Análisis Formal de la Instalación de Aplicaciones en MIDP 3.0

Tesina de grado presentada

por

Cristián Germán Prince

Legajo: P-2938/6

al

Departamento de Ciencias de la Computación

en cumplimiento de los requerimientos para

la obtención del grado de

Licenciado en Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

Av. Pellegrini 250, Rosario, República Argentina

Diciembre de 2014

Director

Carlos D. Luna

Instituto de Computación, Facultad de Ingeniería
Universidad de la República
Julio Herrera y Reissig 565, Montevideo, Uruguay

Resumen

Hoy en día, la telefonía celular se ha vuelto imprescindible para el desarrollo de la vida cotidiana. Además de sus servicios básicos de comunicación, los dispositivos móviles permiten almacenar datos confidenciales y, descargar y ejecutar aplicaciones, lo que conlleva un riesgo para la integridad y privacidad de la información que uno tiene. Es por ello que la seguridad pasa a tener un rol fundamental en el desarrollo y en el éxito de las tecnologías móviles.

En la plataforma Java Micro Edition, el ambiente de ejecución estándar para teléfonos celulares está provisto por el Perfil para Dispositivos de Información Móviles (MIDP, por sus siglas en Inglés). Para la versión 2.0 de MIDP, Zanella, Luna y Betarte, propusieron la primera especificación formal de su modelo de seguridad. Se construyó un modelo abstracto en el Cálculo de Construcciones Inductivas (CIC) del estado de un dispositivo y de los posibles eventos que inducen cambios en dicho estado. Además, se han demostrado propiedades deseables para cualquier implementación del estándar, lo que lo convierte en una poderosa herramienta para razonar sobre el modelo de seguridad y facilitar su comprensión.

Una primera extensión de este modelo fue realizada por Gustavo Mazeikis, quien realizó la formalización del Modelo de Autorización para MIDP 3.0 (la más reciente versión del perfil) y demostró la preservación de las propiedades de seguridad consideradas para MIDP 2.0.

En este trabajo se presenta una nueva extensión de la especificación formal referida. Este proyecto conserva las propiedades de seguridad demostradas por Mazeikis e incorpora el manejo y almacenamiento de vendedores y certificados en el dispositivo que son utilizados al momento de la instalación, como así también cambios sustanciales sobre la definición de este evento crítico. El aporte principal de este trabajo es la verificación sobre el vendedor de una aplicación y su certificado asociado, abarcando el caso en que éste no posea uno. De esta manera, un usuario puede instalar aplicaciones de vendedores cuyos certificados autentican que son confiables o aplicaciones de vendedores que no poseen dicha certificación, como podría ser el caso de un programador que desarrolla una aplicación para uso propio o compartido. En el marco del formalismo extendido, se propone un algoritmo para realizar la instalación de una aplicación y se demuestra su corrección.

Índice general

Capítulo 1

Introducción

La tecnología se ha convertido en una parte fundamental de nuestras vidas. Hoy en día, el alcance de dispositivos tecnológicos ha llegado a todos los rincones del mundo y uno de los más utilizados a nivel global ha sido el teléfono celular. Ya han dejado de ser simples instrumentos para realizar llamadas telefónicas y se han convertido en herramientas imprescindibles para el desarrollo de nuestras vidas cotidianas. Inclusive, el envío de mensajes de texto ha dejado de ser una funcionalidad que sorprendía a más de uno. La posibilidad de realizar video conferencias, chatear, navegar por Internet a altas velocidades, verificar el correo electrónico, descargar música y aplicaciones o conectarse a alguna red social, han convertido a los celulares en algo más que simples teléfonos. Todo este tráfico y procesamiento de datos es posible debido a que cada dispositivo móvil está equipado con una máquina virtual de Java (JVM, por sus siglas en Inglés), la cual es un conjunto de programas de software que permiten la ejecución de instrucciones escritas en Java. En este contexto, la transmisión de datos y la ejecución de aplicaciones abren la puerta a cualquier malintencionado que genere código malicioso. Es por ello que garantizar la seguridad, privacidad e integridad de los datos de cada usuario es fundamental en el avance de la computación móvil.

La plataforma Java Standard Edition (JSE) [10], facilita el desarrollo y ejecución de aplicaciones Java en computadoras de escritorio y servidores, como así también, en ambientes embebidos de hoy en día. Pero para desarrollar aplicaciones que sean pequeñas y portables, se utiliza la plataforma Java Micro Edition (JME) [9], que provee un robusto y flexible ambiente para aplicaciones que corren en móviles y otros dispositivos embebidos, como ser, teléfonos móvi-

les, asistentes personales digitales (PDA's) e impresoras, entre otros. Su propósito es encajar en ambientes limitados y hacer posible la creación de aplicaciones Java para que corran sobre dispositivos pequeños con memorias y energía limitada.

Esta plataforma está bajo supervisión de la Java Community Process (JCP) [8], la cual es la responsable del desarrollo de la tecnología Java. Al ser una organización de carácter abierta e inclusiva de miembros activos y entradas públicas de no miembros, principalmente guía el desarrollo y aprobación de especificaciones técnicas de Java. El trabajo de la Comunidad Java dentro del programa JCP ayuda a asegurar los estándares de estabilidad de la tecnología de Java y la compatibilidad multi-plataforma, permitiéndole operar sobre cientos de millones de dispositivos. Es por esto que al ser esta plataforma una especificación informal (para la cual se dan solamente lineamientos de los requerimientos) y no una implementación concreta, queda a cargo de terceros su implementación.

El modelo de seguridad para aplicaciones ejecutadas en teléfonos móviles fue formalmente especificado por Santiago Zanella en [16][17]. En dicho trabajo se enuncian varias propiedades deseables y se demuestran formalmente que son deducibles a partir del modelo construido. De esta forma, se muestra que dicha formalización permite razonar a un nivel de abstracción adecuado sobre las propiedades del modelo de seguridad. A su vez, se puede utilizar como guía para realizar las pruebas que se diseñen sobre implementaciones concretas.

Con el advenimiento de una nueva versión de la especificación informal para dispositivos móviles [7], la segunda formalización y extensión del modelo de seguridad fue presentada por Mazeikis et al [4]. En dicho trabajo, el autor analiza los cambios introducidos en el modelo de seguridad y propone una extensión de la especificación formal mencionada con respecto a las autorizaciones otorgadas a las aplicaciones. La extensión se plantea de forma tal que las propiedades demostradas para la versión anterior sean preservadas, además de enunciar y demostrar nuevas propiedades del modelo de seguridad. En la formalización extendida se propone un algoritmo para autorizar el acceso de aplicaciones a recursos compartidos. De esta forma se mantiene actualizada la formalización, preservando los resultados anteriores y aportando nuevos resultados en las áreas innovadas.

En el presente trabajo se continúa con la línea de Mazeikis [5][6] y se extienden nuevas propiedades sobre la instalación de aplicaciones. Se realizan cambios sustanciales sobre algunas definiciones realizadas en el trabajo de Zanella, en relación a la carga de software, y se

propone, además, un algoritmo para describir dicho proceso. Se formalizan nuevas condiciones de seguridad, de manera de garantizar que se mantengan las propiedades demostradas en las formalizaciones anteriores, garantizando una extensión conservativa de la especificación formal. El desarrollo del código en Coq puede consultarse en [11] o [12].

Este trabajo se organiza de la siguiente manera: en el resto del capítulo 1 se describen informalmente Java Micro Edition y el Perfil MIDP. En el capítulo 2 se formaliza el modelo de seguridad. Dentro del tercer capítulo se explica la extensión propuesta y se demuestran algunas propiedades relevantes de seguridad. En el capítulo 4 se plantea un algoritmo de instalación de aplicaciones y la demostración de su corrección. Finalmente, las conclusiones y trabajos futuros se presentan en el capítulo 5.

1.1. Arquitectura de Java Micro Edition

Esta plataforma fue creada originalmente para enfrentar los problemas derivados de su uso en dispositivos pequeños. La definición de Java ME se concentra en encajar en dicho entorno limitado, y hacer posible la creación de aplicaciones Java para dispositivos con poca memoria, pantalla pequeña y limitada capacidad de energía / autonomía.

Esta tecnología está basada en tres elementos:

- **Configuración:** provee un conjunto básico de bibliotecas y capacidades de la máquina virtual para un gran abanico de dispositivos. Define una mínima plataforma para una categoría horizontal de dispositivos, cada uno con requerimientos similares en cantidades de memoria total y poder de procesamiento.
- **Perfil:** un conjunto de APIs para soportar un conjunto más reducido de dispositivos. Conforma una capa por encima de la configuración y su objetivo principal es garantizar interoperabilidad con una determinada familia de dispositivos o dominio. Los perfiles incluyen bibliotecas de clases que son más específicas de dominio que las clases provistas en una configuración.
- **Paquetes opcionales:** conjunto de APIs de tecnología específica.

La plataforma Java ME es una colección de tecnologías y especificaciones que pueden ser combinadas para construir un entorno de ejecución Java completo que satisfaga los requerimien-

tos de un dispositivo en particular. Esto ofrece una flexibilidad y co-existencia para todos los que forman parte en el ecosistema, de manera de cooperar sin problemas en la oferta de una experiencia más atractiva para el usuario final.

La plataforma Java ME ha sido a su vez dividida en dos configuraciones de base: una para pequeños dispositivos móviles y otra diseñada para dispositivos móviles con más capacidad, como smart phones.

La primera configuración, para pequeños dispositivos, se conoce como Configuración para Dispositivos con Conectividad Limitada o CLDC [2], por sus iniciales en inglés. La segunda, con más capacidad, se denomina Configuración de Dispositivo Conectado, CDC (Figura 1.1).

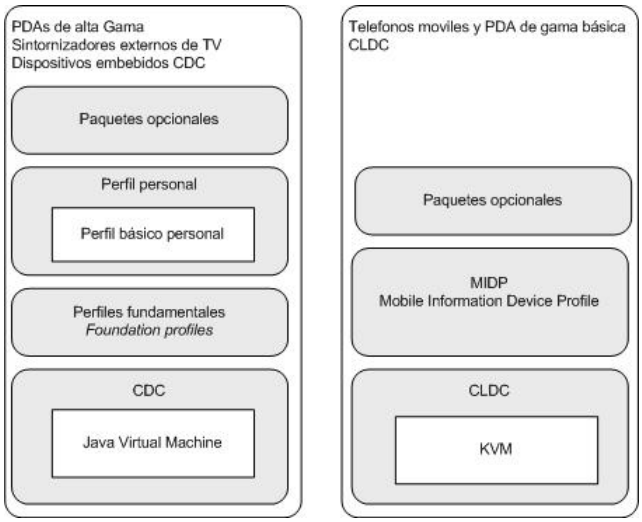

Figura 1.1: Bloques fundamentales de CDC y CLDC

1.2. El Perfil MIDP

El Perfil para Dispositivos de Información Móviles (MIDP) provee el ambiente de ejecución estándar para teléfonos móviles y asistentes de datos personales. Se combina con CLDC para brindar un entorno de ejecución estándar para los dispositivos de información actuales.

1.2.1. La Primera Versión de MIDP y los Ambientes Controlados

MIDP 1.0 es la versión inicial que contiene todas las características requeridas para la codificación Java por Java ME [13]. Trabaja sobre CLDC 1.0 o 1.1. En esta primera versión, cualquier

aplicación era denominada confiable si y sólo si había sido instalada por el fabricante y toda aplicación que fuera descargada por el usuario era caracterizada como no confiable.

Cualquier aplicación no confiable tiene acceso limitado a los recursos sensibles del dispositivo, mientras que las confiables, tienen acceso ilimitado a los mismos. Es por ello que en MIDP 1.0, las no confiables sólo se ejecutan en un ambiente controlado, denominado sandbox (o caja de arena).

1.2.2. La Segunda Versión de MIDP y los Dominios de Protección

En MIDP 2.0 [14], en pos de un modelo menos restrictivo, se abandona la noción de sandbox y se define un nuevo modelo basado en el concepto de dominio de protección.

En esta nueva versión y bajo este nuevo concepto, la confiabilidad sobre una aplicación se determina mediante el uso de firmas digitales utilizando la X.509 Public-Key Infrastructure (PKI, o Infraestructura de Clave Pública X.509). Se considera que una aplicación está firmada si la firma de la aplicación está contenida en la misma, junto con el correspondiente certificado de la clave pública. Dicha firma es creada con la clave privada del vendedor. En el caso en que estos elementos no estén presentes, automáticamente se considera que la aplicación no está firmada.

De por sí, en el dispositivo se preinstalan certificados raíz de entidades certificadoras que son la punta del árbol y es con ellos con quienes se autentica cualquier aplicación que se quiera instalar. Una firma del certificado raíz es algo análogo a notariar una identidad en el mundo físico. En el caso en que no se pueda verificar el origen y autenticidad de una aplicación, ésta se rechaza y no se instala. Por el contrario, si se logra validar el certificado de clave pública del vendedor y se logra verificar la firma de su archivo, se procede con la instalación.

Con el objetivo de brindar mayor flexibilidad a nivel de plataforma, se utilizan dominios de protección, los cuales especifican un conjunto de permisos sobre los recursos sensibles del dispositivo. Cada uno de estos dominios están asociados a un certificado raíz, de forma tal que toda aplicación instalada queda ligada en forma unívoca a los mismos.

En el caso de que una aplicación requiera obligatoriamente más permisos de los que brinda el dominio de protección, su instalación falla. Es por ello que para todas las aplicaciones que se van a instalar, se plantea una nueva restricción: las mismas deben declarar todos los permisos que necesitan para su funcionamiento, sean estos imprescindibles u opcionales.

Los permisos que otorgan los dominios de protección a las aplicaciones que son ligadas a

ellos se dividen en dos grupos: los que son otorgados incondicionalmente y los que requieren autorización previa del usuario. Estos últimos implican un modo que determina la frecuencia y la forma en la que se requerirá la autorización. Existen tres modos denominados:

- **BLANKET:** el permiso otorgado es válido hasta que la aplicación sea desinstalada.
- **SESSION:** el permiso es válido sólo por el tiempo que dure la ejecución de la aplicación. Para una nueva ejecución se solicitará una nueva autorización.
- **ONESHOT:** el permiso es válido sólo para ese uso.

Por lo tanto, el uso de ciertos recursos sensibles involucra una consulta al usuario y en función de una respuesta, se determina la disponibilidad del recurso.

1.2.3. Limitaciones de ambas versiones

Algunas de las limitaciones en la versión 1.0 de MIDP son:

- No tiene rendering de APIs activo.
- No tiene soporte de acceso directo a los píxeles de las imágenes (RGB data).
- No tiene soporte para trabajo en pantalla completa.
- No tiene soporte directo de audio sin una API adicional.
- Sólo soporta HTTP para conexión con el exterior.
- No se puede consultar el estado de las teclas en cualquier momento.
- Las especificaciones no siempre son claras, llevando a diferencias en las implementaciones.

En cuanto a la versión 2.0 de MIDP, algunas de sus limitaciones eran:

- No permitía habilitar y especificar un comportamiento correcto para MIDlets en cada configuración CLDC y CDC (por ejemplo, no permitía múltiples MIDlets concurrentes).
- No permitía bibliotecas compartidas para MIDlets.
- No era compatible con IPv6.
- No permitía múltiples interfaces de red por dispositivo.

1.2.4. La Versión Final: MIDP 3.0

Finalmente, llegamos a la última versión de MIDP, descripta bajo los Requerimientos de Especificación de Java 271 [7].

La Instalación y el Manejo de Vendedores y Certificados

La gran oferta de aplicaciones de cualquier índole conlleva a que muchas veces no se disponga de algunos certificados que validan a dichos productos. Es por ello que existe la posibilidad de almacenar nuevos certificados, en el caso en que dichas aplicaciones posean uno.

Por otra parte, la posibilidad de que cualquier usuario pueda convertirse en desarrollador de su propia aplicación, implica que muchas veces, no se posea un certificado que valide dicho trabajo. No por carecer de un certificado, el código debe ser considerado malicioso.

Es por esto que es necesario llevar un registro de los vendedores y sus certificados, si los tuvieren, de manera de poder instalar cualquier tipo de aplicación, tanto las firmadas como las no firmadas, con el fin de hacer más práctica futuras instalaciones. Con este fin se crea un repositorio de Vendedores y Certificados.

Esta nueva característica implica nuevas propiedades que deben cumplirse antes de poder instalar una aplicación. Más precisamente, se deben verificar ciertas condiciones sobre sus datos de origen. En el momento de proceder con la instalación, se debe comprobar que los permisos y demás privilegios con que gozan las demás aplicaciones ya instaladas no se vean afectadas por la nueva.

El algoritmo de instalación comienza verificando si la aplicación a instalar goza de un certificado (es decir, si está firmada). De confirmarse la existencia de uno, se procede a realizar las verificaciones pertinentes sobre el mismo y sobre los certificados ya almacenados en el repositorio. Una vez finalizadas dichas comprobaciones, se determina si la aplicación se instala o se rechaza.

Caso contrario, si no existe una credencial que autentique la aplicación, se procede a realizar verificaciones sobre el vendedor en el repositorio.

Capítulo 2

Especificación Formal de MIDP 3.0

Tal como lo explica Mazeikis en [4], Zanella, Betarte y Luna han propuesto una especificación formal del modelo de seguridad MIDP 2.0 en el Cálculo de Construcciones Inductivas [CIC], utilizando el asistente de pruebas Coq [15] [1]. Luego, él propuso una extensión para MIDP 3.0 siguiendo la línea de trabajo de los tres primeros autores. En ambas especificaciones formales se plantean y verifican propiedades deseables del modelo de seguridad.

En este capítulo, y para su posterior referencia en la extensión planteada por este trabajo, se describen los principales elementos de la especificación formal planteada para MIDP 3.0, tomando como base lo formalizado en [4] y [16]. Es por este motivo que a continuación se describe la notación utilizada, las definiciones de aplicación, la política de seguridad, el estado del dispositivo, los eventos relacionados con la seguridad y la noción de sesión de ejecución, definida como una secuencia de eventos.

2.1. Notación

En la formalización se usa la notación estándar para los operadores lógicos ($\wedge, \vee, \neg, \rightarrow, \forall, \exists$) y la igualdad. Mientras que la cuantificación universal y la implicación pueden codificarse en Coq utilizando productos dependientes, la igualdad y el resto de los operadores pueden definirse de forma inductiva, como así también los operadores sobre conjuntos. Por su parte, los predicados anónimos se referencian utilizando la notación lambda tradicional, por ejemplo ($\lambda n.n = 0$) es un predicado que aplicado a n es verdadero si n es igual a 0.

El lenguaje de especificación de Coq permite la definición de tipos registros, los cuales son

tipos inductivos no recursivos con funciones de proyección de cada campo del registro. Es decir, la definición de un nuevo registro R se notará como

$$R =_{def} \{campo_1 : A_1, ..., campo_n : A_n\}, \tag{2.1}$$

cuyo único constructor será $mkR : A_1 \to ... \to A_n \to R$ y sus funciones de proyección serán de la forma $campo_i : R \to A_i$. La aplicación de una función de proyección, $campo_i\ r$, se abreviará cuando sea conveniente como $r.campo_i$. Además, se definirá una relación binaria $\equiv_{campo_i}$ como:

$$r_1 \equiv_{campo_i} r_2 =_{def} \forall j, j <> i \to r_1.campo_j = r_2.campo_j \tag{2.2}$$

cuando se quiera decir que se mantienen todos los campos iguales entre r_1 y r_2, excepto el $campo_i$.

Por otra parte, se asumen como tipos inductivos paramétricos predefinidos los siguientes: $optionT$ con los constructores $None : option\ T$ y $Some : T \to option\ T$; y $seq\ T$, que denota las secuencias finitas, con [] la secuencia vacía y $ss^\frown s$ la secuencia que incorpora un elemento s al final de una secuencia ss. La concatenación de secuencias se denota con $\oplus$.

Una relación inductiva I se define a través de reglas de introducción de la forma:

$$\frac{P_1\ ...\ P_n}{I\ x_1\ ...\ x_m}(nombre\_regla) \tag{2.3}$$

donde las variables que ocurren libres se asumen cuantificadas universalmente, e I puede usarse en notación infija (en el caso de relaciones binarias).

Sea P un predicado sobre un tipo T, se define el predicado inductivo $all\ P$ sobre $seq\ T$, que establece que P se verifica para todos los elementos de una secuencia, como sigue:

$$\frac{}{all\ P[]}(all\_[]) \qquad \frac{all\ P ss \quad P\ s}{all\ P\ (ss^\frown s)}(all\_^\frown) \tag{2.4}$$

2.2. Políticas de Seguridad y Aplicaciones

Las denominadas *suites* son paquetes en los que suelen agruparse las aplicaciones MIDP (más conocidas como MIDlets) para su distribución. Una característica de las suites es que pueden

agrupar una o varias MIDlets. Además, son distribuidas en dos archivos: el archivo propio de la aplicación, con sus directorios, clases java, ficheros de imagen, etc., y un archivo descriptor.

Un tipo registro *Suite* es una representación de una aplicación instalada, con tres campos básicos: uno para su identificador, otro para su dominio de protección asociado y el último para su descriptor. Cabe destacar que se denota al conjunto de dominios de protección con el tipo *Domain* y al conjunto de los identificadores válidos para MIDlet Suites con el tipo $SuiteID$. La definición de Suite es la siguiente:

$$Suite =_{def} \{id : SuiteID, domain : Domain, descriptor : Descriptor\} \tag{2.5}$$

Además, se declaran los siguientes tipos y predicados para poder representar luego el descriptor de la aplicación:

- *Permission:* es el tipo de dato que representa al conjunto de los permisos definidos por cada dominio de protección para cada aplicación o función protegida del dispositivo.
- *required:* es el predicado que declara los permisos requeridos por una suite para su normal funcionamiento.
- *optional:* es el predicado que declara los permisos opcionales por una suite para su normal funcionamiento.
- *Vendor:* es el tipo de dato que representa los nombres de vendedores de aplicaciones.
- *Certificate:* es el tipo de dato que representa el conjunto de los certificados de clave pública.
- *Signature:* es el tipo de dato que representa la firma del archivo de la aplicación.
- *domainAuthorization:* es el predicado que declara la autorización de acceso por dominio de protección
- *signerAuthorization:* es el predicado que establece la autorización de acceso a un conjunto de suites firmadas con un certificado particular.
- *vendorUnsignedAuthorization:* es el predicado que denota la autorización de acceso a un conjunto de suites que no están firmadas por parte de un vendedor.

- *vendorSignedAuthorization:* es el predicado que declara la autorización de acceso a un conjunto de suites firmadas con un certificado específico de un vendedor.

Dadas estas definiciones, se procede a mostrar la forma del descriptor:

$$
\begin{aligned}
Descriptor =_{def} \{ & \\
& required, optional : Permission \rightarrow Prop, \\
& vendor : Vendor, \\
& jarSignature : option\ Signature, \\
& signerCertificate : option\ Certificate, \\
& domainAuthorization : Domain \rightarrow Prop, \\
& signerAuthorization : Certificate \rightarrow Prop, \\
& vendorUnsignedAuthorization : Vendor \rightarrow Prop, \\
& vendorSignedAuthorization : Vendor \rightarrow Certificate \rightarrow Prop\}
\end{aligned}
\tag{2.6}
$$

Por otra parte, la ***Política de Seguridad*** define para cada dominio de protección los permisos que pueden ser otorgados a las suites por el usuario en una de tres modalidades, así como también los permisos que son otorgados a aquellas suites ligadas a ese dominio en particular. Debido a que existe la posibilidad que el usuario interactue en cuanto a los permisos, se define un conjunto enumerado denominado $Mode$ que denota dichas tres modalidades: {*oneshot, session, blanket*}, donde *oneshot* determina que el permiso es válido para ese único uso; *session* determina que el permiso es válido durante el tiempo que esté ejecutándose la aplicación; y *blanket* determina que el permiso otorgado tiene validez hasta que la aplicación sea desinstalada. Expuesto esto, se define la Política de Seguridad como una constante *Policy* de la forma:

$$
\begin{aligned}
Policy =_{def} \{ & \\
& allow : Domain \rightarrow Permission \rightarrow Prop, \\
& user : Domain \rightarrow Permission \rightarrow Mode \rightarrow Prop\}
\end{aligned}
\tag{2.7}
$$

Finalmente, los permisos efectivamente otorgados a una suite resultan de la siguiente definición:

$$EPG =_{def} Descriptor.required \cap (Policy.allow \cup Policy.user), \tag{2.8}$$

es decir, resultan de la intersección de los permisos requeridos en su descriptor con la unión de los permisos otorgados incondicionalmente por su dominio de protección y los permisos otorgados explícitamente por el usuario.

2.3. Estado del Dispositivo

Para representar el estado del dispositivo es necesario tener en cuenta ciertos elementos indispensables:

- *suite:* es el predicado que representa el conjunto de suites instaladas.
- *SessionInfo:* es el registro que se encarga de representar a la suite activa, junto con los permisos otorgados o revocados a ella en la sesión.
- *session:* es el predicado que modela la suite activa, en caso de existir alguna.
- *granted:* es el predicado que representa los permisos otorgados a cada una de las suites instaladas.
- *revoked:* es el predicado que representa los permisos revocados a cada una de las suites instaladas.
- *authorized:* es el predicado que modela las autorizaciones concedidas a cada una de las suites instaladas.
- *unauthorized:* es el predicado que modela las autorizaciones rechazadas a cada una de las suites instaladas.

Es necesario, antes de dar la definición del estado, mostrar cómo está modelado el registro que representa la información de la suite activa, denominado $SessionInfo$:

$$\begin{aligned} SessionInfo =_{def} \{ & \\ & id : SuiteID, \\ & granted, revoked : Permission \rightarrow Prop\}, \end{aligned} \tag{2.9}$$

donde *granted* y *revoked*, en este caso, representan los permisos otorgados y revocados a la suite activa.

A continuación se muestra la definición del estado del dispositivo, representado por el registro *State*:

$$\begin{aligned} State =_{def} \{ & \\ & suite : SuiteID \to Prop, \\ & session : optionSessionInfo, \\ & granted, revoked : SuiteID \to Permission \to Prop, \\ & authorized, unauthorized : SuiteID \to SuiteID \to Prop\} \end{aligned} \tag{2.10}$$

Para que una suite pueda instalarse, los permisos requeridos en su descriptor deben ser un subconjunto de los permisos ofrecidos por el dominio de protección. Por ende, a continuación se especifica la relación de compatibilidad entre el descriptor *des* y el dominio de protección *dom*, representada por $\wr\wr$:

$$\begin{aligned} & des \wr\wr dom =_{def} \forall p : Permission, \\ & \quad des\ required\ p \to allow\ dom\ p \vee \exists m : Mode, user\ dom\ p\ m \end{aligned} \tag{2.11}$$

Para definir la validez del estado del dispositivo, es necesario establecer las condiciones que deben cumplirse, ya que no toda combinación de valores de un registro de tipo *State* es válida. Éstas son:

◊ *SuiteCompatible:* En un estado s, toda suite instalada debe ser compatible con su dominio de protección asociado. Es decir:

$$(\forall ms : Suite,\ s.suite\ ms \to ms.descriptor \wr\wr ms.domain) \tag{2.12}$$

◊ *UniqueSuiteID:* Los identificadores de las suites instaladas son únicos. Esto se define como:

$$\begin{aligned} & (\forall ms_1\ ms_2 : Suite, \\ & \quad s.suite\ ms_1 \wedge s.suite\ ms_2 \to ms_1.id\ <>\ ms_2.id) \end{aligned} \tag{2.13}$$

◊ *CurrentInstalled:* El identificador de la suite en la sesión actual corresponde al de una suite instalada. A continuación, la definición:

$$\begin{aligned}(\forall ses : SessionInfo,\ s.session\ =\ Some\ ses \rightarrow \\ \exists ms : Suite,\ s.suite\ ms \wedge ms.id\ =\ ses.id)\end{aligned} \tag{2.14}$$

◊ *ValidSessionGranted:* El conjunto de permisos otorgados por la sesión debe ser un subconjunto de los permisos declarados en el descriptor de la suite activa, y el dominio de protección asociado debe permitir otorgarlos en al menos modo *session*. A saber:

$$\begin{aligned}&(\forall ses : SessionInfo,\ s.session\ =\ Some\ ses \rightarrow \\ &\forall p : Permission,\ ses.granted\ p \rightarrow \\ &\forall ms : Suite,\ s.suite\ ms \rightarrow ms.id\ =\ ses.id \rightarrow \\ &(ms.descriptor.required\ p \vee ms.descriptor.optional\ p) \wedge \\ &(user\ ms.domain\ p\ session \vee user\ ms.domain\ p\ blanket))\end{aligned} \tag{2.15}$$

◊ *ValidGranted:* Los permisos otorgados a cada suite instalada por el resto de su tiempo de vida, deben ser un subconjunto de los permisos declarados en su descriptor, y la política de seguridad del dominio de protección al que está asociada debe permitir otorgarlos en modo *blanket*. Esto es:

$$\begin{aligned}&(\forall\ (p : Permission)(sid : SuiteID),\ s.granted\ sid\ p \rightarrow \\ &\forall ms : Suite,\ s.suite\ ms \rightarrow ms.id\ =\ sid \rightarrow \\ &(ms.descriptor.required\ p \vee ms.descriptor.optional\ p) \wedge \\ &user\ ms.domain\ p\ blanket)\end{aligned} \tag{2.16}$$

◊ *ValidAuthorization:* Una suite no puede estar autorizada y desautorizada al mismo tiempo por otra suite. En otras palabras:

$$\begin{aligned}&(\forall\ idGrn\ idReq : SuiteID, \\ &s.authorized\ idGrn\ idReq \rightarrow \neg s.unauthorized\ idGrn\ idReq)\end{aligned} \tag{2.17}$$

Finalmente, definimos la validez del estado del dispositivo como la conjunción de todas condiciones recién definidas. Definimos entonces el predicado $StValid$ como:

$$
\begin{aligned}
StValid =_{def}\ & SuiteCompatible \wedge \\
& UniqueSuiteID \wedge \\
& CurrentInstalled \wedge \\
& ValidSessionGranted \wedge \\
& ValidGranted \wedge \\
& ValidAuthorization
\end{aligned}
\tag{2.18}
$$

2.4. Eventos

Los eventos relacionados con la seguridad del dispositivo se definen como un tipo inductivo no recursivo *Event*, donde cada constructor corresponde a una clase de evento diferente, tal como se muestra en la siguiente tabla:

Nombre	**Tipo**	**Descripción**
start	$SuiteID \rightarrow Event$	Comienzo de sesión
terminate	$Event$	Fin de sesión
request	$Permission \rightarrow option\, UserAnswer \rightarrow Event$	Solicitud de permiso por parte de la suite activa
install	$SuiteID \rightarrow Descriptor \rightarrow Domain \rightarrow Event$	Instalación de una suite
remove	$SuiteID \rightarrow Event$	Eliminación de una suite
authorization	$SuiteID \rightarrow Event$	Solicitud de autorización de acceso por parte de una suite a los recursos compartidos de la suite activa

Cuadro 2.1: Constructores del tipo *Event*

Vale la pena destacar que para definir el evento *request*, al tener el usuario la posibilidad de interactuar y determinar una respuesta sobre si da permiso o no, y la duración del mismo, es necesario brindar una representación para dichas interacciones. El tipo de las posibles respuestas

se representa como un tipo inductivo *UserAnswer : Set*, con dos constructores de la forma:

$$\begin{aligned} ua\_allow &: Mode \to UserAnswer \\ ua\_deny &: Mode \to UserAnswer \end{aligned} \tag{2.19}$$

2.5. Comportamiento

El comportamiento de los eventos se especifica mediante la definición de sus *pre* y sus *poscondiciones*. Las precondiciones se definen en términos del estado del dispositivo, mientras que las poscondiciones, en términos del estado anterior, posterior, y una respuesta opcional, que es significativa solamente para el evento *request* (indica si la solicitud de permiso es aceptada o rechazada). Esta respuesta se representa como un tipo enumerado de la forma:

$$Response \ : \ Set \ := \ allowed \mid denied, \tag{2.20}$$

donde *allowed* denota la aceptación de la solicitud y *denied* su rechazo. La forma de especificar las pre y poscondiciones es a través de los predicados Pre y Pos, respectivamente, cuya representación es:

$$\begin{aligned} Pre &: State \to Event \to Prop \\ Pos &: State \to State \to option\ Response \to Event \to Prop \end{aligned} \tag{2.21}$$

El estado permanece inalterable cuando la precondición de un evento no se satisface. Por otra parte, el estado cambia cuando la poscondición del evento se establece y su precondición se cumple. Estas dos posibles situaciones se definen como *relación de ejecución en un paso* y se denota con el símbolo ($\Rightarrow$). Ambos casos se definen con las siguientes reglas de introducción:

$$\frac{\neg \ Pre\ s\ e}{s \ \Rightarrow^{e/None} \ s} \quad (npre) \qquad \frac{Pre\ s\ e \quad Pos\ s\ s'\ r\ e}{s \ \Rightarrow^{e/r} \ s'} \quad (pre), \tag{2.22}$$

donde $s \ \Rightarrow^{e/r} \ s'$ se lee como *"a partir de* $\boldsymbol{s}$*, la ejecución del evento* $\boldsymbol{e}$ *da lugar al estado* $\boldsymbol{s'}$ *y a la respuesta* $\boldsymbol{r}$*"*

La relación de ejecución en un paso nos lleva a formalizar el concepto de *sesión de ejecución*, debido a que por sí sola, la relación no permite verificar ciertas propiedades del modelo de

seguridad que dependen de la ejecución de una serie de eventos durante una sesión y no sólo de la ejecución de un evento en particular. Por tal motivo, decimos que una *sesión* para una suite con identificador id queda determinada por un estado inicial s_0, y una secuencia de pasos $(< e_i, s_i, r_i >, i = 1, ..., n)$ que satisfacen:

- $e_1 = start\ id$
- $Pre\ s_0\ e_1$
- $\forall i : 2\ \leq\ i\ \leq\ n-1,\ e_i <> terminate$
- $e_n = terminate$
- $\forall i : 1\ \leq\ i\ \leq\ n,\ s_{i-1} \Rightarrow^{e_i/r_i}\ s_i$
- $StepSession =_{def} \{e : Event,\ s : State,\ r : Response\}$

2.6. Propiedades de Seguridad

La validez del estado del dispositivo definida en (2.18) es una propiedad que se demuestra invariante con respecto a:

- Una sesión de ejecución
- La ejecución en un paso de cualquier evento

El cumplimiento de varias propiedades deseables del modelo de seguridad fue demostrado a partir de la invariancia de $StValid$, las cuales establecen que:

- Las aplicaciones instaladas disponen de permisos compatibles con los del dominio de protección al que están ligadas
- La aplicación activa es una aplicación instalada
- Los permisos otorgados en una sesión a una suite son comptatibles con el dominio de protección y con los permisos declarados
- Los permisos otorgados permanentemente a una suite son compatibles con el dominio de protección y con los permisos declarados

- Ninguna suite puede estar autorizada y, a la vez, desautorizada

Una propiedad invariante en una sesión de ejecución es la desautorización de una aplicación. A partir de este resultado se demuestra que si una suite ha sido desautorizada por el resto de una sesión, cualquier solicitud posterior de autorización de acceso de la misma suite seguirá siendo rechazada.

Capítulo 3

Formalización de Certificados e Instalación de Aplicaciones en MIDP 3.0

En este capítulo se extiende la formalización descripta en el capítulo anterior con respecto a la instalación de aplicaciones en MIDP 3.0. En los trabajos anteriores, las aplicaciones eran instaladas sin importar su procedencia. No se verificaba su vendedor, si tenía o no un certificado asociado, o si este estaba vencido, por ejemplo. Esto dejaba la puerta abierta a cualquier aplicación maliciosa.

Para subsanar este problema, en este trabajo se extiende la definición del evento *install*, se definen y describen nuevos tipos de datos, como así también se reestructuran otros ya definidos. Se crea una especie de repositorio de vendedores y certificados para realizar comprobaciones a la hora de la instalación. Se tienen en cuenta tanto el caso en que un vendedor posea un certificado como el caso en que no, ya que un usuario tiene la posibilidad de programar una aplicación para uso propio y por carecer de una certificación, no implica que no pueda instalarla ya que conoce la procedencia de la misma.

Con el fin de que estas nuevas características sean válidas, se definen dos nuevas condiciones de validez de estado que deben cumplirse y se prueba la preservación de las propiedades demostradas en trabajos anteriores y se analizan otras nuevas relacionadas con esta extensión.

3.1. Reestructuración y Formalización

Se extiende la definición de *certificado* a un concepto un poco más abarcativo. En su primera versión consistía en un simple tipo de dato, de lo cual no se podía saber ninguna característica. Es por ello que se lo extendió a un registro de la forma:

$$\begin{aligned} Certificate =_{def} \{ & \\ & certificate\_serial : nat, \\ & certificate\_vendor : Vendor, \\ & certificate\_signature : Signature, \\ & certificate\_expirationDate : Date\}, \end{aligned} \tag{3.1}$$

donde:

. **certificate_serial** es el número de serie que identifica al certificado,

. **certificate_vendor** es el nombre del vendedor o dueño de la aplicación,

. **certificate_signature** es la firma del propietario y

. **certificate_expirationDate** es la fecha de vencimiento del certificado.

Además, se crea un nuevo tipo de dato en forma de registro que realiza la función de almacenar los datos correspondientes a un vendedor y su certificado asociado. Esta nueva definición se extiende de manera que se puedan tener en cuenta los vendedores de aplicaciones que no posean certificados. Para esto, se define un registro $VendorRepository$ como:

$$\begin{aligned} VendorRepository =_{def} \{ & \\ & vrep\_vendor : Vendor, \\ & vrep\_certificate : option\ Certificate\} \end{aligned} \tag{3.2}$$

Por último, se declara un nuevo tipo de dato denominado *Date*, el cual representa la fecha.

3.2. Nuevo Estado del Dispositivo

Es necesario representar el grupo de vendedores y posibles certificados presentes en el dispositivo. Por tal motivo, se extiende el estado del mismo para tener en cuenta esta nueva extensión con el predicado *vendorRepository*.

$$
\begin{aligned}
&State =_{def} \{ \\
&\quad suite : SuiteID \rightarrow Prop, \\
&\quad session : optionSessionInfo, \\
&\quad granted, revoked : SuiteID \rightarrow Permission \rightarrow Prop, \\
&\quad authorized, unauthorized : SuiteID \rightarrow SuiteID \rightarrow Prop \\
&\quad vendorRepository : VendorRepository \rightarrow Prop\}
\end{aligned}
\tag{3.3}
$$

Debido a este nuevo predicado, se agregan dos nuevas condiciones de validez sobre el estado del dispositivo: una con respecto a los vendedores y su cantidad de certificados y otra sobre la cantidad de firmas iguales entre certificados. Con el fin de obtener mayor claridad, se detallan cada una:

◊ *OneVendorTwoCertificates:* Un mismo vendedor no puede tener dos certificados sin vencer. Es decir:

$$
\begin{aligned}
&(\forall(c1\ c2 : Certificate)(CurrentDate : Date), \\
&\quad certificate\_expirationDate\ c1 \geq CurrentDate \wedge \\
&\quad certificate\_expirationDate\ c2 \geq CurrentDate \wedge \\
&\quad certificate\_vendor\ c1\ =\ certificate\_vendor\ c2 \rightarrow c1\ =\ c2)
\end{aligned}
\tag{3.4}
$$

◊ *OneSignatureTwoCertificates:* Una misma firma no puede aparecer en dos certificados diferentes sin vencer. A continuación, la definición:

$$
\begin{aligned}
&(\forall(c1\ c2 : Certificate)(CurrentDate : Date), \\
&\quad certificate\_expirationDate\ c1 \geq CurrentDate \wedge \\
&\quad certificate\_expirationDate\ c2 \geq CurrentDate \wedge \\
&\quad certificate\_signature\ c1\ =\ certificate\_signature\ c2 \rightarrow c1\ =\ c2)
\end{aligned}
\tag{3.5}
$$

3.3. Instalación

Aquí se realiza un cambio sobre el evento *install* ya definido para que se adapte a los nuevos requerimientos a la hora de realizar una instalación. En representaciones anteriores [4], tenía la forma:

$$install : SuiteID \rightarrow Descriptor \rightarrow Domain \rightarrow Event \tag{3.6}$$

Debido a que necesitamos hacer verificaciones en los certificados en cuanto a sus fechas de caducidad, el elemento que representa la fecha actual es fundamental en la nueva definición de *install*.

$$install : SuiteID \rightarrow Descriptor \rightarrow Domain \rightarrow Date \rightarrow Event \tag{3.7}$$

Este evento tiene como precondición que el dominio de protección de la MIDlet a instalar sea compatible con su descriptor y que su identificador no coincida con el de alguna suite ya instalada.

$$\begin{aligned} &Pre\ s\ (install\ sid\ des\ dom\ cdate)\ =_{def} \\ &des \wr\wr dom\ \wedge\ \forall ms : Suite,\ s.suite\ ms \rightarrow ms.id\ <>\ sid \end{aligned} \tag{3.8}$$

Por otro lado, este mismo evento tiene seis posibles poscondiciones a causa de la extensión planteada. Con el fin de ser lo más claro posible, dichas poscondiciones han sido resumidas a sólo sus características más relevantes con la intención de no confundir al lector. Las definiciones en sus formas completas pueden consultarse en [11] o [12]. A continuación se brindan las poscondiciones mencionadas anteriormente:

1. Si el vendedor de la aplicación no aparece en el repositorio del dispositivo y no posee un certificado asociado, se agrega el nuevo vendedor y se instala la aplicación, por lo que el estado del dispositivo cambia. A su vez, esta nueva aplicación no tendrá permisos otorgados ni revocados, como así tampoco autorizaciones concedidas o rechazadas. Por otro lado, los

demás elementos del estado del dispositivo deben permanecer sin cambios.

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\forall vr : VendorRepository,\ s.vendorRepository\ vr \rightarrow\\
&\qquad vr.vrep\_vendor\ <>\ des.vendor) \rightarrow\\
&(\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms \vee\\
&\qquad ms\ =< sid,\ dom,\ des >_{Suite}) \wedge\\
&s'.suite\ < sid,\ dom,\ des >_{Suite}\ \wedge\\
&s'.granted\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.revoked\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.authorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.unauthorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&(\forall vr : VendorRepository,\ s'.vendorRepository\ vr \rightarrow\\
&\qquad s.vendorRepository\ vr \vee\\
&\qquad vr\ =< des.vendor,\ des.signerCertificate >_{VendorRepository}) \wedge\\
&s'.vendorRepository < des.vendor,\ des.signerCertificate >_{VendorRepository})
\end{aligned}
\tag{3.9}
$$

2. Si el vendedor de la aplicación no existe en el repositorio y posee un certificado asociado sin vencer, se agrega el nuevo vendedor junto con su certificado y se instala la aplicación, la cual no poseerá ni autorizaciones, ni permisos otorgados o revocados, y se mantendrán

inalterados los demás elementos del estado del dispositivo.

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\forall(vr : VendorRepository)(cert : Certificate),\ s.vendorRepository\ vr \rightarrow\\
&\qquad\qquad vr.vrep\_vendor\ <>\ des.vendor \wedge\\
&\qquad\qquad des.signerCertificate\ =\ Some\ cert \wedge\\
&\qquad\qquad cert.expirationDate\ \geq\ cdate) \rightarrow\\
&(\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms \vee\\
&\qquad\qquad ms\ =< sid,\ dom,\ des >_{Suite}) \wedge\\
&s'.suite\ < sid,\ dom,\ des >_{Suite}\ \wedge\\
&s'.granted\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.revoked\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.authorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.unauthorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&(\forall vr : VendorRepository,\ s'.vendorRepository\ vr \rightarrow\\
&\qquad\qquad s.vendorRepository\ vr \vee\\
&\qquad\qquad vr\ =< des.vendor,\ des.signerCertificate >_{VendorRepository}) \wedge\\
&s'.vendorRepository < des.vendor,\ des.signerCertificate >_{VendorRepository})
\end{aligned}
\tag{3.10}
$$

3. Si el vendedor de la aplicación existe en el repositorio y la suite no posee un certificado, únicamente se instala la aplicación, la cual no tendrá ni permisos, ni autorizaciones concedidas o rechazadas, y se mantendrán intalterados los demás elementos del estado del

dispositivo.

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\exists vr : VendorRepository,\ s.vendorRepository\ vr \rightarrow\\
&vr.vrep\_vendor\ =\ des.vendor \wedge\\
&des.signerCertificate\ =\ None \wedge\\
&(\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms \vee\\
&\qquad\qquad ms\ =< sid,\ dom,\ des >_{Suite}) \wedge\\
&s'.suite\ < sid,\ dom,\ des >_{Suite}\ \wedge\\
&s'.granted\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.revoked\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.authorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.unauthorized\ sid\ =\ (\lambda x.\ False)
\end{aligned}
\tag{3.11}
$$

4. Si el vendedor de la aplicación existe en el repositorio, la suite tiene un certificado sin vencer y es el mismo que está instalado, únicamente se instala la aplicación, debiéndose

cumplir las mismas proposiciones mencionadas en la poscondición anterior.

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\exists vr : VendorRepository,\ \forall cert : Certificate,\ s.vendorRepository\ vr \rightarrow\\
&vr.vrep\_vendor\ =\ des.vendor \wedge\\
&des.signerCertificate\ =\ Some\ cert \wedge\\
&vr.vrep\_certificate\ =\ des.signerCertificate \wedge\\
&cert.expirationDate\ \geq\ cdate \wedge\\
&(\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms \vee\\
&\qquad\qquad ms\ =< sid,\ dom,\ des >_{Suite}) \wedge\\
&s'.suite\ < sid,\ dom,\ des >_{Suite}\ \wedge\\
&s'.granted\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.revoked\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.authorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&s'.unauthorized\ sid\ =\ (\lambda x.\ False)
\end{aligned}
\tag{3.12}
$$

5. Si el vendedor de la aplicación existe en el repositorio, la suite tiene un certificado sin vencer y no es el mismo que está instalado, se actualiza ese vendedor con el nuevo certificado y se instala la aplicación, manteniendo inalterados los demás elementos del estado del

dispositivo y con la nueva API sin autorizaciones ni permisos otorgados y/o revocados.

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\exists(vr : VendorRepository),\ \forall(cert : Certificate),\ s.vendorRepository\ vr \rightarrow\\
&\ vr.vrep\_vendor\ =\ des.vendor \wedge\\
&\ des.signerCertificate\ =\ Some\ cert \wedge\\
&\ cert.expirationDate\ \geq\ cdate \wedge\\
&\ vr.vrep\_certificate\ <>\ des.signerCertificate \wedge\\
&\ (\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms \vee\\
&\qquad\qquad ms\ =< sid,\ dom,\ des >_{Suite}) \wedge\\
&\ s'.suite\ < sid,\ dom,\ des >_{Suite}\ \wedge\\
&\ s'.granted\ sid\ =\ (\lambda x.\ False) \wedge\\
&\ s'.revoked\ sid\ =\ (\lambda x.\ False) \wedge\\
&\ s'.authorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&\ s'.unauthorized\ sid\ =\ (\lambda x.\ False) \wedge\\
&\ (\forall vr' : VendorRepository,\ vr\ <>\ vr' \rightarrow\ s.vendorRepository\ vr' \rightarrow\\
&\qquad\qquad s'.vendorRepository\ vr') \wedge\\
&\ (s'.vendorRepository\ vr \rightarrow\\
&\qquad vr\ =< des.vendor,\ des.signerCertificate >_{VendorRepository}) \wedge\\
&\ s'.vendorRepository\ < des.vendor,\ des.signerCertificate >_{VendorRepository}\ \wedge\\
&\ (\forall vr' : VendorRepository,\ vr\ <>\ vr' \rightarrow\\
&\qquad\qquad s'.vendorRepository\ vr' \rightarrow s.vendorRepository\ vr')
\end{aligned}
\tag{3.13}
$$

6. Si el certificado de la aplicación que se quiere instalar está vencido, se rechaza la instalación y se mantiene el estado del dispositivo sin cambios (esta poscondición se brinda completa

por ser muy breve).

$$
\begin{aligned}
&Pos\ s\ s'\ r\ (install\ sid\ des\ dom\ cdate)\ =_{def}\\
&(\forall cert : Certificate,\ des.signerCertificate\ =\ Some\ cert \wedge\\
&\qquad\qquad cert.expirationDate\ <\ cdate\ \wedge\\
&r\ =\ None\ \wedge\\
&(\forall ms : Suite,\ s.suite\ ms \rightarrow s'.suite\ ms)\ \wedge\\
&(\forall ms : Suite,\ s'.suite\ ms \rightarrow s.suite\ ms)\ \wedge\\
&s'.session\ =\ s.session\ \wedge\\
&(\forall sid_1 : SuiteID,\ sid_1\ <>\ sid \rightarrow\\
&\qquad s'.granted\ sid_1\ =\ s.granted\ sid_1\ \wedge\\
&\qquad s'.revoked\ sid_1\ =\ s.revoked\ sid_1\ \wedge\\
&\qquad s'.authorized\ sid_1\ =\ s.authorized\ sid_1\ \wedge\\
&\qquad s'.unauthorized\ sid_1\ =\ s.unauthorized\ sid_1)\ \wedge\\
&(\forall vr : VendorRepository,\ s.vendorRepository\ vr \rightarrow\\
&\qquad\qquad s'.vendorRepository\ vr)\ \wedge\\
&(\forall vr : VendorRepository,\ s'.vendorRepository\ vr \rightarrow\\
&\qquad s.vendorRepository\ vr))
\end{aligned}
\tag{3.14}
$$

3.4. Conservación de las Propiedades de Seguridad

La formalización propuesta en este trabajo introduce varios cambios sustanciales. En primer lugar, modifica el evento **install** para que tome la fecha actual. En segundo lugar, se amplía el estado del dispositivo para poder llevar un registro o repositorio de los vendedores y certificados de los cuales se ha instalado alguna aplicación. Tercero, se definen nuevas condiciones que se deben exigir a la hora de llevar dicho repositorio.

Todos estos cambios conllevan a que los invariantes ya formalizados y demostrados se vean alterados. Es por esto que en este documento se demuestra que a pesar de todas estas modificaciones, la validez del estado se mantiene invariante tras la ejecución de cualquier evento, incluyendo el modificado.

Por una cuestión práctica, a continuación se da una demostración informal de las propiedades enunciadas con anterioridad. La versión completa, formalizada y verificada fue realizada con el asistente de pruebas Coq.

3.4.1. Invariancia de la Validez del Estado

A continuación se brinda un teorema que demuestra que el estado resultante de la ejecución en un paso de cualquier evento, a partir de un estado válido inicial, seguirá siendo válido. Para su demostración se adoptará la estrategia planteada en [4].

TEOREMA 1: Sea $Valid$ el predicado sobre *State* definido como la conjunción de los predicados $StValid$, *OneVendorTwoCertificates* y *OneSignatureTwoCertificates*. Para todo *s s' : State*, *r : Response* y *e : Event*, si se cumplen *Valid s* y $s \Rightarrow^{e/r} s'$, entonces también se cumple *Valid s'*.

Demostración:

La demostración se realiza haciendo análisis de casos sobre $s \Rightarrow^{e/r} s'$.

1. Se utiliza la regla *npre* cuando la precondición, *Pre s e*, no se verifica, en cuyo caso se mantiene que *s = s'*. Partiendo de esta igualdad y de *Valid s*, se obtiene que vale *Valid s'*.

2. Para el caso restante, la regla de introducción *pre* establece *Pos s s' r e*.

En esta instancia de la demostración se vuelve a hacer análisis de casos pero sobre **e**. De manera de mantener claridad y proporcionar un pantallazo de la demostración, sólo se analizará el evento *install*. El comportamiento de este evento se realiza en función de sus seis poscondiciones. Debido a que la demostración es muy extensa, ya que se debe probar que para cada una de las poscondiciones, se cumplen los seis términos de la conjunción en *StValid* y las dos nuevas condiciones sobre el estado definidas con anterioridad, se mostrarán sólo dos casos.

- La aplicación a instalar posee un certificado pero se encuentra vencido (3.15).

  [.] Al no instalarse la suite, en *s'* se mantienen todas las suites de *s*, por lo que se demuestra la compatibilidad y la unicidad de los identificadores. Es decir que valen **SuiteCompatible s'** y **UniqueSuiteID s'**.

[.] El identificador de la sesión activa corresponde a una suite en s y al no instalar la nueva, se verifica **CurrentInstalled s'**, como así también **ValidSessionGranted s'** y **ValidGranted s'**.

[.] Como no se instala la suite, en *s'* se mantienen las autorizaciones y desautorizaciones de s, por lo que se demuestra **ValidAuthorization**.

[.] Al cumplirse los seis miembros de **StValid**, se verifica que vale **StValid s'**.

El certificado está expirado, por lo que no se realizan cambios sobre el repositorio y se preservan todas las características del mismo; es decir que valen **OneVendorTwoCertificates s'** y **OneSignatureTwoCertificates s'**.

[.] Al cumplirse todos los elementos de la conjunción, entonces **Valid s'** se verifica cuando se quiere instalar una aplicación y su certificado está expirado.

- La aplicación tiene un vendedor que no se encuentra en el repositorio y no posee un certificado asociado (3.10).

  [.] Al instalar la nueva suite, se desprende de la precondición que su descriptor es compatible con su dominio de protección y como el resto de las suites se mantienen, queda probado **SuiteCompatible s'**.

  [.] Las suites instaladas en *s'* son exactamente las suites instaladas en s agregando la nueva aplicación, por lo que se mantiene la unicidad de identificadores y se prueba **UniqueSuiteID s'**.

  [.] De esta poscondición se desprende que $s'.session = s.session$, por lo que el identificador de la sesión activa corresponde a una suite instalada en s. Al estar s, en *s'*, el identificador de la sesión continua correspondiendo a una suite instalada y se verifica **CurrentInstalled s'**.

  [.] Se verifica **ValidSessionGranted s'** a partir de la validez de s y de la observación de que la suite activa es una suite instalada en s.

  [.] Sabemos que el identificador de la nueva MIDlet, ms, es distinto a los identificadores de las demás MIDlets instaladas y por la poscondición, se sabe que $s'.granted\ ms.id = (\lambda x.\ False)$. Como todas las suites en *s'* tienen que estar instaladas y tienen que tener un

identificador distinto del de *ms*, deben estar instaladas en *s*. A partir de **ValidSessionGranted s**, el resultado anterior y el hecho de que *s'.granted = s.granted* para todas las suites instaladas sin tener en cuenta la nueva, se determina que vale **ValidGranted s'**.

[.] La nueva suite no posee autorizaciones ni desautorizaciones según la poscondición, y como para toda MIDlet ya instalada se mantiene que *s'.authorized = s.authorized* y *s'.unauthorized = s.unauthorized*, se sigue que vale **ValidAuthorization s'**.

[.] Como el vendedor no existe en el repositorio y no posee un certificado asociado, y a partir que se cumple **OneVendorTwoCertificates s**, se verifica **OneVendorTwoCertificate s'**.

[.] Como el vendedor no existe en el repositorio y todos los demás elementos del mismo se conservan, se verifica trivialmente que vale **OneSignatureTwoCertificates**.

[.] Al cumplirse todos los elementos de la conjunción, entonces **Valid s'** se verifica cuando se quiere instalar una aplicación, cuyo vendedor no se encuentra en el repositorio de vendedores y certificados y, además, no posee un certificado asociado.

QED.

A partir del siguiente teorema, se extiende que la invariancia de la validez del estado se mantiene con respecto a una sesión de ejecución. El **teorema 2** fue enunciado primeramente en [16] y extendido en [4]. Aquí se lo vuelve a extender para abarcar los avances propuestos por este trabajo.

TEOREMA 2: Sea *ss* una sesión de ejecución a partir de un estado inicial válido. Todos los estados de *ss* son válidos: $all(\lambda(step : StepSession).Valid\ step.s)ss$.

La demostración se realiza por inducción estructural en la definición de sesión de ejecución y es esencialmente igual a la desarrollada en [16] y [4].

Capítulo 4

Un Algoritmo Certificado para la Instalación

Con el fin de obtener un prototipo ejecutable, en este capítulo se construye un refinamiento del modelo de seguridad de MIDP. En la concretización planteada, se propone un algoritmo que represente la instalación de aplicaciones para MIDP 3.0. La demostración de que este algoritmo es correcto, se formaliza a través de un teorema de consistencia (soundness).

Algunos de los conceptos planteados en este capítulo fueron presentados en [4] y aquí son nuevamente citados debido a que son parte de las bases de este trabajo.

4.1. Concretización de los elementos

Hasta el momento, todo lo que hemos visto en capítulos anteriores posee un alto nivel de abstracción. Las propiedades del modelo de seguridad fueron expresadas en el *Cálculo de Construcciones Inductivas*, por lo cual, fueron expresadas como elementos de tipo *Prop*. Con el fin de brindar mayor claridad, decimos que cualquier objeto de tipo P es una prueba de la verdad de P, o dicho en otras palabras, si P tiene tipo *Prop* (P : Prop), entonces P es una propiedad lógica. Esto nos lleva a concluir que para cualquier propiedad que escribamos, su demostración se obtiene al construir un término del tipo de dicha propiedad. Es aquí donde el asistente de pruebas *Coq* pasa a ser una herramienta primordial, ya que no sólo provee de tácticas y de una biblioteca de teorías que simplifican ese trabajo, sino que también proporciona mecanismos

interactivos para la construcción de dichos términos.

Otra ventaja que nos brinda el asistente *Coq* es la posibilidad de, a partir de una especificación, extraer un prototipo ejecutable en un lenguaje funcional. El inconveniente que se nos presenta es que a partir de la representación planteada, existen, entre otras cosas, varios elementos del estado del dispositivo que denotan colecciones o predicados sobre dichos elementos que tienen tipo *Prop*. Esta representación no permite explotar la característica mencionada previamente. Por ende, para aprovechar dicha ventaja, es necesario hacer cambios sobre los predicados planteados, los cuales se deben transformar de *Prop* a *Set*.

Con el fin de no perder consistencia ni generalidad, se asume que todos los predicados utilizados para describir el estado del dispositivo son decidibles y tienen un dominio finito. A continuación, se brinda el criterio que se adopta, extraido de [4], para la transformación de un predicado P decidible y definido sobre un conjunto finito S de tipo *Set*:

- ◊ Si el predicado P caracteriza una colección de elementos, su representación está dada como una lista exhaustiva $list$ de elementos de tipo S que satisfacen dicho predicado.

$$\forall a,\ P\ a \Longleftrightarrow a \in list \tag{4.1}$$

- ◊ Si el predicado P caracteriza una propiedad sobre un conjunto de elementos, su representación está dada como una función F de S en un tipo isomorfo a *bool*.

$$\forall a,\ P\ a \Longleftrightarrow F\ a = true \tag{4.2}$$

El primer criterio es para el caso del conjunto de aplicaciones instaladas, mientras que el segundo es el caso de las aplicaciones instaladas que han tenido autorizaciones o desautorizaciones previas.

4.2. Representación Concreta del Estado del Dispositivo y de los Eventos

De manera de obtener una representación concreta del estado del dispositivo y de los eventos, se aplican los criterios enunciados con anterioridad. Con el fin de marcar una diferencia entre esta nueva representación y su par de alto nivel, los tipos de datos se prefijan con la letra C. A continuación, se enuncian los cambios realizados y se brindan sus respectivas definiciones:

▷ **Descriptor de la Suite:** Los permisos y autorizaciones se representan como perdicados sobre booleanos.

$$
\begin{aligned}
&CDescriptor =_{def} \{ \\
&required, optional : Permission \to bool, \\
&vendor : Vendor, \\
&jarSignature : option\ Signature, \\
&signerCertificate : option\ CCertificate, \\
&domainAuthorization : Domain \to bool, \\
&signerAuthorization : CCertificate \to bool, \\
&vendorUnsignedAuthorization : Vendor \to bool, \\
&vendorSignedAuthorization : Vendor \to CCertificate \to bool\}
\end{aligned}
\tag{4.3}
$$

▷ **Estado del Dispositivo:** El conjunto de aplicaciones instaladas y el repositorio de vendedores y certificados se representan como listas, mientras que los permisos concedidos y revocados, y las autorizaciones y desautorizaciones se representan como predicados sobre booleanos.

$$
\begin{aligned}
&CState =_{def} \{ \\
&suite : list\ CMIDletSuite, \\
&session : option\ CSessionInfo, \\
&granted, revoked : SuiteID \to Permission \to bool, \\
&authorized, unauthorized : SuiteID \to SuiteID \to bool \\
&vendorRepository : list\ CVendorRepository\}
\end{aligned}
\tag{4.4}
$$

▷ **Información de sesión:** Los permisos otorgados y revocados a la suite activa se representan como predicados sobre booleanos.

$$
\begin{aligned}
&CSessionInfo =_{def} \{ \\
&id : SuiteID, \\
&granted, revoked : Permission \to bool\},
\end{aligned}
\tag{4.5}
$$

▷ **Precondición:** Establece que el dominio de protección al que se asociará es compatible con su descriptor y que su identificador no coincide con el de ninguna suite ya instalada.

$$\begin{aligned} &CPre\_install\ cst\ ms.descriptor\ ms.domain\ policy\ ms.id\ =_{def} \\ &\quad Compatible\ policy\ des\ dom \rightarrow \\ &\quad \forall ms : CMIDletSuite,\ isMIDletSuite\ ms.id\ cst.suite \rightarrow ms.id\ <>\ sid \end{aligned} \tag{4.6}$$

▷▷ La definición de ***Compatible***:

$$\begin{aligned} &Compatible\ policy\ des\ dom\ =_{def} \\ &\quad \forall p : Permission,\ des.required\ p\ =\ true- > \\ &\qquad policy.allow\ dom\ p \vee \exists m : CMode,\ policy.user\ dom\ p\ m \end{aligned} \tag{4.7}$$

▷ **Poscondiciones:** Por una cuestión de practicidad, se brinda la representación de la poscondición general como la disyunción de las seis poscondiciones del evento y se detalla sólo una de ellas a nivel de ejemplo.

$$\begin{aligned} &CPos\_install\ =_{def} \\ &CPos\_install\_no\_vendor\_no\_signed\_suite \vee \\ &CPos\_install\_no\_vendor\_signed\_suite\_noexp \vee \\ &CPos\_install\_vendor\_no\_signed\_suite \vee \\ &CPos\_install\_vendor\_signed\_suite\_no\_exp\_installed \vee \\ &CPos\_install\_vendor\_signed\_suite\_no\_exp\_dif\_installed \vee \\ &CPos\_install\_exp \end{aligned} \tag{4.8}$$

▷▷ La poscondición cuando el certificado de la aplicación a instalar está expirado se

detalla a continuación:

$$
\begin{aligned}
&CPos\_install\_exp\ cst\ cst'\ r\ ms\ CurrentDate\ =_{def}\\
&\quad hasCertificate\ ms\ =\ true\ \wedge\\
&\quad isCertExpired\ (getCertificate\ ms)\ CurrentDate\ =\ true\ \wedge\\
&\quad r\ =\ None\ \wedge\\
&\quad (\forall ms : CMIDletSuite,\ isMIDletSuite\ ms.id\ cst.suite \rightarrow\\
&\qquad\qquad isMIDletSuite\ ms.id\ cst'.suite)\ \wedge\\
&\quad (\forall ms : CMIDletSuite,\ isMIDletSuite\ ms.id\ cst'.suite \rightarrow\\
&\qquad\qquad isMIDletSuite\ ms.id\ ms\ cst.suite)\ \wedge\\
&\quad cst'.session\ =\ cst.session\ \wedge\\
&\quad (\forall sid0 : SuiteID,\ isMIDletSuite\ sid0\ cst.suite \rightarrow\\
&\qquad cst'.granted\ sid0\ =\ cst.granted\ sid0\ \wedge\\
&\qquad cst'.revoked\ sid0\ =\ cst.revoked\ sid0\ \wedge\\
&\qquad cst'.authorized\ sid0\ =\ cst.authorized\ sid0\ \wedge\\
&\qquad cst'.unauthorized\ sid0\ =\ cst.unauthorized\ sid0)\ \wedge\\
&\quad (\forall vr : CVendorRepository,\ isVendorRep\ vr\ cst.vendorRepository \rightarrow\\
&\qquad\qquad isVendorRep\ vr\ cst'.vendorRepository)\ \wedge\\
&\quad (\forall vr : CVendorRepository,\ isVendorRep\ vr\ cst'.vendorRepository \rightarrow\\
&\qquad\qquad isVendorRep\ vr\ cst.vendorRepository)
\end{aligned}
\tag{4.9}
$$

4.3. Algoritmo para la Instalación

En la presente sección se propone una implementación del algoritmo encargado de la instalación de aplicaciones. Este algoritmo toma como parámetros la MIDlet Suite a instalar (ms), la fecha actual ($CurrentDate$) y el estado concreto actual (cst), retornando un par formado por un nuevo estado posiblemente modificado y la respuesta a una solicitud que en el caso de la instalación es siempre igual a $None$.

El algoritmo comienza a verificar si la aplicación a instalar posee un certificado o no. Este proceso se divide en dos casos:

I. Caso I: La Suite a instalar posee un certificado, por lo que se verifica si el certificado ha expirado. Este caso se divide en dos subcasos:

   [1.]Caso I.1: El certificado asociado está vencido, entonces se retorna el estado inicial sin modificaciones.

   [2.]Caso I.2: El certificado asociado no está vencido, por lo que se procede a verificar si el vendedor de la aplicación se encuentra en el repositorio del dispositivo. Este caso se divide en otros dos subcasos:

      [a.]Caso I.2.a: El vendedor existe en el repositorio, por lo que se procede a verificar si el certificado del MIDlet es el mismo que figura en el repositorio para ese vendedor; lo que nos lleva a dos nuevas posibilidades:

         [i.]Caso I.1.a.i: El certificado es el mismo que se encuentra en el repositorio, por lo que se devuelve el estado modificado, ya que la suite pasa a formar parte de las aplicaciones instaladas.

         [ii.]Caso I.1.a.ii: El certificado es diferente al que se encuentra en el repositorio, por lo que se devuelve el estado modificado, ya que la suite pasa a formar parte de las aplicaciones instaladas y se actualiza el certificado de ese vendedor en particular.

      [b.]Caso I.2.b: El vendedor no se encuentra en el repositorio del dispositivo, por lo que se devuelve el estado modificado, ya que la suite pasa a formar parte de las aplicaciones instaladas y se agrega el nuevo vendedor con su certificado asociado.

II. Caso II: La Suite a instalar no posee un certificado, por lo que se pasa a verificar si el vendedor está o no dentro del repositorio:

   [1.]Caso II.1: El vendedor se encuentra en el respositorio del dispositivo, por lo que se devuelve el estado modificado, ya que la suite pasa a formar parte de las aplicaciones instaladas.

   [2.]Caso II.2: El vendedor no se encuentra en el respositorio del dispositivo, por lo que se devuelve el estado modificado, ya que la suite pasa a formar parte de las aplicaciones instaladas y se agrega el vendedor.

A continuación, se muestra el código Coq del algoritmo descripto más arriba:

```
CAlg_Installation (ms : CMIDletSuite)(CurrentDate : nat)(cst : CState) : CState * option Response =def
match (hasCertificate ms) with      (* Verifica si tiene un certificado)
| true ⇒ match (isCertExpired (getCertificate ms) CurrentDate) with      (* Caso I *)
   | true ⇒ (cst, None)      (* Caso I.1 *)
   | false ⇒ match (isVendorInVR ms.vendor cst.vendorRepository) with      (* Caso I.2 *)
     | true ⇒
        match (isSameCertificate (getCertificate ms) cst.vendorRepository) with      (* Caso I.2.a *)
          | true ⇒ (mkCState      (* Caso I.2.a.i *)
                (addMIDletSuite cst ms) cst.session
                (PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
                (PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
                cst.vendorRepository, None)
          | false ⇒ (mkCState      (* Caso I.2.a.ii *)
                (addMIDletSuite cst ms) cst.session
                (PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
                (PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
                (updateVendorRep (getCertificate ms)cst.vendorRepository), None)
        end
     | false ⇒ (mkCState      (* Caso I.2.b *)
                (addMIDletSuite cst ms) cst.session
                (PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
                (PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
                (addVendorRepository cst
                   (mkCVendorRepository ms.vendor
                      ms.descriptor.signerCertificate)), None)
   end end
|false ⇒ match (isVendorInVR ms.vendor cst.vendorRepository) with      (* Caso II *)
          | true ⇒ (mkCState      (* Caso II.1 *)
              (addMIDletSuite cst ms) cst.session
              (PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
              (PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
              cst.vendorRepository, None)
          | false ⇒ (mkCState      (* Caso II.2 *)
              (addMIDletSuite cst ms) cst.session
              (PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
              (PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
              (addVendorRepository cst (mkCVendorRepository ms.vendor None)), None)
end          end
```

(4.10)

4.4. Certificación del Algoritmo

Si a partir de un estado inicial válido, en el cual se cumple la precondición de un evento, se obtiene por la ejecución de un algoritmo un estado que satisface la poscondición de dicho evento, decimos que dicho algoritmo está ***certificado***. Una propiedad fundamental que podemos verificar en esta formalización concreta es la certificación del algoritmo de ***Instalación*** propuesto. Esta propiedad se formaliza en el **teorema 3**:

TEOREMA 3: Para todo par de estados concretos *(cst cst' : CState)*, toda suite a instalar *(ms : CMIDletSuite)*, toda respuesta opcional *(r : option Response)*, toda fecha actual *(CurrentDate : nat)* y toda política de seguridad *(policy : Policy)*; si en el estado *cst* se cumple la precondición del evento *install* (*CPre_install cst ms.descriptor ms.domain policy ms.id*), entonces en el estado concreto resultante del algoritmo de instalación *(cst', r) = CAlg_Installation ms CurrentDate cst*, se cumple la poscondición del evento *install (CPos_install cst cst' r ms.descriptor ms.id ms CurrentDate)*.

Demostración[1]:

Para realizar la demostración, hay que hacer análisis de casos sobre las condiciones evaluadas por el algoritmo. La estrategia general se basa en inyectar como hipótesis la disyunción de cada una de las condiciones evaluadas en verdadero y falso, ya que en la representación concreta los predicados fueron expresados como funciones cuyos recorridos son los booleanos. Como para dar un ejemplo a modo de ilustración, tomamos la condición de si tiene o no un certificado asociado la aplicación a instalar. Entonces su inyección es a través de una afirmación como la disyunción de sus posibles valores:

$$\begin{aligned} hasCertificate\ ms &= true \vee \\ hasCertificate\ ms &= false \end{aligned} \tag{4.11}$$

[1]Para realizar esta demostración, se adopta la estrategia general de prueba utilizada en [4].

De esta manera, todos los casos detallados en el algoritmo son cubiertos.

Con fines prácticos, solamente se explicitará la demostración de uno de los casos del algoritmo: el Caso I.2.a.i. De la inyección de condiciones se agregan las siguientes hipótesis:

◊ *hasCertificate ms = true,* la suite posee un certificado.

◊ *isCertExpired (getCertificate ms) CurrentDate = false,* el certificado no está expirado.

◊ *isVendorInVR ms.vendor cst.vendorRepository = true,* el vendedor se encuentra en el repositorio de vendedores y certificados del dispositivo.

◊ *isSameCertificate (getCertificate ms) cst.vendorRepository = true,* es el mismo certificado que figura en el repositorio de vendedores y certificados del dispositivo.

Sustituyendo estas hipótesis en el algortimo, se obtiene la salida formada por el par *(cst', r)* que se incorpora como nueva hipótesis. Los valores, en este caso, son:

▷ *cst'* = mkCState (addMIDletSuite cst ms) cst.seesion
(PermissionGr cst ms.id cst.suite) (PermissionRv cst ms.id cst.suite)
(PermissionAu cst ms.id cst.suite) (PermissionAu cst ms.id cst.suite)
cst.vendorRepository

▷ *r* = None.

Para este caso, la única poscondición de *install* que aplica es la que tiene la conjunción de los siguientes predicados:

$$
\begin{aligned}
&CPos\_install\_vendor\_signed\_suite\_no\_exp\_installed\ cst\ cst'\ r\ ms\ CurrentDate\ =_{def}\\
&\quad hasCertificate\ ms\ =\ true\ \wedge\\
&\quad isCertExpired\ (getCertificate\ ms)\ CurrentDate\ =\ false\ \wedge\\
&\quad isVendorInVR\ ms.vendor\ cst.vendorRepository\ =\ true\ \wedge\\
&\quad isSameCertificate\ (getCertificate\ ms)\ cst.vendorRepository\ =\ true\ \wedge\\
&\quad r\ =\ None\ \wedge\\
&\quad cst'.suite\ =\ addMIDletSuite\ cst\ ms\ \wedge\\
&\quad cst'.granted\ ms.id\ =\ NoPermissionGr\ cst\ ms.id\ \wedge\\
&\quad cst'.revoked\ ms.id\ =\ NoPermissionRv\ cst\ ms.id\ \wedge\\
&\quad cst'.authorized\ ms.id\ =\ NoPermissionAu\ cst\ ms.id\ \wedge\\
&\quad cst'.unauthorized\ ms.id\ = NoPermissionUnaucstms.id\ \wedge\\
&\quad cst'.session\ =\ cst.session\ \wedge\\
&\quad (\forall sid0 : SuiteID,\ sid0 <> ms.id \rightarrow isMIDletSuite\ sid0\ cst.suite \rightarrow\\
&\qquad cst'.granted\ sid0\ =\ cst.granted\ sid0\ \wedge\\
&\qquad cst'.revoked\ sid0\ =\ cst.revoked\ sid0\ \wedge\\
&\qquad cst'.authorized\ sid0\ =\ cst.authorized\ sid0\ \wedge\\
&\qquad cst'.unauthorized\ sid0\ =\ cst.unauthorized\ sid0)\ \wedge\\
&\quad cst'.vendorRepository\ =\ cst.vendorRepository
\end{aligned}
\tag{4.12}
$$

- Las primeras cuatro conjunciones se demuestran trivialmente a partir de las hipótesis.
- *r = None*

  [.] Este objetivo se cumple trivialmente por ser una hipótesis incorporada con la salida del algoritmo.

- *cst'.suite = addMIDletSuite cst ms*

  [.] De la descomposición del constructor *mkCState* surge que el campo de las aplicaciones instaladas en el estado resultante, *cst'.suite*, tiene el valor *addMIDletSuite cst ms*. Esta función incorpora la nueva Suite al conjunto de aplicaciones instaladas. Al expandir

la definición de dicha función y reemplazando las hipótesis disponibles, se verifica que la nueva suite se encuentra instalada en el estado resultante cst'.

- *cst'.granted ms.id = NoPermissionGr cst ms.id*

  [.] De la descomposición del constructor *mkCState*, surge que el campo de los permisos otorgados en el estado resultante, *cst'.granted*, tiene el valor *PermissionGr cst ms.id cst.suite ms.id*. Haciendo análisis de casos sobre los identificadores (que son exactamente los mismos), se obtiene que la suite que quiere ser instalada tiene los mismos permisos que en el estado anterior, es decir, ninguno.

  [.] Es por esto que se verifica que la nueva suite no tiene permisos otorgados en el estado resultante cst'.

- *cst'.revoked ms.id = NoPermissionRv cst ms.id*

  [.] En este caso, la demostración es análoga a la de los permisos otorgados.

- *cst'.authorized ms.id = NoPermissionAu cst ms.id*

  [.] Al descomponer el constructor *mkCState*, se obtiene que el conjunto de las autorizaciones en el estado resultante $cst'.authorized$ tiene el valor *PermissionAu cst ms.id cst.suite ms.id*. Al aplicar la función a los mismos identificadores de la suite, se obtiene que la suite tiene las mismas autorizaciones que en el estado incial, es decir, ninguna.

  [.] Es por esto que se verifica que la nueva suite no tiene autorizaciones en el estado resultante cst'.

- *cst'.unauthorized ms.id = NoPermissionUnau cst ms.id*

  [.] La demostración para el caso de las desautorizaciones es análoga a la de las autorizaciones.

- *cst'.session = cst.session*

  [.] Se descompone el constructor *mkCState*, del estado resultante de la ejecución del algoritmo. Como la sesión activa es la misma luego de la instalación, se prueba la relación entre ambos estados.

- En el predicado que sigue, del estado resultante de la aplicación del algoritmo se descompone el constructor $mkCState$ en un conjunto de equivalencias para todas las suites distintas de la que se quiere instalar y que realmente estén instaladas. Considerando la connjunción de las equivalencias entre los campos que no se modifican, la relación entre ambos estados queda probada.

- *cst'.vendorRepository = cst.vendorRepository*

  [.] De la descomposición del constructor $mkCState$, del estado resultante de la ejecución del algoritmo, se obtiene que el repositorio no cambia porque no se agregan ni quitan vendedores, por lo que se prueba la relación entre ambos estados.

Al verificar cada uno de los predicados, se demuestra que la poscondición del evento *install* se cumple.

QED.

Capítulo 5

Conclusiones y Trabajos Futuros

El presente proyecto presenta una extensión en cuanto a la instalación de aplicaciones sobre MIDP 3.0. En este trabajo, se demuestra la conservación de las propiedades verificadas en las dos formalizaciones anteriores que fueron la base de ésta. Se incorpora un repositorio para almacenar vendedores y certificados, y se brinda una nueva descripción del elemento $Certificate$. Esta formalización brinda la posibilidad de realizar verificaciones sobre los vendedores de las aplicaciones al momento de su instalación. Se tienen en cuenta los dos casos posibles: que un vendedor tenga un certificado asociado o que no lo tenga. Esto aporta cierta inclusión, ya que por no poseer un certificado, no se puede concluir que la aplicación sea maliciosa.

Por otro lado, se brinda una definición más robusta sobre el evento *install*, de manera que tenga en cuenta la fecha para poder realizar verificaciones sobre la caducidad de los certificados de las aplicaciones al momento de su instalación. Esto previene la instalación de una aplicación de la cual no se tienen garantías de su vendedor.

Además, se han definido nuevas condiciones sobre el estado que deben verificarse para mantener la validez del mismo. Una de las condiciones es que un vendedor no pueda poseer dos certificados sin vencer. Esto previene la multiplicidad de un mismo vendedor y establece que un vendedor sólo posee un certificado.

Otro elemento destacable de este proyecto es el desarrollo de un algoritmo para evaluar la instalación de aplicaciones a partir de un refinamiento seguro de la formalización extendida.

Sin embargo, esta extensión plantea el punto de partida para continuar con la profundización de este evento tan importante. Aquí sólo se tiene en cuenta si la MIDlet Suite:

- ▷ Tiene certificado y está vencido, entonces no se instala la aplicación.
- ▷ Tiene certificado y es el mismo que está en el repositorio para ese vendedor, entonces se intala la aplicación.
- ▷ Tiene certificado y no es el mismo que está en el repositorio para ese vendedor, entonces se intala la aplicación y se actualiza dicho certificado para ese vendedor.
- ▷ Tiene certificado, pero el vendedor no se encuentra en el repositorio, entonces se instala la aplicación y se agrega dicho vendedor con su certificado asociado al repositorio.
- ▷ No tiene certificado y no se encuentra el vendedor en el repositorio, entonces se instala la aplicación y se agrega el vendedor al repositorio.
- ▷ No tiene certificado y se encuentra el vendedor en el repositorio, entonces se instala la aplicación.

Queda planteado como trabajo futuro la formalización de una característica importante a la hora de la verificación de un certificado: ***la verificación de la cadena de certificados***. Es decir, un certificado es tratado como confiable, sólo si se tiene confianza en la autoridad que lo certifica. Para ello, es necesario recorrer todo el "árbol de certificados" para saber si se confía en el emisor de dicho certificado y, a fin de cuentas, en el vendedor en sí.

Bibliografía

[1] Y. Bertot and P. Castéran. Interactive Theorem Proving and Program Development, Coq'Art: The Calculus of Inductive Construction. Springer-Verlag Berlin Heidelberg 2004.

[2] Sun Microsystems Inc. JSR 139: Connected Limited Device Configuration 1.1. `http://jcp.org/jsr/detail/139.jsp`, Último acceso: Enero 2013.

[3] Sun Microsystems Inc. JSR 218: Connected Device Configuration (CDC) 1.1. `http://jcp.org/jsr/detail/218.jsp`, Último acceso: Enero 2013.

[4] G. Mazeikis. Formalización y Análisis del Modelo de Seguridad de MIDP 3.0. Proyecto de Grado, Universidad de la República del Uruguay, 2009.

[5] G. Mazeikis, G. Betarte, and C. Luna. Formal Specification and Analysis of the MIDP 3.0 Security Model. sccc, pp.59-66, 2009 International Conference of the Chilean Computer Science Society, 2009.

[6] G. Mazeikis and C. Luna. Autorización de Acceso en MIDP 3.0. Congreso Iberoamericano de Seguridad Informática, CIBSI'09, Uruguay, Noviembre de 2009.

[7] Nokia. Mobile Information Device Profile for Java Micro Edition, Version 3.0, 2009. `http://www.developer.nokia.com/Community/Wiki/MIDP_3.0`, Último acceso: Diciembre 2012.

[8] Oracle. Java Community Process. `http://jcp.org/en/introduction/overview`, Último acceso: Julio 2012.

[9] Oracle. The Java Micro Edition Platform. `http://www.oracle.com/technetwork/java/javame/index.html`, Último acceso: Diciembre 2012.

[10] Oracle. The Java Standard Edition Platform. http://www.oracle.com/technetwork/java/javase/overview/index.html, Último acceso: Julio 2012.

[11] C. Prince. MIDP Security Model, Código Coq de la Especificación Formal, 2013. https://github.com/cristianprince/Install_MIDP3.0, Último acceso: Junio 2013.

[12] C. Prince. MIDP Security Model, Código Coq de la Especificación Formal, 2013. http://www.fing.edu.uy/~cluna/Prince_tesina.zip., Último acceso: Junio 2013.

[13] Inc. Sun Microsystems. Mobile Information Device Profile (JSR-37), 2000. http://download.oracle.com/otndocs/jcp/7580-j2me_midp-1.0a-fr-spec-oth-JSpec/, Último acceso: Febrero 2013.

[14] Inc. Sun Microsystems and Inc. Motorola. Mobile Information Device Profile for Java 2 Micro Edition, 2002. http://download.oracle.com/otndocs/jcp/7121-midp-2.0-fr-spec-oth-JSpec/, Último acceso: Febrero 2013.

[15] The Coq Development Team. The Coq Proof Assistant Reference Manual, 2011. http://coq.inria.fr/distrib/V8.4/refman/, Último acceso: Junio 2013.

[16] S. Zanella. Especificación Formal del Modelo de Seguridad de MIDP 2.0 en el Cálculo de Construcciones Inductivas. Masther's Thesis, Universidad Nacional de Rosario, 2006.

[17] S. Zanella, G. Betarte, and C. Luna. A Formal Specification of the MIDP 2.0 Security Model. Formal Aspects in Security and Trust 2006: 220-234, Hamilton, Ontario, Canada.