios(aEvent.getAppID()); (1)
    String longitud = Integer.toHexString(Datos.length()/2); (1)
    String longitud1 = utiles.VerificarLongitud(longitud); (1)
   String data1 = utiles.ArmarDatosTLV("4F", longitud1, Datos); (1)
    byte[] data = utiles.hexStringToByteArray(data1); (1)
    String lClient = aEvent.getConnector().getId(); (1)
     int seguridad = con.getSecuridad().VALUE; (1)
    comandoAPDU = new ApplicationDeleteCommand(data, seguridad).getIAPDU(); (1)
    CommandAPDU APDUcomando = new CommandAPDU(comandoAPDU); (1)
    if (niverseguridad == niverseguridad.MAC || niverseguridad == niverseguridad.
    MACEncryption) (2) (3)
     {
       CommandAPDU apdu = new CommandAPDU(comandoAPDU); (4)
       APDUcomando = con.SendSecureAPDU(apdu); (4)
    }
     transmit(lClient, getTerminals(lClient).toArray()[0].toString(), APDUcomando, new
    JcResponseCallback(new TransactionContext(getEm(), aEvent, null)); (5)
```

}

La figura 11 muestra el grafo del flujo de prueba *Application Delete*.

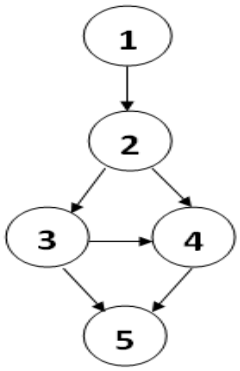

Figura 11: Grafo de flujo del caso de prueba *Application Delete*. Fuente: Elaboración propia.

Complejidad ciclomática

V (G): Número de regiones del grafo. **V (G)** = A – N + 2 **V (G)** = P + 1

A: Número de aristas del grafo. A= 6

N: Número de nodos. N= 5

P: Número de nodos predicados. P= 2

V (G) = 3

Caminos:

1, 2, 3, 5

1, 2, 4, 5

1, 2, 3, 4, 5

Casos de prueba para cada camino

Camino: 1, 2, 3, 5

Entrada: identificador del applet o paquete a borrar.

Salida: se lanza una excepción con mensaje de error en caso de fallo, en caso contrario se muestra ’90 00’ indicando la ejecución correcta del comando.

Precondiciones: haberse autenticado en este caso de forma *NoSecurity.*

Camino: 1, 2, 4, 5

Entrada: identificador del *applet* o paquete a borrar.

Salida: se lanza una excepción con mensaje de error en caso de fallo, en caso contrario se muestra’90 00’ indicando la ejecución correcta del comando.

Precondiciones: haberse autenticado en este caso de forma *MAC*.
Camino: 1, 2, 3, 4, 5
Entrada: identificador del *applet* o paquete a borrar.
Salida: se lanza una excepción con mensaje de error en caso de fallo, en caso contrario se muestra ’90 00’ indicando la ejecución correcta del comando.
Precondiciones: haberse autenticado en este caso de forma *MACEncryption*.
En el anexo 10 se muestra otro caso de prueba realizado.

3.5.2. Pruebas de cajas negras o funcionales

Las pruebas de caja negra, deben su nombre a los elementos que estas revisan y las condiciones en que se hace la revisión. Se basan en los requerimientos funcionales del sistema y se llevan a cabo desde el exterior de la aplicación. Este tipo de prueba es importante a la hora de medir el grado de cumplimiento de los requerimientos solicitados por el cliente y se aplican sobre la interfaz de la aplicación observando las respuestas del sistema ante determinadas acciones y los datos de salida para determinados datos de entrada. (34)

Las pruebas de caja negra pretenden encontrar los tipos de errores siguientes: (34)

- Funciones incorrectas o ausentes
- Errores en la interfaz
- Errores en estructuras de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Diseño de casos de pruebas de caja negra

La prueba verifica que el elemento que se está probando, cuando se dan las entradas apropiadas produce los resultados esperados. Los datos de prueba se escogerán atendiendo a las especificaciones del problema, sin importar los detalles internos del programa, a fin de verificar que el programa o la aplicación se ejecuten correctamente.

La tabla 5 muestra el caso de prueba de *Authentication* y la tabla 6 muestra el caso de prueba *Application Delete*.

Caso de prueba *Authentication*

Escenario	Descripción	CLA	INS	P1	P2	Lc	DATA	Le	Respuesta	Flujo central
SC1	Selecciona el *applet* de forma correcta.	00	A4	04	00	08	Clave del *Card Manager*.	-	El sistema muestra el APDU Respuesta '90''00', que indica que se ejecutó el comando de forma correcta.	Se selecciona el nivel de seguridad para la autenticación y se presiona el botón *Authenticate*. Se muestran en la consola el comando APDU enviado a la tarjeta y la respuesta '90''00'.
SC2	No se hace posible la selección del *applet*.	00	A4	04	00	08	Clave del *Card Manager*.	-	El sistema muestra un mensaje indicando que se ejecutó el comando de forma incorrecta.	Se selecciona el nivel de seguridad para la autenticación y se presiona el botón *Authenticate*. Se muestra un mensaje indicando un fallo en el proceso de selección.
SC3	Se inicializa el	80	50	00	00	08	Arreglo de		El sistema	Si SC1 se

	applet de forma correcta.						números aleatorios.	-	muestra el APDU Respuesta ‘90’’00’, que indica que se ejecutó el comando de forma correcta.	ejecuta de forma correcta y al presionar el botón *Authenticate* no ocurre ningún fallo, se muestran en la consola el comando APDU enviado a la tarjeta y la respuesta ‘90’’00’.
SC4	Se inicializa el *applet* de forma incorrecta.	80	50	00	00	08	Arreglo de números aleatorios.	-	El sistema muestra un mensaje indicando que se ejecutó el comando de forma incorrecta.	Si SC1 se ejecuta de forma correcta y al presionar el botón *Authenticate* ocurre un error, se muestra un mensaje indicando un fallo en el proceso.
SC5	Se autentica de forma correcta.	84	82	00	00	08	Arreglo de números aleatorios.	-	El sistema muestra el APDU Respuesta ‘90’’00’, que indica que se ejecutó el comando de	Si SC2 se ejecuta de forma correcta y al presionar el botón *Authenticate* no ocurre ningún fallo, se

									forma correcta.	muestran en la consola el comando APDU enviado a la tarjeta y la respuesta '90''00'.
SC6	Se autentica de forma incorrecta.	84	82	00	00	08	Arreglo de números aleatorios.	-	El sistema muestra un mensaje indicando que se ejecutó el comando de forma incorrecta.	Si SC1 se ejecuta de forma correcta y al presionar el botón *Authenticate* no ocurre ningún error, se muestra un mensaje indicando un fallo en el proceso.

Tabla 5 - Caso de prueba *Authentication*

Caso de prueba *Application Delete*

Escenario	Descripción	CLA	INS	P1	P2	Lc	DATA	Le	Respuesta	Flujo central
SC1	Elimina la aplicación de forma correcta.	80 u 84	E4	00	00	-	Contiene la llave del *applet* a eliminar.	-	El sistema muestra el APDU Respuesta '90''00', que indica que se ejecutó el comando de	Se introduce el identificador del *applet* o paquete a eliminar y luego se pulsa el botón *Delete*. Se muestra un mensaje en la

									forma correcta.	consola ‘90’’00’ indicando que se eliminó correctamente.
SC2	Elimina la aplicación de forma incorrecta.	80 u 84	E4	00	00	-	Contiene la llave del *applet* a eliminar.	-	El sistema muestra un mensaje indicando que se ejecutó el comando de forma incorrecta.	Se introduce el identificador del *applet* o paquete a eliminar y luego se pulsa el botón *Delete*. Se muestra un mensaje indicando un fallo en el proceso.

Tabla 6 - Caso de prueba *Application Delete*

Resultados de las pruebas

En las pruebas de caja negra realizadas a las funcionalidades que requieren entrada de datos se identificaron seis casos de prueba en total para las tres iteraciones con un número variable de escenarios para cada caso. En la primera iteración se realizó un caso de prueba detectándose tres no conformidades a las cuales se les dio solución. En la segunda iteración se realizaron cuatro casos de prueba detectándose seis no conformidades, las mismas fueron resueltas en su totalidad. En la tercera iteración se llevaron a cabo dos casos de prueba detectándose cinco no conformidades. En las tres iteraciones efectuadas se detectó un total de catorce no conformidades, las cuales en su mayoría respondían a errores de bajo impacto en el correcto funcionamiento de la aplicación y todas tuvieron solución en un tiempo máximo de cinco días. Posteriormente se realizó una iteración adicional para verificar que no existieran no conformidades, la misma arrojó un resultado de cero no conformidades, lo que indica que la aplicación *web* para administrar tarjetas inteligentes con *GlobalPlatform* 2.1.1 cumple con los requerimientos definidos.

La figura 12 muestra la gráfica representando las no conformidades que se le encontraron a los diferentes casos de prueba en cada una de las iteraciones desarrolladas. Se puede apreciar en el eje y el número de

no conformidades encontradas y en el eje x las iteraciones, a la derecha se observan los casos de prueba, en cada iteración se analizaron varios casos de prueba para detectar las no conformidades.

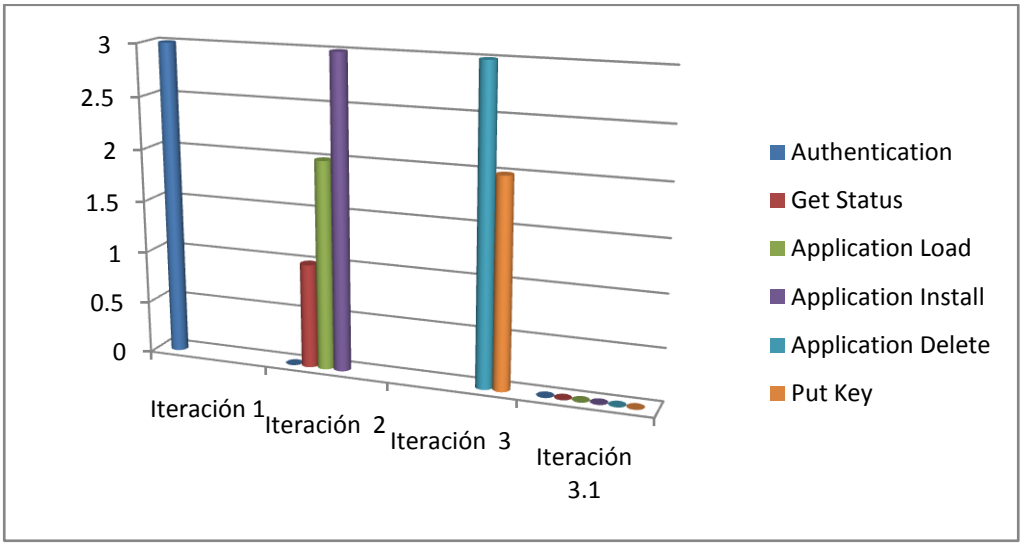

Figura 12: Resultado de las pruebas de Caja negra. Fuente: Elaboración propia.

3.6. Conclusiones

- En este capítulo se desarrolló el diagrama de estados para la aplicación que fue usado como guía para implementar las diferentes fases que atraviesa el *middleware* implementado.
- Se describieron los elementos físicos del sistema, sus relaciones y se indicaron los enlaces de comunicación existentes entre los distintos componentes en tiempo de ejecución con la elaboración de los diagrama de componentes y despliegue.
- Las pruebas realizadas a la aplicación validaron las funcionalidades descritas, comprobando que el sistema cumple con los requerimientos de seguridad definidos.

Conclusiones generales

Con el desarrollo de este trabajo se ha podido demostrar cómo se han cumplido los objetivos principales definidos para la creación de una aplicación *web* que permita la administración de tarjetas inteligentes, con la utilización del estándar *GlobalPlatform*. Algunas de las conclusiones arribadas luego de terminado este producto son:

- El análisis de las principales soluciones que utilizan tarjetas inteligentes en la *Web* posibilitó conocer el funcionamiento de las aplicaciones *web,* siendo de gran utilidad para la definición de los requisitos de la aplicación.
- El análisis de diferentes herramientas, tecnologías, lenguajes y metodologías ha permitido definir cuáles utilizar en el desarrollo del sistema.
- El desarrollo de la aplicación *web* para la administración de tarjetas inteligentes que implementen el estándar GlobalPlatform ha permitido aumentar los niveles de seguridad requeridos, ya que se utiliza como protocolo de comunicación WebSocket, permitiendo que el middleware se encuentre en el servidor.
- La ejecución de las pruebas ha posibilitado validar el correcto funcionamiento del sistema.

Recomendaciones

Al finalizar la presente investigación, se plantean las siguientes recomendaciones:

- Confeccionar una documentación detallada del *middleware* que se utilice como Manual de usuario para los futuros desarrolladores.
- Posibilitar en desarrollos posteriores que el *middleware* pueda soportar otros algoritmos de criptografía asimétrica para aumentar las posibilidades de uso en ambientes reales.

Bibliografía citada

1. Effing, Wolfgang and Rankl, Wolfgang. 2002. Smart Card Handbook Third Edition: John Wiley & SonsLtd. 2002.

2. GlobalPlatform. 2003. Card Specification Versión 2.1.1. 2003.

3. ERLICH, J. Especificación Formal de la Máquina Virtual Java Card Disponible en:. [Online].www.fing.edu.uy/.../informacion/.../documentacion_especificacionjavacard.doc.

4 .SOTOLONGO DAYRON ALMEIDA Solución para el control de acceso a la información de las entidades externas, en la cédula de identificación electrónica de la República Bolivariana de Venezuela. 2008.

5. ORTEGA, M. Implementación de Tarjeta Inteligente Java Card para el Control de Acceso a Instalaciones. Disponible en: http://www.eatis.org/eatis2010/portal/paper/memoria/html/files/sistemas/Maria%20Ortega.pdf.

6. VIÑOLO, K. P. y SANTANA, V. F. Plataforma para el desarrollo de servicios en línea utilizando tarjetas inteligentes. 2010.

7. Perovich, Daniel, Rodríguez, Leonardo and Martín, Varela.Proyecto de Taller V Programación de JavaCards.

8. PLATFORM, J. JavaTM 2 Platform Standard Ed. 5.0. Disponible en: http://download.oracle.com/. [Online].

9. http://sconnect.software.informer.com/. [Online].

10. Gemalto. 2007. Coesys Issuance Solutions for the Public Sector.

11. http://www.icard.net/web/icard/controlparental. [Online].

12.https://www.firmaprofesional.com/index.php/esp/ca/suport/certificats-arrel/114-ca-sin-categoria/263-certificats-ca-arrel-informacio-per-usuaris-finals. [Online].

13.http://www.macroseguridad.net/productos/smartcards/payment_card/especificaciones.php. [Online].

14. http://www.athena-scs.com/.../athena-smartcard. [Online].

15. http://www.rfidpoint.com/fundamentos/middleware/. [Online].

16. Castells, M. La galaxia Internet – Reflexiones sobre Internet, empresa y sociedad. Barcelona (Plaza & Janés). [Online]. 2006.

17. http://www.masadelante.com/faqs/que-significa-http. [Online].

18. Masó, Rolando Santamaría. EventsPlugIn Developer Guide,Reference Documentation Version 1.0.

19. http://www.slideshare.net/rtorres462003/metologa-agiles-desarrollo-software-xp-1709082. [Online].

20. http://www.cad.com.mx/historia_del_lenguaje_java.htm. [Online].

21. Paradigmas de la Programación: JavaScript y PythonAgosto 2010.

22. https://jwebsocket.org. [Online].

23. http://www.programacionweb.net/cursos/curso.php?num=2. [Online].

24. Rankl, Wolfgang y Effing, Wolfgang. Smart Card Handbook Third Edition. West Sussex : John Wiley & Sons Ltd, 2003.

25. http://www.altova.com/umodel.html. [Online].

26. http://netbeans.org/index_es.html. [Online].

27. http://www.gemalto.com/products/jcardmanager/. [Online].

28. Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. 2002. proyectos Ágiles. Qué es SCRUM | proyectos Ágiles. [Online].2002. http://www.proyectosagiles.org/que-es-scrum.

29. Sommerville, Ian. Ingeniería del software. Madrid : Pearson Educación, 2005.

30. PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad). [Online]. [Citado el: 11 de marzo de 2013.] http://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-

patrones-de-software-para-la-asignacion-general-de-responsabilidadparte-ii.

31. http//:www.dcc.uchile.cl/~psalinas/uml/modelo.html. [Online]

32.Modelo de implementación. Diagramas de componentes y Despliegue. [Online]. [Citado el: 25 de marzo de 2013.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

33. http://www.slideshare.net/rinconsete/pruebas-de-caja-blanca-y-negra. [Online]

34. http://www.globetesting.com/2012/08/pruebas-de-caja-negra/. [Online]

Bibliografía consultada

1. Effing, Wolfgang and Rankl, Wolfgang. *2002. Smart Card Handbook Third Edition: John Wiley & SonsLtd. 2002.*

2. *GlobalPlatform. 2003. Card Specification Versión 2.1.1. 2003.*

3. ERLICH, J. Especificación Formal de la Máquina Virtual Java Card Disponible en:. [Online]. www.fing.edu.uy/.../informacion/.../documentacion_especificacionjavacard.doc.

4. SOTOLONGO., DAYRON ALMEIDA. *Solución para el control de acceso a la información de las entidades externas, en la cédula de identificación electrónica de la República Bolivariana de Venezuela. 2008.*

5. ORTEGA, M. Implementación de Tarjeta Inteligente Java Card para el Control de Acceso a Instalaciones. Disponible en: http://www.eatis.org/eatis2010/portal/paper/memoria/html/files/sistemas/Maria%20Ortega.pdf.

6. VIÑOLO, K. P. y SANTANA, V. F. Plataforma para el desarrollo de servicios en línea utilizando tarjetas inteligentes. 2010.

7. Perovich, Daniel, Rodríguez, Leonardo and Martín, Varela. Proyecto de Taller V Programación de JavaCards.

8. PLATFORM, J. JavaTM 2 Platform Standard Ed. 5.0. Disponible en: http://download.oracle.com/. [Online].

9. http://sconnect.software.informer.com/. [Online].

10. Gemalto. 2007. Coesys Issuance Solutions for the Public Sector.

11. http://www.icard.net/web/icard/controlparental. [Online].

12.https://www.firmaprofesional.com/index.php/esp/ca/suport/certificats-arrel/114-ca-sin-categoria/263-certificats-ca-arrel-informacio-per-usuaris-finals. [Online].

13. http://www.macroseguridad.net/productos/smartcards/payment_card/especificaciones.php. [Online].

Bibliografía consultada

14. http://www.athena-scs.com/.../athena-smartcard. [Online].

15. http://www.rfidpoint.com/fundamentos/middleware/. [Online].

16. Castells, M. *La galaxia Internet – Reflexiones sobre Internet, empresa y sociedad. Barcelona (Plaza & Janés). [Online]. 2006.*

17. http://www.masadelante.com/faqs/que-significa-http. [Online].

18. Masó, Rolando Santamaría. *EventsPlugIn Developer Guide,Reference Documentation Version 1.0.*

19. http://www.slideshare.net/rtorres462003/metologa-agiles-desarrollo-software-xp-1709082. [Online].

20. http://www.cad.com.mx/historia_del_lenguaje_java.htm. [Online].

21. *Paradigmas de la Programación: JavaScript y PythonAgosto 2010.*

22. https://jwebsocket.org. [Online].

23. http://www.programacionweb.net/cursos/curso.php?num=2. [Online].

24. Rankl, Wolfgang y Effing, Wolfgang. Smart Card Handbook Third Edition. West Sussex : John Wiley & Sons Ltd, 2003.

25. http://www.altova.com/umodel.html. [Online].

26. http://netbeans.org/index_es.html. [Online].

27. http://www.gemalto.com/products/jcardmanager/. [Online].

28. Pressman, Roger S. Ingeniería del Software. Un enfoque práctico. 2002. proyectos Ágiles. Qué es SCRUM | proyectos Ágiles. [Online]. 2002. http://www.proyectosagiles.org/que-es-scrum..

29. Alliance, Smart Card. *Tarjetas Inteligentes y Sistemas de Identificación Seguros: Construyendo una Cadena de Confianza. .* 2006.

30. Garrett, Jesse James. *Ajax: A New Approach to Web Applications.* 2005.

31. *ISO Organization INTERNATIONAL STANDARD ISO/IEC 7816-4.* 2005.

32. Martínez, Ander. *NUnit. Manual de instalación en cliente.*

33. PC/SC Workgroup. *PC/SC Workgroup.*2010.

34. Sommerville, Ian. *Ingeniería del software.* Madrid : Pearson Educación, 2005.

35.PATRONES GRASP (Patrones de Software para la asignación General de Responsabilidad). [Online]. [Citado el: 11 de marzo de 2013.] http://jorgesaavedra.wordpress.com/2007/05/08/patrones-grasp-patrones-de-software-para-la-asignacion-general-de-responsabilidadparte-ii.

36.Modelo de implementación. Diagramas de componentes y Despliegue. [Online]. [Citado el: 25 de marzo de 2013.] http://www.dsi.uclm.es/asignaturas/42530/pdf/M2tema12.pdf.

37.http://www.slideshare.net/rinconsete/pruebas-de-caja-blanca-y-negra. [Online].

38.http://www.globetesting.com/2012/08/pruebas-de-caja-negra/. [Online].

Glosario de términos

AID (*Application Identifier*): identificador de la aplicación.

APDU: Unidad de Datos del Protocolo de Aplicación.

***Applet*:** aplicación que se ejecutan dentro de las tarjetas inteligentes y gestiona la información almacenada en ellas.

ISO (*International Organization for Standardization*): la Organización Internacional para la Normalización es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional.

Llave simétrica: consiste en que haya una contraseña con la cual sea posible codificar y decodificar los mensajes comunicados entre dos o más partes. Esta contraseña, aplicada a una función junto con el mensaje original, produce una salida tan difícil de descifrar como sea posible. Para descifrarla, se aplica de nuevo la misma función, obteniendo el mensaje original.

SW: palabra de estado definida por el estándar GlobalPlatform que muestra la respuesta a un determinado comando enviado a la tarjeta.

TLV *(Tag Length Value)***:** estructura definida por el estándar GlobalPlatform para determinados comandos, indicando para cada uno de ellos la operación a realizar (*Tag*), la longitud del campo de datos (*Length*) y finalmente el valor del campo de datos (*Value*) definidos en el comando.

Anexos

Anexo 1: Cronograma de trabajo

No.	Acciones a realizar	Responsable	Fecha de entrega
1	Fundamentación teórica de la investigación y estudio de la situación actual de la administración de tarjetas inteligentes en aplicaciones *web*.	Rita Milena Hernández Díaz Yasiel Conde Bernal	Diciembre 2012
2	Fundamentación de la metodología, estándares, herramientas y tecnologías a utilizar para el desarrollo de la solución propuesta.	Rita Milena Hernández Díaz	Diciembre 2012
3	Análisis del SmartCardFramework desarrollado en el Departamento de Tarjetas Inteligentes del CISED.	Yasiel Conde Bernal	Diciembre 2012
4	Realización del análisis y diseño de las extensiones (PlugIn) que actúan como eje central del *middleware* del lado del servidor *jWebSocket*.	Rita Milena Hernández Díaz	Febrero 2013
5	Realización del diseño de las interfaces correspondientes a las extensiones generadas en el servidor.	Rita Milena Hernández Díaz	Febrero 2013
6	Implementación de las extensiones del lado del servidor *JWebSocket*.	Yasiel Conde Bernal	Febrero 2013
7	Implementación de las interfaces correspondientes a las extensiones del lado del cliente.	Rita Milena Hernández Díaz	Abril 2013
8	Realización de las pruebas unitarias y de aceptación a la aplicación *web* para verificar el cumplimiento de los requisitos definidos.	Yasiel Conde Bernal	Abril 2013

9	Realización de pruebas de funcionamiento a la aplicación utilizando las tarjetas disponibles en el Departamento De Tarjetas Inteligentes del CISED.	Rita Milena Hernández Díaz	Abril 2013

Tabla 7 - Cronograma de trabajo

Anexo 2: Estructura del comando APDU

Estructura Comando APDU.		
- Campo	- Tamaño	- Descripción
- CLA	- 1 byte	- Clase de instrucción. Indica la estructura y el formato.
- INS	- 1 byte	- Código de instrucción. Especifica la instrucción del comando.
- P1	- 1 byte	- Parámetros de la instrucción. Proveen más información sobre la instrucción.
- P2	- 1 byte	- Número de bytes en el Data Field del APDU.
- Data Field	- LC bytes	- Secuencia de bytes con información.
- LE	- 1 byte	- Cantidad máxima de bytes esperados como respuesta.
- P1	- 1 byte	- Parámetros de la instrucción. Proveen más información sobre la instrucción.

Tabla 8 - Estructura de los comandos APDU

Anexo 3: Historias de usuario

Historia de Usuario	
Número: HU_3	**Nombre de Historia de Usuario:** *Application Load*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 2
Prioridad en negocio: Alta	**Puntos estimados:** 1.0
Riesgo en desarrollo: Alto	**Puntos reales:** 1.2
Descripción: Permite cargar una aplicación. Para ello se carga un *applet* (archivo .cap) del que se obtiene el identificador, se le introduce el identificador del *applet* de dominio de seguridad y se le pasan los parámetro que se quieran cargar permitiendo que se ejecute la aplicación.	
Observaciones: Responderá con una error en caso que: • Exista un espacio insuficiente de memoria	

Tabla 9 - HU *Application Load*

Historia de Usuario	
Número: HU_4	**Nombre de Historia de Usuario:** *Application Install*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 2
Prioridad en negocio: Alta	**Puntos estimados:** 1.1
Riesgo en desarrollo: Alto	**Puntos reales:** 1.2
Descripción: Permite instalar una aplicación, para eso se le pasa el identificador del paquete, del *applet* y de la instancia y luego se le pasan los parámetros específicos.	
Observaciones: Responderá con una error en caso que: • Los parámetros que se pasen en el campo de datos sean incorrectos. • Sea insuficiente el espacio de la memoria	

Tabla 10 - HU *Application Install*

Historia de Usuario	
Número: HU_4	**Nombre de Historia de Usuario:** *Application Delete*
Modificación de Historia de Usuario Número: Ninguna	

Usuario: Desarrollador	**Iteración asignada**: 3
Prioridad en negocio: Alta	**Puntos estimados:** 1.2
Riesgo en desarrollo: Alto	**Puntos reales:** 1.3
Descripción: Permite eliminar una aplicación. Para ello se le pasa el identificador del *applet* que se desea borrar y se elimina dicho *applet*.	
Observaciones: Responderá con una error en caso que: • El identificador del *applet* que se desea eliminar no coincida con el identificador de los *applets* de la tarjeta. • Se introduzca un caracter extraño o una letra que no pertenezca al lenguaje numérico hexadecimal.	

Tabla 11 - HU *Application Delete*

Historia de Usuario	
Número: HU_5	**Nombre de Historia de Usuario:** *Put Key*
Modificación de Historia de Usuario Número: Ninguna	
Usuario: Desarrollador	**Iteración asignada**: 3
Prioridad en negocio: Alta	**Puntos estimados:** 1.2
Riesgo en desarrollo: Alto	**Puntos reales:** 1.5
Descripción: Se utiliza para insertarle la llave al *applet*. Se pueden emplear dos tipos de llaves, las generadas por el algoritmo 3DES o por el RSA; en caso de que se seleccione el primero de ellos, es necesario introducir una llave madre para la creación de la misma y si se usa el segundo se debe poner una llave del módulo y una llave exponente.	
Observaciones: Responderá con una error en caso que: • No haya suficiente espacio en la tarjeta	

Tabla 12 - HU *Put Key*

Anexo 4: Plan de entregas

Entregable	Fin de iteración 1	Fin de iteración 2	Fin de iteración 3	Fin de iteración 3.1
Aplicación *web* para la administración de tarjetas inteligentes	Diciembre 2012	Marzo 2013	Abril 2013	Mayo2013

con *GlobalPlatform* 2.1.1				

Tabla 13 - Plan de entregas

Anexo 5: Estimación de tiempo

La tabla 14 muestra la estimación del tiempo realizada para cada funcionalidad.

Historia de usuario	Estimación (días)
Authentication	0.9
Get Status	1.2
Application Load	1
Application Install	0.9
Application Delete	1.5
Put Key	1

Tabla 14 - Estimación de tiempo

Anexo 6: Plan de iteraciones

La tabla 15 muestra el plan de iteraciones realizado.

Iteración	No.HU	Historia de usuario	Duración estimada (semanas)
1	HU_1	*Authentication*	6
2	HU_2	*Get Status*	8
	HU_3	*Application Load*	
	HU_4	*Application Install*	
3	HU_5	*Application Delete*	8
	HU_6	*Put Key*	

Tabla 15 - Plan de iteraciones

Anexo 7: Fragmentos 1 y 2 del diagrama de clases del Canal Seguro

Figura 13: Fragmento 1 del diagrama de clases del Canal Seguro. Fuente: Elaboración propia.


GlobalPlatformPlugin
(from src::org::jwebsocket::plugins::jc)
objeto=new Clasutil()
rnLog:Logger=Logging.getLogger()
rnAppName:byte[*]=new byte[][](byte) 0xA0, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x18,(byte) 0x43, (byte) 0x40, (byte) 0x00)
nivelseguridad:security
cstate
CSsesion
appletseleccionado:boolean=false
rnAppName1:byte[*]=new byte[][](byte) 0xA0, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x18, (byte) 0x50, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x52,
con:control=new control()
utiles=new Clasutil()
autenticate:boolean=false
processEvent(in aEvent:Canal_seguro, in aResponseEvent:C2SResponseEvent):void
setCSsesion(in CSsesion):void
processEvent(in aEvent, in aResponseEvent:C2SResponseEvent):void
processEvent(in aEvent, in aResponseEvent:C2SResponseEvent):void
ArmarDatos(in tag:String, in leng:String, in value:String):String
VerificarLongitud(in var:String):String
QuitarEspacios(in var:String):String
JCPlugIn
C2SEvent
-con
-nivelseguridad
control
(from src::org::jwebsocket::plugins::jc::canal_seguro)
seguridad:security=NoSecurity
keyDiversificationData:byte[*]
keyInformation:byte[*]
cardChallenge:byte[*]
cardCrytogram:byte[*]
hostCrytogram:byte[*]
hostChallenge:byte[*]=new byte[8]
util=new Clasutil()
ICV:byte[*]=new byte[][](0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00)
sessionENCKey:byte[*]
sessionMACKey:byte[*]
sessionDESKey:byte[*]
CMAC:byte[*]
GLL:Generarllaves=new Generarllaves()
keys:Map<T1->String,T2->byte[]>
kencther=new LLavemadre()
mAppName:byte[*]=new byte[][](byte) 0x00, (byte) 0x44, (byte) 0x04, (byte) 0x00, (byte) 0x10, (byte) 0xA0, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x18, (byte) 0x50, (byte) 0x00, (byte) 0x00, (
tf:TerminalFactory=TerminalFactory.getDefault()
ct:CardTerminals=tf.terminals()
l:List<T1->CardTerminal>=null
card:Card=null
c:CardTerminal=null
auc:String
keyVersionNumber:byte=0x00
keyIdentifier:byte=0x00
obje=new Clasutil()
enviarexternal(in seguro:security):CommandAPDU
computeauthenticate():void
GenerateHostChallenger():byte[*]
nivelDeseguridad(in nivel:String):security
EvaluacionRespuesta(in response:ResponseAPDU):void
Generalasesionkey():void
GetDerivationData():byte[*]
Formarcomadnoexternal():void
getKeyDiversificationData():byte[*]
setKeyDiversificationData(in keyDiversificationData:byte[*]):void
getKeyInformation():byte[*]
setKeyInformation(in keyInformation:byte[*]):void
getCardChallenge():byte[*]
setCardChallenge(in cardChallenge:byte[*]):void
getCardCrytogram():byte[*]
setCardCrytogram(in cardCrytogram:byte[*]):void
getHostChallenge():byte[*]
setHostChallenge(in hostChallenge:byte[*]):void
getHostCrytogram():byte[*]
getICV():byte[*]
getSessionMACKey():byte[*]
getSessionDESKey():byte[*]
getSessionENCKey():byte[*]
Canal_seguro
(from src::org::jwebsocket::plugins::jc::event)
nivelseguridad:String
appName:String
Property1:C2SEvent
getNivelseguridad():String
setNivelseguridad(in nivelseguridad:String):void
getAppName():String
setAppName(in appName:String):void
«enumeration»
security
(from src::org::jwebsocket::plugins::jc::canal_seguro)
NoSecurity = (byte) 0
MAC = (byte) 1
MACEncryption = (byte) 3
-seguridad

Figura 14: Fragmento 2 del diagrama de clases Canal Seguro. Fuente: Elaboración propia.

Anexo 8: Descripción de las clases

La tabla 16 describe la clase GetStatusComand de *GlobalPlatform*

Nombre: GetStatusComand	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	Byte[]
Para cada responsabilidad:	
Nombre:	GetStatusComand (byte Securitylevel, byte parameter, byte [] hostCryptogram)
Descripción:	Construye un comando APDU con las variables pasadas por parámetro y las instrucciones del estándar *GlobalPlatform*.
Nombre:	getlAPDU()
Descripción:	Devuelve el APDU creado.

Tabla 16 - Descripción de la clase GetStatusComand

La tabla 17 describe la clase PutKeyComand de *GlobalPlatform*

Nombre: PutKeyComand	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	Byte[]
Para cada responsabilidad:	
Nombre:	PutKeyComand (byte KVN, byte KI, byte [] dataField)
Descripción:	Construye un comando APDU con las variables pasadas por parámetro y las instrucciones del estándar *GlobalPlatform*.
Nombre:	getPut_Key()
Descripción:	Devuelve el APDU creado.

Tabla 17 - Descripción de la clase PutKeyComand

La tabla 18 describe la clase *ApplicationLoadComand* de *GlobalPlatform*

Nombre: *AplicationLoadComand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	Byte[]
Para cada responsabilidad:	
Nombre:	AplicationLoadComand (byte KVN, byte KI, byte [] dataField)
Descripción:	Construye un comando APDU con las variables pasadas por parámetro y las instrucciones del estándar *GlobalPlatform*.
Nombre:	getLoad ()
Descripción:	Devuelve el APDU creado.

Tabla 18 - Descripción de la clase *ApplicationLoadComand*

La tabla 19 describe la clase *ApplicationInstallComand* de *GlobalPlatform*

Nombre: *ApplicationInstallComand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	Byte[]
Para cada responsabilidad:	
Nombre:	ApplicationInstallCommand(byte P1, byte[] dataField)
Descripción:	Construye un comando APDU con las variables pasadas por parámetro y las instrucciones del estándar *GlobalPlatform*.
Nombre:	getlAPDU()
Descripción:	Devuelve el APDU creado.

Tabla 19 - Descripción de la clase *ApplicationInstallComand*

La tabla 20 describe la clase *ApplicationDeleteCommand* de *GlobalPlatform*

Nombre: *ApplicationDeleteCommand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	Byte[]
Para cada responsabilidad:	
Nombre:	ApplicationDeleteCommand(byte P2, byte[] Data)
Descripción:	Construye un comando APDU con las variables pasadas por parámetro y las instrucciones del estándar *GlobalPlatform*.
Nombre:	getlAPDU()
Descripción:	Devuelve el APDU creado.

Tabla 20 - Descripción de la clase *ApplicationDeleteCommand*

La tabla 21 describe la clase *ApplicationDeleteEvent* de *GlobalPlatform*

Nombre: *ApplicationDeleteEvent*	
Tipo de clase: Entidad	
Atributo	**Tipo**
appID	string
Para cada responsabilidad:	
Nombre:	getAppID ()
Descripción:	Devuelve el valor de la variable appID
Nombre:	setAppID(string appID)
Descripción:	Cambia el valor de la variable appID por el pasado por parámetro

Tabla 21 - Descripción de la clase *ApplicationDeleteEvent*

La tabla 22 describe la clase *LoadEvent* de *GlobalPlatform*

Nombre: LoadEvent

Tipo de clase: Entidad	
Atributo	**Tipo**
appID	string
Para cada responsabilidad:	
Nombre:	getAppID ()
Descripción:	Devuelve el valor de la variable appID

Tabla 22 - Descripción de la clase *LoadEvent*

La tabla 23 describe la clase *AplicationInstallEvent* de *GlobalPlatform*

Nombre: *AplicationInstallEvent*	
Tipo de clase: Entidad	
Atributo	**Tipo**
packageAID	String
*applet*AID	String
instanceAID	String
applicationSpecificParameters	String
volatileDataSpaceLimit	String
nonVolatileDataSpaceLimit	String
applicationPrivileges	String
makeSelectable	String
Para cada responsabilidad:	
Nombre:	getPackageAID()
Descripción:	Devuelve el valor de la variable packageAID.
Nombre:	setPackageAID(String packageAID)
Descripción	Cambia el valor de la variable packageAID por el pasado por parámetro.
Nombre:	get*Applet*AID ()
Descripción	Devuelve el valor de la variable *applet*AID.
Nombre:	set*Applet*AID (String *applet*AID)
Descripción	Cambia el valor de la variable *applet*AID por el pasado por parámetro.
Nombre:	getInstanceAID ()

Descripción	Devuelve el valor de la variable instanceAID.
Nombre:	setInstanceAID (String instanceAID)
Descripción	Cambia el valor de la variable instanceAID por el pasado por parámetro.
Nombre:	getApplicationSpecificParameters ()
Descripción	Devuelve el valor de la variable applicationSpecificParameters.
Nombre:	setApplicationSpecificParameters (String applicationSpecificParameters)
Descripción	Cambia el valor de la variable applicationSpecificParameters por el pasado por parámetro.
Nombre:	getVolatileDataSpaceLimit ()
Descripción	Devuelve el valor de la variable volatileDataSpaceLimit.
Nombre:	setVolatileDataSpaceLimit (String volatileDataSpaceLimit)
Descripción	Cambia el valor de la variable volatileDataSpaceLimit por el pasado por parámetro.
Nombre:	getNonVolatileDataSpaceLimit ()
Descripción	Devuelve el valor de la variable nonVolatileDataSpaceLimit
Nombre:	setNonVolatileDataSpaceLimit (String nonVolatileDataSpaceLimit)
Descripción	Cambia el valor de la variable nonVolatileDataSpaceLimit por el pasado por parámetro.
Nombre:	getApplicationPrivileges ()
Descripción	Devuelve el valor de la variable applicationPrivileges.
Nombre:	setApplicationPrivileges (String applicationPrivileges)

Descripción	Cambia el valor de la variable applicationPrivileges por el pasado por parámetro.
Nombre:	getMakeSelectable ()
Descripción	Devuelve el valor de la variable makeSelectable.
Nombre:	setMakeSelectable (String makeSelectable)
Descripción	Cambia el valor de la variable makeSelectable por el pasado por parámetro.

Tabla 23 - Descripción de la clase *AplicationInstallEvent*

La tabla 24 describe la clase *PutKeyEvent* de *GlobalPlatform*

Nombre: *PutKeyEvent*	
Tipo de clase: Entidad	
Atributo	**Tipo**
algorithm	String
motherKey	String
keyModulus	String
keyExponent	String
keySetVersion	String
createKeySet	String
Para cada responsabilidad:	
Nombre:	getMotherKey()
Descripción:	Devuelve el valor de la variable motherKey.
Nombre:	setMotherKey(String motherKey)
Descripción:	Cambia el valor de la variable motherKey por el pasado por parámetro.
Nombre:	getAlgorithm ()
Descripción:	Devuelve el valor de la variable algorithm.
Nombre:	setAlgorithm (String algorithm)
Descripción:	Cambia el valor de la variable algorithm por el pasado por parámetro.
Nombre:	getKeyModulus ()

Descripción:	Devuelve el valor de la variable keyModulus.
Nombre:	setKeyModulus (String keyModulus)
Descripción:	Cambia el valor de la variable keyModulus por el pasado por parámetro.
Nombre:	getKeyExponent ()
Descripción:	Devuelve el valor de la variable keyExponent.
Nombre:	setKeyExponent (String keyExponent)
Descripción:	Cambia el valor de la variable keyExponent por el pasado por parámetro.
Nombre:	getKeySetVersion ()
Descripción:	Devuelve el valor de la variable keySetVersion.
Nombre:	setKeySetVersion (String keySetVersion)
Descripción:	Cambia el valor de la variable keySetVersion por el pasado por parámetro.
Nombre:	getCreateKeySet ()
Descripción:	Devuelve el valor de la variable createKeySet.
Nombre:	setCreateKeySet (String createKeySet)
Descripción:	Cambia el valor de la variable createKeySet por el pasado por parámetro.

Tabla 24 - Descripción de la clase *PutKeyEvent*

La tabla 25 describe la clase *GetStatusEvent* de *GlobalPlatform*

Nombre: *GetStatusEvent*	
Tipo de clase: Entidad	
Atributo	**Tipo**
estados	String
P2	String
data	String
tag	String
Para cada responsabilidad:	
Nombre:	getEstados()
Descripción:	Obtiene el valor de la variable estado.
Nombre:	setEstados (String estado)

Descripción:	Cambia el valor de la variable estados por la pasada por parámetros.
Nombre:	getP2()
Descripción:	Obtiene el valor de la variable p2
Nombre:	setP2(String p2)
Descripción:	Cambia el valor de la variable p2 por la pasada por parámetros.
Nombre:	getData()
Descripción:	Obtiene el valor de la variable data
Nombre:	setData(String data)
Descripción:	Cambia el valor de la variable data por la pasada por parámetros.
Nombre:	getTag()
Descripción:	Obtiene el valor de la variable tag
Nombre:	setTag(String tag)
Descripción:	Cambia el valor de la variable tag por la pasada por parámetros.

Tabla 25 - Descripción de la clase *GetStatusEvent*

La tabla 26 describe la clase *AuthPlugIn* de Canal Seguro

Nombre: *AuthPlugIn*	
Tipo de clase: Controladora	
Atributo	**Tipo**
mLog	Logger
mState	StateMachine
mAppName	byte[]
nivelSeguridad	Security
cstate	StateCS
con	Control
Para cada responsabilidad:	
Nombre:	processEvent(Canal_SeguroEvent aEvent, C2SResponseEvent aResponseEvent)

Descripción:	Permite abrir la comunicación entre el *applet* y la tarjeta.

Tabla 26 - Descripción de la clase *AuthPlugIn*

La tabla 27 describe la clase *StateCS* de Canal Seguro

Nombre: *StateCS*
Tipo de clase: Enumerativo
Descripción
Enumerativo que guarda los siguientes estados: *READY,APP_SELECTED,INICIALIZE y AUTHENTICATE*

Tabla 27 - Descripción de la clase *StateCS*

La tabla 28 describe la clase *StateMachine* de Canal Seguro

Nombre: *StateMachine*
Tipo de clase: Enumerativo
Descripción
Enumerativo que devuelve el valor de la máquina de estado que puede ser (*READY, AUTHENTICATED, APP_SELECTED,SITE_SELECTED,GET_USER,GET_PASSWORD*).

Tabla 28 - Descripción de la clase *StateMachine*

La tabla 29 describe la clase *InitializeUpdateCommand* de Canal Seguro

Nombre: *InitializeUpdateCommand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
mLog	byte[]
Para cada responsabilidad:	
Nombre:	InitializeUpdateCommand(byte keyVersionNumber, byte keyIdentifier, byte[] hostChallenge)
Descripción:	Construye un objeto de la clase

	InitializeUpdateCommand
Nombre:	getIAPDU()
Descripción:	Devuelve el valor de la variable lAPDU.

Tabla 29 - Descripción de la clase *InitializeUpdateCommand*

La tabla 30 describe la clase *SelectCommand* de Canal Seguro

Nombre: *SelectCommand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
mAppName	byte[]
Para cada responsabilidad:	
Nombre:	SelectCommand (byte[] aAppName)
Descripción:	Inicializa el valor de la variable mAppName con el valor de la variable pasada por parámetro.
Nombre:	getBytes()
Descripción:	Devuelve el valor de la variable bytes.
Nombre:	getData()
Descripción:	Construye un arreglo de tipo byte y lo inicializa con los valores establecidos por el *GlobalPlatform* para el comando *Select*.

Tabla 30 - Descripción de la clase *SelectCommand*

La tabla 31 describe la clase *ExternalAuthenticateCommand* de Canal Seguro

Nombre: *ExternalAuthenticateCommand*	
Tipo de clase: Entidad	
Atributo	**Tipo**
lAPDU	byte[]
Para cada responsabilidad:	
Nombre:	ExternalAuthenticateCommand (byte securityLevel, byte[] hostCryptogram)
Descripción:	Inicializa un arreglo de tipo byte de la clase

	ExternalAuthenticateCommand y lo inicializa con los valores establecidos por el *GlobalPlatform* para el comando ExternalAuthenticate.
Nombre:	getlAPDU()
Descripción:	Devuelve el valor de la variable lAPDU.

Tabla 31 - Descripción de la clase *ExternalAuthenticateCommand*

La tabla 32 describe la clase Canal_SeguroEvent de Canal Seguro

Nombre: Canal_SeguroEvent	
Tipo de clase: Entidad	
Atributo	**Tipo**
nivelseguridad	String
appName	String
Para cada responsabilidad:	
Nombre:	getNivelseguridad()
Descripción:	Devuelve el valor de la variable Nivelseguridad
Nombre:	setNivelseguridad(String nivelseguridad)
Descripción:	Cambia el valor de la variable nivelseguridad de la clase por la pasada por parámetro.
Nombre:	getAppName
Descripción:	Devuelve el valor de la variable AppName.
Nombre:	setAppName(String appName)
Descripción:	Cambia el valor de la variable appName de la clase por la pasada por parámetro.

Tabla 32 - Descripción de la clase Canal_SeguroEvent

La tabla 33 describe la clase ClassUtil de Canal Seguro

Nombre: ClasUtil	
Tipo de clase: Entidad	
Atributo	**Tipo**
cardCryptogram	byte[]

hostCrytogram	byte[]
CMAC	byte[]
Para cada responsabilidad:	
Nombre:	check9000(ResponseAPDU ra)
Descripción:	Comprueba si la respuesta del comando APDU pasado por parámetro es la correcta.
Nombre:	ComputeAuthenticationCryptograms(byte[] *card*Challenge, byte[] hostChallenge, byte[] ENCKey)
Descripción:	Encripta el comando con nivel de seguridad MAC.
Nombre:	get*Card*Cryptogram()
Descripción:	Devuelve el valor de la variable CardCryptogram
Nombre:	getHostCrytogram()
Descripción:	Devuelve el valor de la variable HostCrytogram.
Nombre:	getCMAC()
Descripción:	Devuelve el valor de la variable CMAC
Nombre:	hexStringToByteArray(String s)
Descripción:	Convierte una variable string hexadecimal a byte.
Nombre:	hacerpalabra(byte[] respuesta)
Descripción:	Convierte el arreglo de byte pasado por parámetro a string.
Nombre:	arrayToHex(byte[] data)
Descripción:	Convierte el arreglo de byte pasado por parámetro a hexadecimal.
Nombre:	PonerEspacios(String var)
Descripción:	Coloca los espacios en caso de ser necesario.
Nombre:	QuitarEspacios(String var)
Descripción:	Elimina los espacios que aparezcan que sean innecesarios.
Nombre:	ArmarDataInstall(String pAID, String aAID, String iAID, String appPrivileges)
Descripción:	Arma el DataInstall utilizando los valores pasados por parámetro.
Nombre:	ArmarParametersField(String VDSL, String NVDSL,

	String appSpecificParameters)
Descripción:	Arma el ParametersField utilizando los valores pasados por parámetro.
Nombre:	TLV(String vdsl, String nvdsl)
Descripción:	Devuelve el valor del TLV.
Nombre:	RellenarCero(String var, int lon)
Descripción:	Rellena los ceros que aparezcan.
Nombre:	ArmarDatosTLV(String tag, String leng, String value)
Descripción:	Arma los datos TLV a partir de los valores pasados por parámetro.
Nombre:	VerificarLongitud(String var)
Descripción:	Verifica la longitud del valor pasado por parámetro.

Tabla 33 - Descripción de la clase ClassUtil

La tabla 34 describe la clase Sesión de Canal Seguro

Nombre: Sesión	
Tipo de clase: Entidad	
Atributo	**Tipo**
securitylevel	Security
csAuthenticate	boolean
sessionENCKeys	byte[]
sessionMACKeys	byte[]
sessionDEKKeys	byte[]
ICV	byte[]
sessioncontrol	Control
functionUtil	ClaseUtil
Para cada responsabilidad:	
Nombre:	Sesión(Security securitylevel, byte [] SessionENCKey,byte[]SessionMACKey)
Descripción:	Se inicializan los atributos de la clase.
Nombre:	getSecuritylevel()
Descripción:	Obtiene el nivel de seguridad del canal de

	comunicación.
Nombre:	getSessionENCKeys()
Descripción:	Devuelve el valor de la variable SessionENCKeys
Nombre:	getSessionMACKeys()
Descripción:	Devuelve el valor de la variable SessionMACKeys
Nombre:	getSessionDEKKeys()
Descripción:	Devuelve el valor de la variable SessionDEKKeys.
Nombre:	SendSecureAPDU(CommandAPDU command)
Descripción:	Envía el comando pasado por parámetro de forma segura a la tarjeta.

Tabla 34 - Descripción de la clase Sesión

La tabla 35 describe la clase Security de Canal Seguro

Nombre: Security
Tipo de clase: Enumerativo
Descripción
Clase que define el tipo de seguridad de la comunicación con la tarjeta que puede ser (NoSecurity,MAC,MACEncryption)

Tabla 35 - Descripción de la clase Security

La tabla 36 describe la clase CryptoUtil de Canal Seguro

Nombre: CryptoUtil	
Tipo de clase: Entidad	
Atributo	**Tipo**
uti	ClaseUtil
Para cada responsabilidad:	
Nombre:	GenerateRandomData(int dataLength)
Descripción:	Devuelve un número aleatorio del tamaño de la variable pasado por parámetro.
Nombre:	SymmetricCipher(String cipherAlgorithmConfig, byte[] data, byte[] key, byte[] icv, boolean forEncription)
Descripción:	Este método realiza un algoritmo de cifrado simétrico que devuelve un arreglo con la información encriptada.
Nombre:	InvertKey(byte[] key)
Descripción:	Este método copia para un arreglo auxiliar desde la

	posición cero hasta los últimos 8 bits del arreglo pasado por parámetro.
Nombre:	TripleDES16(byte[] key, byte[] data)
Descripción:	Este método realiza un algoritmo de cifrado, se le pasa información y una llave y la encripta con esa llave.
Nombre:	FullTripleDESMAC(byte[] data, byte[] key, byte[] icv)
Descripción:	Este método realiza un algoritmo de cifrado.
Nombre:	GenerateKeyPartsFromKey(byte[] sourceKey, int keyPartsCount)
Descripción:	Genera nuevas particiones para la llave.
Nombre:	XORByteArrays(byte[] array1, byte[] array2)
Descripción:	Unifica los arreglos pasados por parámetros.
Nombre:	CreateKeyFromKeyParts(ArrayList keyPartsList)
Descripción:	Crea nuevas aplicaciones en las llaves.
Nombre:	ComputeKeyCheckValue(byte[] key)
Descripción:	Chequea el valor de la llave.
Nombre:	Concat(byte[]... arrays)
Descripción:	Concatena los valores del arreglo de byte pasado por parámetro.

Tabla 36 - Descripción de la clase CryptoUtil

La tabla 37 describe la clase GenerarLlaves de Canal Seguro

Nombre: GenerarLlaves	
Tipo de clase: Entidad	
Atributo	**Tipo**
uti	ClaseUtil
keys	Map<String,byte[]>
Para cada responsabilidad:	
Nombre:	getStatickeys(byte [] keyDiversificationData, byte [] motherKey)
Descripción:	Algoritmo que genera las llaves estáticas del dominio de seguridad a partir de los datos de diversificación contenido en la respuesta del comando InitializeUpdate.

Tabla 37 - Descripción de la clase GenerarLlaves

La tabla 38 describe la clase LlaveMadre de Canal Seguro

Nombre: LlaveMadre	
Tipo de clase: Entidad	
Atributo	**Tipo**
b	ClaseUtil
llavemadre	byte[]
Para cada responsabilidad:	
Nombre:	getLlavemadre()
Descripción:	Devuelve el valor de una llave madre estática contenida en la clase.

Tabla 38 - Descripción de la clase LlaveMadre

La tabla 39 describe la clase Control de Canal Seguro

Nombre: Control	
Tipo de clase: Entidad	
Atributo	**Tipo**
seguridad	Security
keyDiversificationData	byte[]
keyInformation	byte[]
cardChallenge	byte[]
cardCryptogram	byte[]
hostCrytogram	byte[]
hostChallenge	byte[]
util	ClaseUtil
ICV	byte[]
sessionENCKey	byte[]
sessionMACKey	byte[]
sessionDESKey	byte[]
CMAC	byte[]
GLL	GenerarLlaves
keys	Map<String, byte[]>
kemother	LLaveMadre
mAppName	byte[]

tf	TerminalFactory
ct	CardTerminals
l	List<CardTerminal>
card	Card
c	CardTerminal
keyVersionNumber	byte
keyIdentifier	byte
obje	ClaseUtil
Para cada responsabilidad:	
Nombre:	CommandAPDU enviarexternal(Security seguro)
Descripción:	Comprueba el nivel de seguridad y utiliza el comando *EXTERNAL AUTHENTICATE* para autenticarse con la tarjeta.
Nombre:	computeautenticate()
Descripción:	Compara los criptogramas de la tarjeta para en caso de que sean iguales realizar la autenticación correctamente.
Nombre:	GenerateHostChalenger()
Descripción:	Crea un número aleatorio de tipo byte para identificar el origen de las conexiones.
Nombre:	niveldeseguridad(String nivel)
Descripción:	Asigna los niveles de seguridad de la tarjeta de acuerdo a los valores del enumerador Security.
Nombre:	EvaluacionRespuesta(ResponseAPDU response)
Descripción:	Obtiene el keyDiversificationData, el kitInformation,el CardChallenge y el cardCryptogram del atributo de tipo ResponseAPDU pasado por parámetro.
Nombre:	Generatesesionkey()
Descripción:	Genera las llaves del encriptado.
Nombre:	GetDerivationData()
Descripción:	Copia para el arreglo DerivationData el cardChallenge y el hostChallenge.
Nombre:	getKeyDiversificationData()

Descripción:	Devuelve el valor de la variable keyDiversificationData.
Nombre:	setKeyDiversificationData(byte[] keyDiversificationData)
Descripción:	Cambia el valor de la variable keyDiversificationData por el pasado por parámetro.
Nombre:	getKeyInformation()
Descripción:	Devuelve el valor de la variable keyInformation.
Nombre:	setKeyInformation(byte[] keyInformation)
Descripción:	Cambia el valor de la variable keyInformation por el pasado por parámetro.
Nombre:	getCardChallenge()
Descripción:	Devuelve el valor de la variable CardChallenge.
Nombre:	setCardChallenge(byte[] cardChallenge)
Descripción:	Cambia el valor de la variable cardChallenge por el pasado por parámetro.
Nombre:	getCardCryptogram()
Descripción:	Devuelve el valor de la variable CardCryptogram.
Nombre:	setCardCryptogram(byte[] cardCryptogram)
Descripción:	Cambia el valor de la variable cardCryptogram por el pasado por parámetro.
Nombre:	getHostChallenge()
Descripción:	Devuelve el valor de la variable HostChallenge
Nombre:	setHostChallenge(byte[] hostChallenge)
Descripción:	Cambia el valor de la variable hostChallenge por el pasado por parámetro.
Nombre:	getHostCrytogram()
Descripción:	Devuelve el valor de la variable HostCrytogram
Nombre:	getICV()
Descripción:	Devuelve el valor de la variable ICV
Nombre:	getSessionMACKey()
Descripción:	Devuelve el valor de la variable sessionMACKey
Nombre:	getSessionDESKey()
Descripción:	Devuelve el valor de la variable sessionDESKey
Nombre:	getSessionENCKey()

Descripción:	Devuelve el valor de la variable sessionENCKey

Tabla 39 - Descripción de la clase Control

Anexo 9: Descripción de comandos del *GlobalPlatform* usados.

El *middleware* para la administración de tarjetas inteligentes soporta los comandos que se muestran a continuación en la tabla 40, los cuales están definidos en el estándar *GlobalPlatform* 2.1.1.

Comando	Descripción
INITIALIZE UPDATE	Se utiliza para transmitir información de la tarjeta y la sesión de datos de canal seguro entre la tarjeta y el anfitrión.
EXTERNAL AUTHENTICATE	Comando que permite a la tarjeta autenticarse con el terminal.
SELECT	Se usa para seleccionar una aplicación determinada.
PUT KEY	Se puede emplear para: • Reemplazar una clave existente con una nueva clave. • Reemplazar múltiples claves existentes con las nuevas llaves •Agregar una nueva clave única. • Añadir múltiples claves nuevas.
LOAD	Se utiliza para cargar archivos.
INSTALL	Este comando es asignado al Dominio de seguridad para iniciar o llevar a cabo varios pasos requeridos por el Administrador de contenido de la tarjeta.
GET STATUS	Este comando se usa para obtener el estado del dominio de seguridad del distribuidor, archivo ejecutable de carga, el módulo ejecutable, la aplicación o dominio de seguridad del ciclo de vida de acuerdo a un criterio de búsqueda dado.

Tabla 40 - Comandos de *GlobalPlatform* utilizados

Descripción del *SELECT* command

El comando *SELECT* se utiliza para seleccionar una aplicación determinada. Se codifica de acuerdo con la tabla 41.

Código	Valor	Significado
CLA	'00'	ISO/IEC 7816-4 comando
INS	'A4'	*SELECT*
P1	'xx'	Parámetro de control de referencia P1
P2	'xx'	Parámetro de control de referencia P2
Lc	'xx'	Longitud de la AID
Data	'xxxx…'	AID de la aplicación seleccionada
Le	'00'	

Tabla 41 - Codificación del comando *SELECT*

Parámetro de control de referencia P1

El parámetro de control de referencia P1 se codificará de acuerdo con la tabla 42.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
					1			Selección por el nombre

Tabla 42 - Codificación del parámetro P1 cuando se usa el *SELECT*

Parámetro de control de referencia P2

El parámetro de control de referencia P2 se codificará de acuerdo con la tabla 43.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
						0	0	Primera o única ocurrencia
						1	0	Próxima ocurrencia

Tabla 43 - Codificación del parámetro P2 cuando se usa el *SELECT*

Campo de datos enviados en el mensaje de comando

El campo de datos del comando deberá contener el AID de la aplicación que desea seleccionar. La longitud de los datos (Lc) y el campo de datos del comando *SELECT* pueden ser omitidos si el dominio de seguridad emisor está siendo seleccionado. En este caso, le será tendrá el valor '00 según la norma ISO / IEC 7816-4.

Estructura del mensaje de respuesta

Campo de datos devueltos en el mensaje de respuesta

La respuesta del campo de datos *SELECT* consta de información específica de la aplicación seleccionada. La codificación de la información de control de archivos para la seguridad del dominio emisor y dominios de seguridad será de acuerdo con la tabla 44.

Etiqueta	Descripción	Presencia
'6F '	Archivo de información de control (plantilla FCI)	Obligatorio
'84'	Aplicación / archivo AID	Obligatorio
'A5'	Datos de propiedad	Obligatorio
'73'	Dominio de Seguridad de Datos de Gestión	Opcional
'9F6E'	Aplicación del ciclo de vida de producción de datos	Opcional
'9F65'	La longitud máxima del campo de datos en el mensaje de comando	Obligatorio

Tabla 44 - Codificación de la información de dominios cuando se usa el *SELECT*

Mensaje de respuesta de estado del proceso

Una ejecución exitosa de la orden se indicará con palabras de estado del '90 ''00'.
La orden podría devolver la condición de advertencia siguiente cuando el dominio de seguridad emisor se está seleccionando.

Este comando puede devolver una de las condiciones de error que se enumeran en la tabla 45.

SW1	SW2	Significado
'68'	'82'	Mensajería segura no soportada
'6A'	'81'	Función no soportada, por ejemplo, Ciclo de vida de la tarjeta está *CARD_LOCKED*
'6A'	'82'	Seleccionar aplicación\archivo no encontrado

Tabla 45 - Condición de error cuando se usa el *SELECT*

Descripción del *INITIALIZE UPDATE Command*

Este comando inicia la apertura de una sesión de canal seguro.

En cualquier momento durante un canal seguro actual, el comando *INITIALIZE UPDATE* puede ser emitido a la tarjeta para iniciar una nueva sesión de canal seguro.

Estructura del mensaje de comando

El mensaje de comando *INITIALIZE UPDATE* se codifica según la tabla 46.

Código	Valor	Significado
CLA	'80'	
INS	'50'	*INITIALIZE UPDATE*
P1	'xx'	Número de versión de clave
P2	'xx'	Identificador de clave
Lc	'08'	Longitud del desafío anfitrión
Data	'xx xx…'	Reto del anfitrión
Le	'00'	

Tabla 46 - Codificación del comando *INITIALIZE UPDATE*

Parámetro de Control de Referencia P1 - Número de versión de clave

El número de versión de clave define el dominio de seguridad que se utiliza para iniciar la sesión del canal seguro. Si este valor es cero, se utilizará la primera clave disponible elegida por el dominio de seguridad.

Parámetro de Control de Referencia P2 - Identificador de clave

La clave de identificación junto con el número de versión de clave definido en el parámetro P1 proporciona una referencia única para el conjunto de teclas que se utilizan para iniciar la sesión del canal seguro.

Campo de datos enviados en el mensaje de comando

El campo de datos del mensaje de comando contiene 8 bytes de desafío host. Este reto, elegido por la entidad externa de la tarjeta debe ser único para esta sesión.

El campo de datos del mensaje de respuesta contendrá la concatenación sin delimitadores de los elementos que se muestran en la tabla 47.

Nombre	Longitud

Datos de diversificación de llave	10 bytes
Información de llave	2 bytes
Desafío de tarjeta	8 bytes
Criptograma de la tarjeta	8 bytes

Tabla 47 - Concatenación del mensaje de respuesta

Los datos de la llave de diversificación normalmente son utilizados por un sistema para obtener las llaves de las tarjetas estáticas.

La información de la llave incluye el número de versión de clave y el identificador de protocolo *Secure Chanel* que en este caso es '01' y que se utiliza en el inicio de la sesión del canal seguro.

El desafío de la tarjeta es un número aleatorio generado internamente. El criptograma de la tarjeta es un criptograma de autenticación.

Estado de procesamiento de retorno en el mensaje de respuesta

Una exitosa ejecución de la orden se indicará con la instrucción '90' '00'.
Este comando puede devolver una condición de error general.

La condición de error general se lleva a cabo como se muestra en la tabla 48.

SW1	SW2	Significado
'6A'	'88'	No se encuentran los datos de referencia

Tabla 48 - Condición de error cuando se usa el *INITIALIZE UPDATE*

Descripción del *EXTERNAL AUTHENTICATE Command*

El comando *EXTERNAL AUTHENTICATE* es utilizado por la tarjeta para autenticar el host y para determinar el nivel de seguridad requerido para los comandos posteriores. Una ejecución exitosa del comando *INITIALIZE UPDATE* precederá este comando.

Estructura del mensaje de comando

El mensaje de comando *EXTERNAL AUTHENTICATE* se codifica según la tabla 49.

Código	Valor	Significado
CLA	'84'	
INS	'82'	*EXTERNAL AUTHENTICATE*
P1	'xx'	Nivel de seguridad
P2	'00'	Parámetro de control de referencia P2
Lc	'10'	Longitud del criptograma anfitrión y MAC
Data	'xx xx…'	Anfitrión del criptograma y MAC
Le		No presente

Tabla 49 - Codificación del comando *EXTERNAL AUTHENTICATE*

Parámetro de control de referencia P1 - Nivel de seguridad

El parámetro de control de referencia P1 define el nivel de seguridad para los comandos de mensajería segura que se muestran en la tabla 50.

B8	B7	B6	B5	B4	B3	B2	B1	Descripción
0	0	0	0	0	0	1	1	C-Descifrado y MAC
0	0	0	0	0	0	0	1	C-MAC
0	0	0	0	0	0	0	0	No se espera mensajería segura

Tabla 50 - Codificación del parámetro P1 con el *EXTERNAL AUTHENTICATE*

Parámetro de referencia de control P2

El parámetro de control de referencia P2 siempre se fijará a ‘00’.

Campo de datos enviados en el mensaje de comando

El campo de datos del mensaje de comando contiene el criptograma anfitrión y el *MAC* comando APDU.

Campo de datos devueltos en el mensaje de respuesta

El campo de datos del mensaje de respuesta no está presente.

Procesamiento de estados devueltos en el mensaje de respuesta

Una exitosa ejecución de la orden se indicará con la instrucción '90' '00'.

Este comando puede devolver una condición de error general como se muestra en la tabla 51.

SW1	SW2	Significado
'63'	'00'	La autenticación del criptograma anfitrión ha fallado

Tabla 51 - Condición de error cuando se usa el *EXTERNAL AUTHENTICATE*

Descripción del *PUT KEY command*

Este comando es usado para:

- Remplazar una llave existente con una nueva: la llave nueva tiene la misma versión o una diferente, pero el mismo número de identificador de la que se quiere remplazar.
- Remplazar múltiples llaves existentes con una nueva: la llave nueva puede tener la misma versión o una diferente (la misma para todas las llaves nuevas), pero deben tener el mismo identificador de la llave que las que están siendo reemplazadas.
- Adicionar una llave nueva: la nueva llave debe tener una combinación diferente de identificador de llave/versión de las llaves existentes.
- Adicionar múltiples llaves nuevas: las nuevas llaves deben tener diferentes combinaciones de identificador de llave/versión (idéntica para las llaves nuevas) de las llaves existentes.

Cuando la operación administración de llaves requiere múltiples comandos PUT KEY, se recomienda encadenar los múltiples comandos para asegurar la integridad de la operación.

La estructura del mensaje del comando se muestra en la tabla 52.

Código	Valor	Significado
CLA	'80'o '84'	
INS	'D8'	*PUT KEY*
P1	'xx'	Parámetro de control de referencia P1
P2	'xx'	Parámetro de control de referencia P2
Lc	'xx'	Longitud del campo de datos
Data	'xxxx…'	Datos de la Llave(Y MAC si está presente)
Le	'00'	

Tabla 52 - Codificación del comando *PUT KEY*

Parámetro de control de referencia P1

Este parámetro define un número de versión para la llave y si más comandos *PUT KEY* seguirán a este. El número de versión de la llave identifica una llave o un grupo de llaves que están presentes en la tarjeta. Un valor '00' indica que un nuevo grupo de llaves está siendo adicionado. El nuevo número de versión de la llave se indica en el campo de datos del mensaje de comando. El número de versión de la llave esta codificado desde '01' hasta '7f'.

Este parámetro será codificado de acuerdo a la tabla 53.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
0								Último o único comando
1								Más comandos *PUT KEY*
	*	*	*	*	*	*	*	Número de versión de la llave

Tabla 53 - Codificación del parámetro P1 cuando se usa el *PUT KEY*

Parámetro de control de referencia P2

Este parámetro define el identificador de la llave y si una o múltiples llaves están contenidas en el campo de datos.

Cuando una llave está contenida en el mensaje del comando el parámetro indica el identificador de la llave de esta. Cuando múltiples llaves están contenidas en el mensaje, el parámetro indica el identificador de la primera llave en el campo de datos. Cada llave subsecuente en el mensaje tiene implícita una llave que será secuencialmente incrementada en uno comenzando por la primera llave. Los identificadores son codificados desde '00' hasta '7f'.

Este parámetro se codificará de acuerdo a la tabla 54.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
0								Llave simple
1								Llave múltiple
	*	*	*	*	*	*	*	Identificador de la llave

Tabla 54 - Codificación del parámetro P2 cuando se usa el *PUT KEY*

Datos enviados en el mensaje del comando

Formato 1

El campo de datos del mensaje del comando contiene un nuevo número de versión de la llave seguido por uno o múltiples campos de datos de llaves como se representa en la tabla 55.

Nuevo número de versión
Campo de datos de la llave(Llave implícita id P2+0)
Campo de datos de la llave(Llave implícita id P2+1)
Campo de datos de la llave(Llave implícita id P2+2)

Tabla 55 - Número de versión de la llave en el campo de datos con el *PUT KEY*

El nuevo número de versión define uno de los casos siguientes:

- El número de versión de un nuevo grupo de llaves a ser creadas en la tarjeta (El número de versión indicado en P1 es 0).
- El número de versión de un nuevo grupo de llaves que remplazarán un grupo de las llaves existentes (P1 diferente de 0).

Si el campo de datos contiene múltiples llaves, las llaves comparten el número de versión y la secuencia en el campo de datos del comando refleja la secuencia incremental de los identificadores.

El campo de datos de la llave se estructura acorde a la tabla 56.

Longitud	Significado
1	Tipo de llave
1	Longitud de la llave o componente de la llave
Variable: 1-n	Llave o componente de la llave
1	Longitud del valor de chequeo de la llave
Variable: 0-n	Valor de chequeo de la llave(Si existe)

Tabla 56 - Estructura del campo de datos de la llave con el *PUT KEY*

Formato 2

Reservado para uso futuro.

Procesando reglas

Cuando se remplazan las llaves, las nuevas llaves deberán ser presentadas a la tarjeta con el mismo formato que tenían las anteriores en otras palabras no es posible cambiar el tamaño y el algoritmo criptográfico asociado a una llave.

Cuando se usa el comando para cargar o remplazar llaves privadas o secretas, los valores de la llave deberán ser encriptados, la referencia a la llave de encriptado y el algoritmo usado serán conocidos implícitamente en el contexto actual. Los valores de la llave pública serán mostrados en texto claro.

Cuando el encadenado se usa para cargar o remplazar una llave conformada por más de un componente, el comando subsecuente debe hacer referencia al mismo identificador de llave y número de versión de llave del primer *PUT KEY* usado por el primer componente. Un componente de llave no deberá separarse en dos comandos *PUT KEY*.

Si el campo de datos contiene múltiples llaves o componentes de llaves, la tarjeta deberá manejar las múltiples llaves o componentes de manera atómica. Cuando el comando *PUT KEY* es encadenado, la tarjeta deberá manejar los componentes transferidos en la cadena de comandos de manera atómica.
El componente modular de una Llave RSA deberá ser el primer componente de la llave.

Es responsabilidad del receptor interno de la tarjeta verificar el valor de chequeo de la llave cuando está presente.

Estructura del mensaje de respuesta

Datos retornados en el mensaje de respuesta.

Formato 1

El campo de datos del mensaje de respuesta contiene en texto claro el número de versión de la llave seguido del valor(es) de chequeo de la llave, no precedidos por una longitud, si existe(n), como se

presente en el mensaje del comando. El servidor de personalización puede usar el número de versión de la llave retornado y el valor de chequeo para verificar que se haya cargado correctamente la(s) llave(s).

Procesando el estado retornado en el mensaje de respuesta

Una ejecución exitosa del comando deberá ser indicada con la instrucción '90''00'.Este comando puede contener uno de los errores generales listados en la tabla 57.

SW1	SW2	Significado
'65'	'81'	Fallo de memoria
'6A'	'84'	No hay suficiente espacio en memoria
'6A'	'88'	Datos referenciados no encontrados
'94'	'84'	Algoritmo no soportado
'94'	'85'	Valor de chequeo de llave inválido

Tabla 57 - Condición de error cuando se usa el *PUT KEY*

Descripción del *LOAD command*

Múltiples comandos *Load* pueden ser usados para transferir el archivo de carga a la tarjeta. El archivo de carga se divide en componentes más pequeños para la transmisión. Cada enumeración del comando *load* deberá ser estrictamente secuencial incrementando en 1.La tarjeta será informada del último bloque del Archivo de carga.

Después de recibir el último bloque del archivo de carga, la tarjeta deberá ejecutar los procesos internos necesarios para el archivo de carga y cualquier proceso adicional identificado en el comando *Install* (para carga) que precede al comando *Load*.

Estructura del mensaje del comando

Este comando será codificado de acuerdo a la tabla 58.

Código	Valor	Significado
CLA	'80'o '84'	
INS	'E8'	*LOAD*
P1	'xx'	Parámetro de control de referencia P1

P2	'xx'	Número de bloqueo
Lc	'xx'	Longitud del campo de datos
Data	'xxxx…'	Datos de la Llave(Y MAC si está presente)
Le	'00'	

Tabla 58 - Codificación del comando *LOAD*

Parámetro de control de referencia P1

La tabla 59 describe la codificación de ese parámetro, indicando si el bloque contenido en el mensaje de comando es el primero en una secuencia de bloques o el último.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
0								Más bloqueo
1								Último bloqueo
	X	X	X	X	X	X	X	RFU

Tabla 59 - Codificación del parámetro P1 cuando se usa el *LOAD*

Parámetro de control de referencia P2

Este parámetro contiene el número de bloqueo, y deberá ser codificado secuencialmente de 00 hasta FF.

Datos enviados en el mensaje de Comando

El campo de datos del mensaje del comando contiene una porción del archivo de carga. Un archivo de carga de *GlobalPlatform* completo está estructurado como se define en la tabla 60.

Etiqueta	Longitud	Valor
'C3'	Variable	Bloqueo DAP
'4F'	5-16	Dominio de seguridad AID
'C3'	Variable	Archivo de datos de carga de bloque de firma
:	:	:
:	:	:
'E2'	Variable	Bloqueo DAP
'4F'	5-16	Dominio de seguridad AID

'C3'	Variable	Archivo de datos de carga de bloque de firma
'C4'	Variable	Archivo de datos de carga de bloque

Tabla 60 - Estructura de un archivo de *GlobalPlatform* completo con el *LOAD*

Estructura del mensaje de respuesta

Los datos deberán ser siempre retornados en el mensaje de respuesta. El contenido de los datos solo será relevante en caso de Administración Delegada, si un último comando *LOAD* está siendo enviado al dominio de seguridad con el privilegio de administración delegada, la respuesta puede estar presente en el dato dependiendo de la política de seguridad del distribuidor de la tarjeta.

Datos retornados en el mensaje de respuesta

Si el comando *Load* no contiene el último bloque de la secuencia, un solo byte 00 deberá ser retornado indicando que no hay más datos presentes.

Si el dominio de seguridad del distribuidor procesa el comando *LOAD* conteniendo el último bloque en la secuencia, un solo byte 00 deberá ser retornado indicando que no hay datos adicionales.

Para un comando *Load* conteniendo el último bloque de una secuencia siendo enviada a un dominio de seguridad con privilegios de administración delegada, los datos pueden contener la confirmación del procedimiento de confirmación. La longitud total de dicho mensaje no debe exceder los 256 bytes.

La tabla 61 describe la estructura de los datos de la respuesta del *LOAD*.

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud de la confirmación de carga
Condicional	0-1	Confirmación de carga

Tabla 61 - Estructura de datos de respuesta del *LOAD*

Procesando el estado retornado en el mensaje de respuesta

Una ejecución exitosa del comando será indicada con la instrucción ‘90’’00’.Este comando puede retornar errores generales o una de las condiciones de error que se muestra en la tabla 62.

SW1	SW2	Significado
'65'	'81'	Fallo de memoria
'6A'	'84'	No hay suficiente espacio en memoria

Tabla 62 - Condición de error cuando se usa el *LOAD*

Descripción del *INSTALL command*

Definición y rango

Este comando es emitido a un dominio de seguridad para iniciar o llevar a cabo los pasos requeridos por el administrador de contenidos de la tarjeta.

Estructura del mensaje del comando

El mensaje del comando *Install* deberá ser codificado de acuerdo a la tabla 63.

Código	Valor	Significado
CLA	'80'o '84'	
INS	'E6'	*INSTALL*
P1	'xx'	Parámetro de control de referencia P1
P2	'00'	Parámetro de control de referencia P2
Lc	'xx'	Longitud del campo de datos
Data	'xxxx…'	Datos de la Llave(Y MAC si está presente)
Le	'00'	

Tabla 63 - Codificación del comando *INSTALL*

Parámetro de control de referencia P1

Este parámetro del comando *Install* está codificado de acuerdo a la tabla 64.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
		1	0	0	0	0	0	Para personalización
		0	1	0	0	0	0	Para extradición
		0	0	1	X	0	0	Para hacer seleccionable
		0	0	X	1	0	0	Para instalar

		0	0	0	0	1	0	Para cargar
X	X							RFU

Tabla 64 - Codificación del parámetro P1 cuando se usa el *INSTALL*

Los comandos b6 a b1 deberán ser codificados de la siguiente forma:

b6=1 indica que el dominio de seguridad actualmente seleccionado deberá personalizar una de sus aplicaciones asociadas y será esperado un almacén de datos subsecuentes.

b5=1 indica que la aplicación será extraditada.

b4=1 indica que la aplicación deberá ser seleccionable .Esto se utiliza cuando una aplicación está siendo instalada o ya está instalada.

b3=1 indica que la aplicación deberá ser instalada.

b2=1 indica que el archivo de carga debe ser cargado. Un comando *Load* subsecuente es esperado.
Una combinación de las opciones (para Instalar) y (para hacer seleccionable) puede ser aplicada.

Parámetro de control de referencia P2

Este parámetro siempre deberá ser puesto en '00'.

Datos enviados en el mensaje del comando

El campo de datos del mensaje del comando contiene datos codificados TLV. Estos datos son representados sin delimitadores.

Datos del *install* (Para cargar)

La tabla 65 detalla el comando *Install* (Para cargar)

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud del Archivo de cargado AID
Obligatorio	5-16	Archivo de cargado AID
Obligatorio	1	Longitud del Dominio de seguridad
Condicional	0-16	Dominio de seguridad

Obligatorio	1	Longitud del comodín del bloque de datos del archivo a cargar
Condicional	0-n	Comodín del bloque de datos del archivo a cargar
Obligatorio	1	Longitud del campo de los parámetros de carga
Condicional	0-n	Campo de los parámetros de carga
Obligatorio	1	Longitud del objeto Cargado
Condicional	0-n	Objeto cargado

Tabla 65 - Comando *INSTALL* cuando se hace la operación de cargar

El comodín del bloque de datos del archivo de carga y el objeto cargado son obligatorios para la administración delegada. El comodín del bloque de datos del archivo de carga es obligatorio si el archivo cargado contiene bloques DAP. En otros casos el comodín del bloque de datos del archivo de cargado es opcional y puede ser verificado por la tarjeta. El archivo de cargado AID y los parámetros de carga deberán ser consistentes con la información contenida en el Bloque de archivos de carga (si existe).

La tabla 66 detalla los datos del *Install* (Para instalar)

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud de archivo de cargado ejecutable AID
Obligatorio	5-16	Archivo de cargado ejecutable AID
Obligatorio	1	Longitud del módulo ejecutable AID
Obligatorio	5-16	Módulo ejecutable AID
Obligatorio	1	Longitud de la aplicación AID
Obligatorio	5-16	Aplicación AID
Obligatorio	1	Longitud de los privilegios de la Aplicación
Obligatorio	1	Privilegios de la aplicación
Obligatorio	1	Longitud de los parámetros de instalación
Obligatorio	2-n	Parámetros de instalación
Obligatorio	1	Longitud del objeto Instalar
Condicional	0-n	Objeto Instalar

Tabla 66 - Comando *INSTALL* cuando se hace la operación de instalar

El objeto instalar es obligatorio para la administración delegada, de lo contrario no deberá estar presente.

El módulo ejecutable AID es el AID del módulo ejecutable previamente cargado. Este deberá estar presente dentro del archivo de carga.

Las tarjetas *GlobalPlatform* usan la instancia AID para indicar el AID con el cual será seleccionada la aplicación instalada.

La presencia de los privilegios de aplicación es requerida. Si una aplicación está solamente siendo instalada y no hecha seleccionable con el mismo comando *install* los privilegios seleccionados por defecto no pueden ser alterados.

En la instancia AID, los privilegios de la aplicación y los parámetros específicos deberán ser reconocidos para la misma.

La tabla 67 muestra los datos del *Install* (Para hacer seleccionable)

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud del AID de la aplicación
Obligatorio	5-16	AID de la aplicación
Obligatorio	1	Longitud de los privilegios de la aplicación
Obligatorio	1	Privilegios de la aplicación
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud del Objeto *install*
Condicional	0-n	Objeto *Install*

Tabla 67 - Comando *INSTALL* cuando se hace la operación de seleccionar

Si los privilegios seleccionados por defecto son puestos en el campo de privilegios de la aplicación, el registro de *GlobalPlatform* deberá actualizarse acorde a las reglas.

Cualquier otro privilegio puesto en el campo de privilegios de la aplicación será ignorado por la tarjeta.
El objeto *Install* es impuesto para la administración delegada en un comando *Install* (para hacerla seleccionable).

La tabla 68 muestra los datos del *Install* (para extradición)

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud del AID del dominio de seguridad
Obligatorio	5-16	AID del dominio de seguridad
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud del AID de la Aplicación
Obligatorio	5-16	AID de la aplicación
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud del Objeto Extradición
Condicional	0-n	Objeto Extradición

Tabla 68 - Comando *INSTALL* cuando se hace la operación de extradición

El AID del dominio de seguridad indica a qué dominio de seguridad esta aplicación está siendo extraditada.

La tabla 69 muestra los datos del *Install* (Para personalización)

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud del AID de la aplicación
Obligatorio	5-16	AID de la aplicación
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud='00'
Obligatorio	1	Longitud='00'

Tabla 69 - Comando *INSTALL* cuando se hace la operación de personalización

Parámetros del *Install* (Para cargado) y el *Install* (Para instalar)

Los campos de parámetros del cargar y el *install* son valores TLV estructurados incluyendo parámetros específicos del sistema y parámetros específicos de las aplicaciones opcionales. Mientras la presencia de parámetros específicos de la aplicación es opcional tanto para la instalación como para el cargado, incluso

si están presentes, no se requiere que el sistema los tome en cuenta, su presencia deberá ser anticipada pero el contenido puede ser ignorado.

La tabla 70 identifica las posibles etiquetas a usar en el campo de parámetros del cargar.

Etiqueta	Longitud	Valor (nombre)	Presencia
'EF'	Variable	Parámetros específicos del sistema	Condicional
'C6'	2	Código de limite de espacio no volátil	Opcional
'C7'	2	Datos de límite de espacio volátil	Opcional
'C8'	2	Datos de límite de espacio no volátil	Opcional

Tabla 70 - Etiquetas para el campo de parámetros de cargar con el comando *INSTALL*

La presencia de las etiquetas C6, c7 o C8 en el comando *Install* (para cargar) indica el tamaño mínimo de los recursos que se requiere se encuentren disponibles en la tarjeta. La indicación habilita el OPEN para rechazar la carga requerida si los recursos restantes en la tarjeta son insuficientes para el *Load* o la subsecuente instalación.

La tabla 71 identifica las posibles etiquetas a usar en el campo de parámetros de la instalación.

Etiqueta	Longitud	Valor (nombre)	Presencia
'C9'	Variable	Parámetros específicos de la aplicación	Condicional
'EF'	Variable	Parámetros específicos del sistema	Opcional
'C7'	2	Datos de límite de espacio volátil	Opcional
'C8'	2	Datos de límite de espacio no volátil	Opcional

Tabla 71 - Etiquetas para la instalación con el comando *INSTALL*

Cuando está presente en el comando *Install* (Para instalar), la información contenida en la etiqueta C9 (Longitud +datos) deberá ser pasada a la aplicación que está siendo instalada. La presencia de las etiquetas C7 o C8 en el *Install* (Para instalar) indica el tamaño máximo de recursos de la tarjeta ubicados para la aplicación. Esta información puede ser usada por el OPEN para validar los recursos pedidos por la aplicación.

Estructura del mensaje de respuesta

Un campo de datos siempre deberá ser retornado en el mensaje de respuesta. El contenido del campo de datos solo será relevante en caso de administración delegada, si un *INSTALL* (Para instalar), Install (Para hacer seleccionable) o *INSTALL* (para extradición) es transmitido a un dominio de seguridad con privilegios de administración delegada.

Datos retornados en el mensaje de respuesta

Si el dominio de seguridad del distribuidor procesa el comando *Install*, un solo byte '00' deberá ser retornado indicando que no hay más datos adicionales presentes

Si un comando *Install* (Para cargar) o un comando Install (para hacer seleccionable) está siendo enviado a un dominio de seguridad con privilegios de administración delegada, un solo byte '00' deberá ser retornado indicando que no hay más datos adicionales presentes.

Para un *Install* (para instalar), *Install* (Para hacer seleccionable) e *Install* (Para extradición) siendo enviados a un dominio de seguridad con privilegios de administración delegada, el campo de datos puede contener la confirmación del proceso de instalación. La longitud global del mensaje de respuesta no deberá exceder de los 256 bytes.

La tabla 72 describe la estructura del campo de datos del mensaje de respuesta del *Install*.

Presencia	Longitud de los bytes	Nombre
Obligatorio	1	Longitud de la confirmación de instalación
Condicional	0-n	Confirmación de Instalación

Tabla 72 - Estructura de campo de datos del mensaje de respuesta con el comando *INSTALL*

Procesando el estado retornado en el mensaje de respuesta

Una ejecución exitosa del comando deberá ser indicada con las palabras de estado '90''00'.Este comando puede contener uno de los errores que se muestran en la tabla 73.

SW1	SW2	Significado
'65'	'81'	Fallo de memoria
'6A'	'80'	Parámetros incorrectos en el campo de datos
'6A'	'84'	No hay suficiente espacio en memoria
'6A'	'88'	Datos referenciados no encontrados

Tabla 73 - Condición de error cuando se usa el *INSTALL*

Descripción del *GET STATUS command*

Este comando es usado para obtener la información del estado de acuerdo a un criterio de búsqueda dado del dominio de seguridad del distribuidor, el archivo de carga ejecutable, la aplicación o dominio de seguridad del ciclo de vida. Este comando complementa el SET STATUS Command.

Estructura del mensaje del comando

El mensaje del comando deberá estar codificado de acuerdo a la tabla 74.

Código	Valor	Significado
CLA	'80'o '84'	
INS	'F2'	*GET STATUS*
P1	'xx'	Parámetro de control de referencia P1
P2	'xx'	Parámetro de control de referencia P2
Lc	'xx'	Criterio de búsqueda (y MAC si está presente)
Le	'00'	

Tabla 74 - Codificación del comando *GET STATUS*

Parámetros de control de referencia P1

El parámetro P1 es usado para seleccionar un subconjunto de estados a ser incluido en el mensaje de respuesta. Los siguientes valores de dicho parámetro pueden ser aplicables.

'80' Dominio de seguridad del distribuidor solamente. En este caso el criterio de búsqueda se ignora y la información del dominio de seguridad del distribuidores retornada.

'40' Aplicación y dominio de seguridad solamente.

'20' Archivos de carga ejecutables.

'10' Archivos de carga ejecutables y sus módulos ejecutables solamente.

Otros valores para este parámetro de referencia pueden ser asignados en el futuro.

La habilidad de diferenciar entre un dominio de seguridad y una aplicación es alcanzada por medio de los privilegios de aplicación.

Parámetro de control de referencia P2

Este parámetro controla el número de comandos *Get Status* consecutivos e indica el formato del mensaje de respuesta. Deberá ser codificado acorde a la tabla75.

B8	B7	B6	B5	B4	B3	B2	B1	Significado
X	X	X	X	X	X			RFU
						X	0	Obtener la primera o todas las ocurrencias
						X	1	Obtener la(s) próxima(s) ocurrencia(s)
						0	X	Estructura de los datos de respuesta
						1	X	Datos de respuesta

Tabla 75 - Codificación del parámetro P2 cuando se usa el *GET STATUS*

El indicador para obtener nueva ocurrencia (*get next occurrence*) deberá ser rechazado si no hay previamente un *GET STATUS* (*Get first or all ocurrence*(s)) dentro de la sesión de aplicación actual.

Datos enviados en el mensaje del comando.

El mensaje de este comando contiene el código calificador de búsqueda TLV.

La etiqueta 4f deberá ser usada para indicar el Identificador de la Aplicación (AID).Deberá ser posible buscar todas las ocurrencias que se correspondan con el criterio de búsqueda de acuerdo con el parámetro P1 usando un criterio de búsqueda '4f''00'.

Deberá ser posible buscar todas las aplicaciones con el mismo RID.

Estructura del mensaje de respuesta

Datos retornados en el mensaje de respuesta.

Basado en el criterio de búsqueda de los datos del comando y de los parámetros P1 y P2 pueden producirse múltiples ocurrencias con la estructura de datos de la tabla 76.

Longitud	Valor	Significado
1	'xx'	Longitud del identificador de la aplicación
1-n	'xxxx…'	Identificador de la aplicación
1	'xx'	Ciclo de vida del estado
1	'xx'	Privilegios de la Aplicación

Tabla 76 - Estructuras de datos posibles con el comando *GET STATUS*

Etiquetas	Longitud	Valor	Presencia
'E3'	Variable	Datos relacionados con el Registro de *GlobalPlatform*	
'4F'	1-n	Identificador de la aplicación	
'9F70'	'01'	Ciclo de vida del estado	
'C5'	'01'	Privilegios de la Aplicación	Condicional
'84'	1-n	Primer o único Módulo ejecutable AID	Condicional
…	…	…	…
'84'	1-n	Último Modulo Ejecutable	Condicional

Tabla 77 - Estructura de dominios para el comando *GET STATUS*

En la tabla 77 se muestra como el campo de privilegios de la aplicación deberá estar presente en el dominio de seguridad del distribuidor, los dominios de seguridad y la aplicación, debiendo estar ausente para los módulos de cargado ejecutables. Los módulos ejecutables solo estarán presentes si el parámetro P1 indica archivos de cargado ejecutable y sus módulos de cargado son solamente ejecutables.

Procesando el estado retornado en el mensaje de respuesta

Una ejecución exitosa del comando deberá ser indicada con las palabras de estado '90"00'. El comando puede retornar la condición de advertencia que se muestra en la tabla 78 para indicar que un *Get Status* (*Get next ocurrence*(s)) subsecuente puede ser emitido para obtener datos adicionales.

SW1	SW2	Significado
'63'	'10'	Más datos disponibles.

Tabla 78 - Condición de advertencia cuando se usa el *GET STATUS*

Este comando puede retornar un error general acorde a las condiciones de error listadas en la tabla 79.

SW1	SW2	Significado
'6A'	'88'	Datos referenciados no encontrados
'6A'	'80'	Valores incorrectos en los datos del comando.

Tabla 79 - Condición de error cuando se usa el *GET STATUS*

Anexo 10: Caso de prueba de caja Blanca de un segmento de código correspondiente a la historia de usuario *Authentication*.

```
public security niveldeseguridad(String nivel)
    {
     security a=null; (1)
     if (nivel.equals("0")) (2)
      {
        a= security.NoSecurity; (3)
      }
     else if(nivel.equals("1")) (4)
      {
        a=security.MAC; (5)
      }
     else if(nivel.equals("3")) (6)
      {
        a=security.MACEncryption; (7)
      }
   return a; (8)
}
```

La figura 15 muestra el grafo de flujo del caso de prueba Authentication.

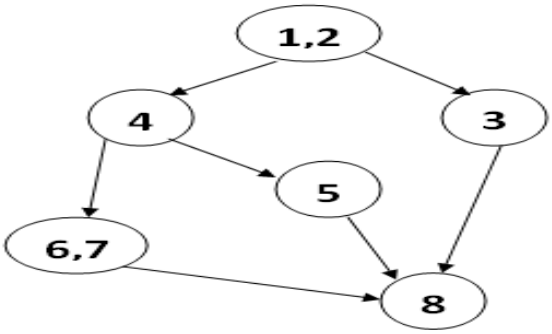

Figura 15: Grafo de flujo del caso de prueba *Authentication*. Fuente: Elaboración propia.

Complejidad ciclomática

V (G): Número de regiones del grafo. **V (G)** = A – N + 2 **V (G)** = P + 1

A: Número de aristas del grafo. A= 7

N: Número de nodos. N= 6

P: Número de nodos predicados. P= 2

V (G) = 3

Caminos:

1, 2, 3, 8

1, 2, 4, 5, 8

1, 2, 4, 6, 7, 8

Casos de prueba para cada camino

Camino: 1, 2, 3, 8

Entrada: Valor cero.

Salida: devuelve como nivel de seguridad NoSecurity.

Precondiciones: debe haberse ejecutado el *Select* y el *InitializeUpdate*.

Camino: 1, 2, 4, 5, 8

Entrada: Valor 1.

Salida: devuelve como nivel de seguridad MAC.

Precondiciones: debe haberse ejecutado el *Select* y el *InitializeUpdate*.

Camino: 1, 2, 4, 6, 7, 8

Entrada: Valor 3.

Salida: devuelve como nivel de seguridad *MACEncryption*.

Precondiciones: debe haberse ejecutado el *Select* y el *InitializeUpdate*.