Présenté à: Simposio Argentino de Informática y Salud (SIS) de la JAIIO 2001 (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina, septembre 2001. (http://www.ing.unlpam.edu.ar/jaiio2001/index_principal.html)

Historias Clínicas en Tarjetas Inteligentes

Ing. Leonardo Rodríguez[a], Ing. Daniel Perovich[a], Ing. Martín Varela[a]
Co-autor: Dr. Luis Martínez[b]

[a] Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay
[b] Hospital de Clínicas, Facultad de Medicina, Universidad de la República, Uruguay

Resumen

Las tarjetas inteligentes han sido utilizadas desde los años '70, y sus aplicaciones han evolucionado gracias a avances tecnológicos en el área. Una tarjeta de salud (HealthCard) es una tarjeta inteligente que contiene información personal, médica y de seguridad, utilizada en sistemas médicos. En éstos sistemas existen dos grandes problemas, la heterogeneidad de los sistemas componentes, que dificultan su interoperabilidad, y la confidencialidad de la información médica que éstos manejan. Se puede adoptar el uso de HealthCard para resolver estos problemas. El presente artículo se enmarca en un proyecto más ambicioso donde se realizó un estado del arte de la tecnología de SmartCard y JavaCard, se desarrolló herramientas que asisten en la construcción de aplicaciones para esta tecnología, y se elaboró un prototipo de Historias Clínicas en Tarjetas Inteligentes como caso de estudio.

Palabras calves:

SmartCard; JavaCard; HealthCard; Tarjeta de Salud; Historias Clínicas; Historias Clínicas Resumidas; Sistemas Embebidos

Introducción

Numerosos campos de aplicación de la Internet y de las tarjetas inteligentes requieren que datos y recursos sean protegidos por medio de mecanismos de seguridad. El lenguaje de programación Java representa una respuesta práctica a las cuestiones de movilidad y seguridad sobre Internet. Sun Microsystems Inc. publicó la definición de un nuevo miembro de la tecnología Java, llamado *JavaCard* [1], que está orientada a la programación de tarjetas inteligentes. *JavaCard* fue diseñado excluyendo ciertas construcciones de Java complejas o no aplicables a la programación de tarjetas inteligentes y agregando funciones específicas como el manejo de transacciones en tarjetas inteligentes.

En el proyecto marco se estudió las herramientas de desarrollo existentes así como las prestaciones y limitaciones de la plataforma y la viabilidad de elementos de seguridad en el desarrollo de aplicaciones. El conocimiento adquirido permitió identificar problemas y limitaciones de la plataforma, y desarrollar herramientas tanto de software como metodológicas, para solucionar los mismos. Así mismo permitió encarar un proyecto original para la captura y el almacenamiento de datos en la JavaCard, proponiendo para esto una estructura de datos eficiente.

El presente artículo resume el estado del arte de la tecnología de SmartCard y JavaCard incluyendo aspectos de seguridad. Introduce las herramientas de software y metodológicas utilizadas y construidas en el proyecto marco. Presenta, por último, el caso de estudio: un prototipo de Historias Clínicas en Tarjetas Inteligentes.

Estado del arte

SmartCard

Una SmartCard es un dispositivo del tamaño de una tarjeta de crédito[1], el cual almacena y procesa información mediante un circuito de silicio embebido en el plástico de la tarjeta [2]. Las tarjetas magnéticas utilizadas en cajeros automáticos y como tarjetas de crédito son antecesoras de las SmartCards. Las mismas almacenan información en una banda magnética de tres pistas que llevan adherida sobre la superficie. Las SmartCards han existido en varias formas desde 1974. Desde ese entonces, gracias a la motivación de compañías como Gemplus y Schlumberger, han recibido gran atención en el mercado de los dispositivos de control. Según la consultora Frost & Sullivan más de 600 millones de SmartCards fueron emitidas en 1996 y se espera un consumo de 21 billones para el año 2010 [3].

Puede clasificarse a las SmartCards en dos tipos:

- Memory Cards (también llamadas Low-End): Tienen

[1] Se debe aclarar que la tecnología de SmartCards no se reduce estrictamente a tarjetas, como lo demuestra el amplio uso de los iButtons de Dallas Semiconductor.

circuitos de memoria que permiten almacenar datos. Estas tarjetas utilizan cierta lógica de seguridad, a nivel del hardware, para controlar el acceso a la información.

- Microprocessor Cards (también llamadas High-End): Tienen un microprocesador que les brinda una limitada capacidad de procesamiento de datos. Tiene capacidad de lectura, escritura y procesamiento.

Las SmartCards contienen tres tipos de memoria: ROM, EEPROM y RAM. Normalmente no contienen fuente de poder, display o teclado, e interactúan con el mundo exterior utilizando una interfaz serial con ocho puntos de contacto[2]. Las dimensiones y ubicación de los mismos están especificados en la parte 2 de estándar ISO 7816 [4].

Son utilizadas en conjunto con un Card Acceptance Device (CAD o Reader), que puede, aunque no necesariamente, estar conectado a una computadora. Este dispositivo tiene como objetivo el proveer la fuente de poder e interfaz de comunicación para las tarjetas. La interacción entre la tarjeta y el CAD puede hacerse mediante un punto de contacto (la tarjeta debe ser insertada en el CAD) o sin contacto (utilizando radiofrecuencia).

JavaCard

Dentro de la categoría de SmartCards con microprocesador se encuentran las llamadas JavaCards o Java SmartCards. Una JavaCard es una SmartCard capaz de ejecutar programas desarrollados en Java. Originalmente, las aplicaciones para SmartCards eran escritas en lenguajes específicos de los proveedores de SmartCards (normalmente el ensamblador del microprocesador utilizado, o eventualmente C). El lema "escribe una vez, corre en cualquier lado" hizo que Java fuese una solución a este problema [2]. La primera JavaCard en salir al mercado fue producida por Schlumberger, aún antes de que Sun fijara el standard [5]. En pocas palabras, una JavaCard es una tarjeta con microprocesador que puede ejecutar programas (llamados applets) escritos en un subset[3] del lenguaje Java.

Los componentes principales dentro de una JavaCard son el microprocesador y las memorias. La arquitectura básica de una JavaCard consiste de Applets, JavaCard API, JavaCard Virtual Machine (JCVM) [6], JavaCard Runtime Environment (JCRE) [7], y el sistema operativo nativo de la tarjeta. La máquina virtual corre sobre el sistema operativo de la tarjeta, y se puede ver como una abstracción del mismo. Sobre la máquina virtual se encuentra el JavaCard Framework, que es un conjunto de clases necesarias para el funcionamiento del JCRE y JavaCard API, así como clases utilitarias comerciales o extensiones propietarias del fabricante de la tarjeta. Finalmente, haciendo uso de este ambiente se encuentran los applets [2, 3].

JavaCard Applet

Los applets son las aplicaciones que corren embarcadas en una JavaCard. Dichas aplicaciones interactúan en todo momento con el JCRE utilizando los servicios que éste brinda, e implementan la interfaz definida en la clase abstracta `javacard.framework.Applet`, la cual deben extender [8]. Se puede decir que un applet comienza su ciclo de vida al ser correctamente cargado en la memoria de la tarjeta, linkeditado y preparado para su correcta ejecución [6]. Una vez registrada ante el JCRE un applet está en condiciones de ejecutar. Este applet normalmente existe durante el resto de la vida de la tarjeta.[4]

Seguridad física

Las SmartCards son muy seguras y extremadamente difíciles de atacar, ya sea física o lógicamente. Algunos fabricantes proclaman que sus tarjetas son físicamente inviolables, y que disponen de numerosas defensas contra ataques físicos, como ser detección de ciclos de reloj anormales en frecuencia, microprobing, retiro de la cubierta de resina epoxi o exposición del microprocesador a luz ultra violeta. Sin embargo, hay claros ejemplos en los que se ha logrado obtener, a través de diversos medios, la información contenida en una SmartCard [9, 10]. En la Universidad de Cambridge, un grupo de investigadores ha desarrollado varios métodos de extracción de información protegida dentro de una SmartCard, y los resultados que plantean no son muy alentadores. Según la clasificación de los posibles atacantes propuesta por IBM [8, 9], un atacante de clase I, que podría ser un estudiante de grado de Ingeniería Eléctrica con menos de US$ 400 de equipamiento, podría lograr extraer información de una SmartCard. De todos modos, los ataques registrados como exitosos sucedieron hace ya bastante tiempo, y los fabricantes afirman que los componentes son cada vez más seguros.

Seguridad lógica

Las SmartCards multi-aplicación se están haciendo más y más populares. Los usuarios desean reducir el número de tarjetas en su billetera y las empresas emisoras desean reducir los costos de desarrollo y emisión de las mismas así como ampliar los servicios brindados. Además, el uso de SmartCards multi-aplicación permite la cooperación entre socios de negocios para lograr nuevas oportunidades de negocio.

Las SmartCard proveen seguridad de activación mediante mecanismo de PIN (Personal Identification Number) en la propia tarjeta. La especificación actual de JavaCard provee un API para el uso de algoritmos criptográficos, generación de claves, etc. también embarcados en la tarjeta. El hecho

[2] Existen, además, SmartCards que se comunican por radiofrecuencia.

[3] Estrictamente hablando, este lenguaje no es exactamente un subset de Java, ya que introduce algunos elementos nuevos en la JVM, como ser mecanismos de seguridad adicionales.

[4] Si bien algunas implementaciones permiten el borrado de applets, normalmente una vez que la tarjeta es distribuida, los applets que tuviese embarcados permanecen en la misma

de ser multi-aplicación conlleva a un problema de seguridad donde un applet podría extraer información de otro applet, si autorización. Para solucionar este problema la especificación prevee brinda un mecanismo, llamado firewall, el cual no permite a un applet acceder a la información de otro directamente.

Herramientas de Desarrollo y Metodologías

Emulador

Junto a su kit de desarrollo, Sun provee un emulador de JavaCard [11]. El mismo consiste de varios componentes, básicamente clases, verificador de bytecode, conversor y emulador. Para la utilización del emulador se requiere de dos componentes: el emulador, y una herramienta de scripting de APDUs (apdutool). El emulador se ejecuta y queda a la espera de APDUs en un puerto dado. El apdutool se conecta a dicho puerto, y actúa como driver del CAD, enviando APDUs que toma de un script y generando un log con las respuestas del emulador. El script consiste en una secuencia de números hexadecimales, que representan los comandos a enviar a la tarjeta. La utilización del emulador presenta algunos inconvenientes:

- No permite interacción con el applet, ya que el script es una sesión completa entre el CAD y la tarjeta.
- La generación de los APDUs es manual, ya que no hay herramientas adecuadas en el kit para esta tarea[5]. Esto resulta extremadamente tedioso y muy propenso a errores.

Proxy

Una posible solución a los problemas del emulador es la implementación de una herramienta que sustituya al apdutool, interactuando directamente con el emulador desde una interfaz que pudiera ser manipulada interactivamente por el usuario. El desarrollo de esta herramienta se ve dificultado por la falta de documentación referente al protocolo utilizado entre el apdutool y el emulador. Además una interfaz de usuario permitiría lograr interactividad en el testeo de las applets, pero la creación de los APDUs seguiría siendo manual, o reprogramada de alguna forma para cada applet.

Se optó por un enfoque más elaborado, buscando la encapsulación de la generación de APDUs en una interfaz de más alto nivel. Se desarrolló una herramienta llamada Proxy, conceptualmente similar al mecanismo de RPC. La completa abstracción de la comunicación entre el applet y la terminal tiene como ventaja que permite aumentar la velocidad de desarrollo tanto de las aplicaciones de terminal, como de los applets, disminuyendo además la cantidad de errores debido al manejo directo de APDUs. La misma permite al desarrollador de la aplicación de terminal (CAD) ver al applet en la tarjeta como si fuera un objeto local, y al desarrollador del applet implementar el mismo como una clase normal. Si bien la implementación actual todavía presenta algunos problemas que impiden su correcto funcionamiento, se está trabajando para solucionarlos. La nueva especificación de la tecnología JavaCard, que aun no está disponible, promete incorporar un mecanismo similar al aquí propuesto.

Kit

El caso de estudio fue implementado utilizando el kit de desarrollo de Schlumberger, que incluye el reader Reflex 72 RS232 Reader, dos Cyberflex SmartCards y dos Criptoflex SmartCards.

El kit, llamado Cyberflex Access SDK, provee las siguientes facilidades:

- Interacción con la tarjeta a nivel de APDU y posibilidad de realizar scripts de APDU
- Monitoreo de la comunicación entre la tarjeta y el reader
- Framework para el desarrollo de aplicaciones que necesitan comunicarse con las tarjetas Cyberflex

Tanto el reader como las tarjetas Cyberflex presentan algunos problemas al momento de desarrollar una aplicación. Los mismos llevaron a un tiempo de desarrollo mayor para el caso de estudio.

Metodología

Las JavaCard son multi-aplicación, esto es, permiten embarcar más de un applet. La especificación actual permite la interacción entre los applets embarcados en una tarjeta, y brinda mecanismos de seguridad de forma de que éstos protejan sus datos de otros applets. El mecanismo que provee la especificación para compartir objetos tiene algunos problemas [12]. En el proyecto marco se elaboró una metodología que permite resolver estos problemas sin requerir de modificaciones en la especificación [13].

Caso de estudio

El caso de estudio estuvo enfocado en el desarrollo de una HealthCard, que es una tarjeta inteligente que contiene información personal, médica y de seguridad utilizada por los sistemas médicos. Esta tarjeta puede ser leída por lectores de tarjetas ubicados en los sistemas partícipes, por ejemplo: hospitales, sistemas de emergencia médica móvil, farmacias, clínicas, consultorios particulares. Las personas que participen del sistema llevan consigo su tarjeta, la cual guarda información importante, accesible en forma inmediata. Esta tecnología está en uso en varios países (un ejemplo de estos sistemas puede encontrarse en [14]).

Este desarrollo fue realizado sobre una JavaCard, de forma de poner en práctica los conocimientos obtenidos y evaluar las herramientas de desarrollo.

[5] Hay productos comerciales provistos por terceros que si lo permiten

Los principales objetivos fueron:

- Definir la información relevante a almacenar dentro de la HealthCard.
- Desarrollar un prototipo funcional de una HealthCard utilizando tecnologías JavaCard.
- Desarrollar una aplicación para estaciones de trabajo que permita interactuar con la HealthCard, permitiendo así manipular la información almacenada en ella.

Análisis del problema – Fuentes y Enfoque

El análisis de requerimientos del caso de estudio tuvo como fuente principal reuniones en el Hospital de Clínicas, la asistencia a eventos de informatización de historias clínicas (SUIS - Sociedad Uruguaya de Informática en la Salud), la documentación publicada por el Dr. Roberto Rocha, que está siendo utilizada en Brasil [15, 16], e información en Internet.

En otros países se utiliza la tecnología de SmartCard aplicada a historias clínicas con un fin diferente al propuesto por nosotros. Su principal uso es por concepto de reducción de gastos en papelería y trámites que una persona realiza cada vez que se atiende, así como para evitar fraudes.

El enfoque que se ha tomado en este trabajo difiere con el arriba descrito. Se detectó que la utilidad del sistema para el propio Hospital de Clínicas era baja, ya que una red local con un sistema de información adecuado satisface de mejor manera las necesidades planteadas. La SmartCard brinda un valor agregado cuando no es posible tener la información centralizada; por ejemplo en sistemas de emergencia móvil y en sistemas de salud descentralizados con pequeños consultorios remotos sin conexión a un sistema central.

Análisis del problema – Requerimientos

Un primer análisis detectó que el principal objetivo era determinar la información a almacenar dentro de la HealthCard. Una tarjeta ideal debería almacenar toda la historia clínica del paciente, de forma de que ante cualquier emergencia el médico tenga la posibilidad de acceder a la mayor cantidad de información disponible del paciente. Dada la escasez de recursos de memoria en las tarjetas existentes, es necesario almacenar solamente los datos fundamentales de la misma. Por esta razón fue necesario definir una historia clínica resumida con los datos de mayor importancia.

Actualmente a nivel nacional no existe ninguna propuesta de especificación formal o informal de que información debe tener una historia clínica resumida. Tomando como base experiencias realizadas en Brasil [15, 16] se llegó a una especificación informal de los datos relevantes.

Ésta marcaba siete categorías de datos fundamentales a almacenar en la tarjeta, estas son:

1. Datos administrativos
2. Datos de la institución de salud
3. Alergias y reacciones adversas
4. Enfermedades crónicas
5. Diagnósticos realizados
6. Procedimientos realizados
7. Fármacos en uso

En la Tabla 1 se muestra en detalle la información de cada categoría, así como un análisis del tamaño requerido (considerando una representación estática) para su almacenamiento en la tarjeta.

Tabla 1 – Información relevante

Categoría	Item	Tipo	Tamaño (bytes)
Datos administrativos	Código del paciente	char	10
	Tipo de documento	char	3
	Documento	char	15
	Nombre	char	50
	Fecha de nacimiento	date	8
	Edad aparente	byte	1
	Departamento de nacimiento	char	2
	País de nacimiento	char	2
	Sexo	char	1
	Dirección	char	100
	Teléfono	char	20
	Nombre del padre	char	50
	Nombre de la madre	char	50
	Situación familiar	char	2
	Color de piel	byte	1
	Nombre del contacto	char	50
	Teléfono del contacto	char	20
	Dirección del contacto	char	100
	Descripción del contacto	char	50
	Fecha actualización	date	8
Subtotal para esta categoría			543
Datos de la institución de salud	Nombre de la institución	char	100
	Departamento	char	2
	País	char	2
Subtotal para esta categoría			104
Alergias y reacciones adversas	Alergia o reacción	char	100
	Tabla de diagnóstico	byte	1
	Codificación de diagnóstico	byte	10
Subtotal para esta categoría (10 entradas)			1110
Enfermedades crónicas	Tabla de diagnóstico	byte	1

	Codificación del diagnóstico y dolencias crónicas	byte	10
Subtotal para esta categoría (10 entradas)			110
Diagnósticos realizados	Tabla de diagnósticos utilizada	byte	1
	Codificación de diagnóstico principal	byte	10
	Codificación de diagnóstico secundaria	byte	10
Subtotal para esta categoría (10 entradas)			210
Procedimientos realizados	Método de codificación de procedimientos	byte	1
	Código del procedimiento	byte	10
	Nombre o código del profesional que lo realizó	char	50
	Fecha del procedimiento	date	8
Subtotal para esta categoría (10 entradas)			690
Fármacos en uso	Nombre o denominación	char	50
	Codificación del fármaco	byte	10
	Dosis	char	100
	Unidades	char	5
	Vía	char	10
	Intervalo	char	25
	Tiempo de utilización	char	25
Subtotal para esta categoría (10 entradas)			2250
Tamaño Total			5017

El tamaño calculado es de aproximadamente 5 KB en total. El tamaño es relativamente adecuado para ser almacenado en una tarjeta. Notar que se consideraron en forma estática 10 entradas por cada una de las colecciones.

Por la limitación en el espacio también se consideraron formas de codificar la información que así lo permitía. Se buscó codificaciones estándares para las enfermedades, procedimientos, fármacos, etc.

Análisis del problema – Características del problema

De un análisis más profundo de la estructura y características de la información que maneja el problema, se observó lo siguiente:

- La información está compuesta por partes estáticas (como el nombre del paciente) y partes dinámicas (como la colección de fármacos)
- Presenta una estructura arborescente, donde la información se encuentra en las hojas del mismo.
- La cantidad de elementos en las diferentes colecciones depende mucho de cada paciente. Por ejemplo, puede haber un paciente sin enfermedades crónicas, pero al que se le han realizado varios procedimientos. Por lo tanto, poner un limite arbitrario, que no contemple dicho problema, puede traer aparejado más problemas. Además, los límites no pueden fijarse muy altos como para obviar estos inconvenientes, debido a las restricciones de espacio.
- La aplicación resultante debería ser fácilmente personalizable, en lo posible sin la necesidad de reimplementar. El sistema está orientado a trabajar con sistemas de emergencia móvil. En este momento no se cuenta con una especificación estándar de las historias clínicas resumidas a nivel nacional, por lo cual es muy probable que una aplicación de este tipo va a requerir cambios al trabajar con distintos sistemas de emergencia móvil.
- La solución debería estar preparada a evolucionar y adaptarse a nuevos requerimientos, sin necesidad de embarcar nuevamente todas las tarjetas. Este es un proceso muy costoso en tiempo, el cual debe ser evitado.

Solución al problema

El desarrollo de una aplicación utilizando tarjetas inteligentes, normalmente consta de tres grandes componentes. Una aplicación del lado de la tarjeta, una aplicación del lado del terminal y un protocolo de comunicación. La aplicación del lado del terminal es la que interactúa con el usuario y a través del protocolo de comunicación interactúa con la aplicación en la tarjeta.

Se debe tener esto en cuenta al momento desarrollar la HealthCard. A continuación se presenta dos soluciones posibles, para luego hacer un comparativo entre ambas.

Solución al problema – Versión inicial

Una solución simple para este problema es realizar una representación estática de la información, como se presentó en la Tabla 1. Esto se logra fácilmente haciendo un modelo orientado a objetos de la realidad, obteniendo como resultado el diagrama de clases que lo modela.

Esta solución presenta la ventaja que es muy fácil de implementar y plantea un diseño muy cercano a la realidad que se está modelando. Por otro lado, presenta ciertas carencias que no la hacen una solución óptima, a saber:

- Manejo ineficiente de la memoria. Al ser una solución estática, los tamaños de las colecciones y los tamaños de los campos individuales (como puede ser el nombre del paciente) se tienen que fijar al

momento de instanciar el applet en la tarjeta. Por lo tanto es necesario fijar un tamaño arbitrario para las colecciones, no permitiendo que una colección que requiera más espacio pueda utilizar el espacio que no está siendo utilizado por otra. Este mismo manejo ineficiente de la memoria se da a nivel de los campos individuales. Por ejemplo, el tamaño del string que representa el nombre tiene que ser fijo y suficientemente grande para soportar nombres largos. En la mayoría de los casos los nombres van a ser relativamente cortos, por lo cual se desperdicia espacio.

- No está apta para evolucionar, sin tener que embarcar nuevamente la aplicación en todas las tarjetas. Cualquier agregado en los requerimientos implica cambios en el modelo de objetos representado a nivel de la tarjeta y para poder reflejar estos cambios es necesario volver a embarcar la aplicación.
- No permite personalizar la solución sin tener que reimplementar la misma. Las razones son las mismas que en el punto anterior, un cambio en los requerimientos lleva a reimplementar alguna de las clases que conforman la solución a nivel de la tarjeta.

Solución al problema – Presentación de la versión final

Esta solución intenta resolver los problemas que plantea la solución inicial, que son: mala administración de la memoria, dificultades para evolucionar y dificultades para personalizar. La idea detrás de esta solución surge de la abstracción de la estructura de la información que se va a almacenar en la tarjeta.

Dos de las características del problema que fueron mencionadas son:

- Se puede representar mediante una estructura arborescente.
- La información se encuentra almacenada solamente en la hojas del árbol.

Esto permitiría utilizar un árbol para estructurar y almacenar la información. Este permitiría representar la información estática, como el nombre del paciente, en un nodo del mismo. También permite almacenar las colecciones, como un nodo que representa a la colección y los hijos del mismo que representan a los elementos de la colección. Este árbol debe permitir amoldar su estructura a la estructura de la información del problema, como se puede ver en el ejemplo de la Figura 1.

El utilizar esta estructura tiene como ventaja que asimila fácilmente los cambios en la estructura de la información del problema, que pueden presentarse como resultado del mantenimiento de la aplicación o por la propia evolución de la realidad modelada. No es necesaria la reimplementación del applet, basta, en estos casos, con un cambio en la estructura del árbol para incorporar el nuevo requerimiento.

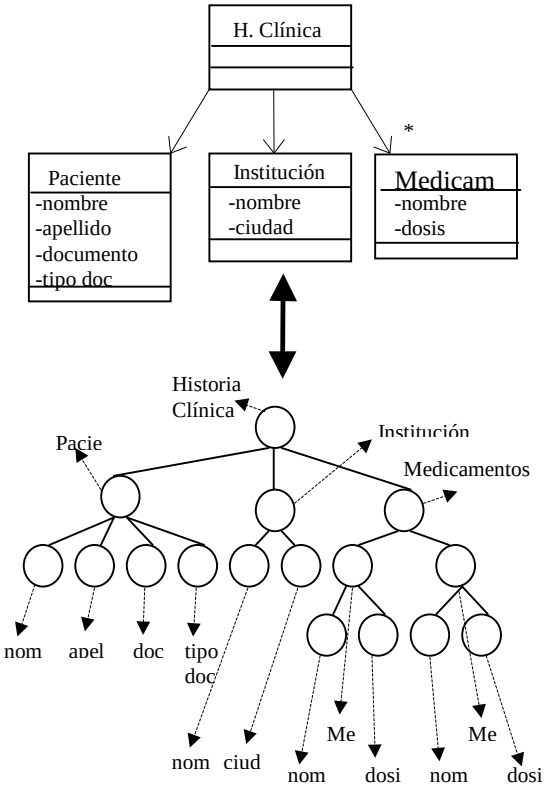

Figura 1

La implementación de un árbol genérico como propone esta solución, no está limitado únicamente a resolver el problema de la representación de historias clínicas, sino que permite resolver cualquier otro de similares características. Todo problema que requiera almacenar un conjunto de datos cuya estructura sea representable por un árbol con información en las hojas puede ser resuelto utilizando esta solución.

Para poder lograr esto primero se debe encontrar un mecanismo para representar el árbol dentro de la tarjeta. La estructura del árbol deberá ser dinámica para poder asimilar los cambios de estructura antes mencionados. La JCVM por el momento, no dispone de un mecanismo para la liberación de la memoria [6], por lo cual la memoria necesaria para almacenar la estructura del árbol deberá ser reservada al momento de embarcar el applet.

Esta limitación impuesta por la JCVM afecta también a la información almacenada dentro de los nodos del árbol, ya que obliga a fijar de forma arbitraria el tamaño máximo de cada uno de los campos. Es necesario poder reservar dinámicamente la memoria de estos campos, logrando así utilizar solamente la memoria necesaria. Esto permitiría realizar un manejo más eficiente de la memoria

Para solucionar estos problemas claramente es necesario un manejador de memoria que permita tener un manejo más flexible de la memoria en la tarjeta que el que brinda la JCVM. Por esto se decidió implementar un manejador de memoria dentro de la tarjeta, obteniendo así la flexibilidad necesaria.

Solución al problema - Implementación de la versión final

A modo de resumen, esta solución permite resolver los problemas que presentaba la solución inicial mediante un árbol para representar la información (solucionando los problemas de cambios en la estructura de la información del problema) y la implementación de un manejador de memoria (para hacer posible lo anterior y realizar un manejo eficiente de la memoria).

Para minimizar el uso de la memoria de la tarjeta, se optó por no almacenar la estructura del árbol en la tarjeta, sino solamente almacenar la información de los nodos. La estructura del árbol se almacena del lado de la terminal.

Para ello se utiliza un template. Un template es, a su vez, un árbol que representa el esqueleto de la estructura de la información almacenada en la tarjeta, y define el tipo de los datos almacenados en cada hoja, así como los nodos que son colecciones y la estructura de los elementos de las colecciones.

En base a la información que provee el template y con la información dentro de la tarjeta es posible reconstruir el árbol con la información. Esto se logra recorriendo el template y simultáneamente construyendo el árbol final. A medida que se procesa cada uno de los nodos del template se lee de la tarjeta la información del nodo, y se la interpreta en base al tipo de dato definido en el template. En el caso que sea una colección, se leen todos los nodos que la componen.

La aplicación en la terminal tiene como principal responsabilidad la representación del árbol, la cual es el núcleo de la solución propuesta. La aplicación también va a ser responsable de reconstruir el árbol, así como de almacenar las modificaciones realizadas.

Dado que la estructura de la información que se maneja es arborescente, sería muy fácil representar esta información utilizando *Simple Markup Language (SML)* [17, 18], el cual es un lenguaje de tags simple, especial para representar información con estructura arborescente.

La aplicación en el terminal utiliza esta característica para facilitar el flujo de información y lograr interoperabilidad con otros sistemas. Esto lo hace permitiendo especificar tanto los templates y la información de los pacientes utilizando SML.

El la Figura 2 se muestra un esquema representando la arquitectura general de la aplicación en el terminal.

Como se ve en la figura la aplicación en la terminal va a permitir manejar indistintamente si la entrada o salida proviene de un archivo en SML o de los datos almacenados en la tarjeta inteligente. Modelando la solución de esta manera, es posible realizar conversiones en forma natural entre la información dentro de una tarjeta a un archivo SML y viceversa.

Por detalles sobre la implementación de la misma referirse a [19].

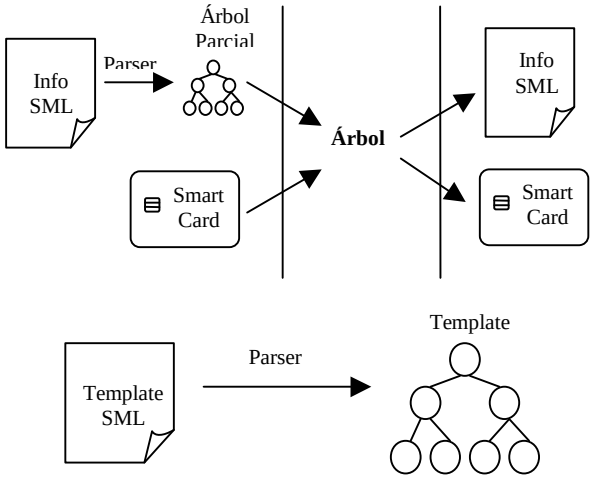

Figura 2

Solución al problema – Aplicación de la solución general

Resumiendo lo presentado en los puntos anteriores, se dispone de una solución genérica que permite resolver problemas que involucren un conjunto de datos que presenten una estructura arborescente, y que deban ser almacenados, consultados y actualizados.

Aplicar la solución a este tipo de problemas sugiere dos etapas de desarrollo. La primer etapa consiste en definir la estructura de la información y cargar los datos en las tarjetas, y la segunda consiste en la construcción de una aplicación del lado de la terminal que haga uso de los componentes descriptos anteriormente para acceder a la información.

En la primer etapa se define el template que especifica la estructura de la información que interesa almacenar. Esto se puede hacer especificando la estructura en SML y generando el árbol del template a partir de este archivo, o también implementando un programa en Java que construya el árbol. Es evidente que el especificar la estructura en SML es la opción más amigable, ya que hacerlo en Java presenta una tarea engorrosa.

Una vez generado el árbol del template es posible obtener un árbol final, con la información propia de la instancia del problema. Para esto es necesario disponer de esta información en alguno de los medios de los cuales la aplicación en la terminal permite leer, que pueden ser de una tarjeta o de un archivo SML. Ya que el objetivo es cargar la información por primera vez a la tarjeta, se va a obtener la información del paciente de un archivo SML.

Después de generado el árbol, sólo queda almacenarlo en la tarjeta inteligente, culminando así la primer etapa.

En la segunda etapa se construye una aplicación que utilice la capa lógica de la solución, para que el usuario pueda manipular la información almacenada en la tarjeta. La aplicación es la encargada de manipular el árbol, permitiendo consultar o modificar la información y cargar o

almacenanr la información cuando sea necesario.

Discusión

Comparando la solución implementada con la solución propuesta inicialmente, se observa que la misma corrige todas las carencias que tenía la solución inicial, pero a su vez introduce algunas desventajas.

Las ventajas que tiene sobre la solución inicial son las siguientes:

- Realiza un manejo más eficiente de la memoria, administrando dinámicamente el área de memoria disponible. No plantea límites de tamaños en los datos, ni para la cantidad de elementos de las colecciones
- Plantea una solución completamente independiente de la estructura de los datos del problema. Esto permite que esta solución se puede reutilizar para resolver otros problemas con características similares.
- Permite que el applet evolucione para cumplir nuevos requerimientos, sin la necesidad de reimplementarlo ni de embarcar nuevamente las tarjetas.
- Permite personalizar un applet sin la necesidad reimplementarlo

Las desventajas que introduce son las siguientes:

- Tiene un nivel más de indirección. Para manipular la información siempre se debe pasar por el manejador de memoria, agregando así un nivel extra de indirección
- Es más difícil de implementar. Requiere implementar el manejador de memoria, mientras que la solución inicial solo requería implementar las clases del modelo de objetos, que son solamente propiedades y colecciones.
- Requiere más espacio para el código del applet. La lógica que se requiere para el manejador de memoria aumenta los requerimientos de espacio para el código. La implementación realizada ocupa aproximadamente 3.5 KB.

Conclusiones

La tecnología de SmartCards brinda un valor agregado fundamental en contextos donde la información debe acompañar al usuario y donde la seguridad es importante. La tecnología JavaCard permite mejorar la calidad de las aplicaciones, acortando los tiempos de desarrollo e independizando a la solución del hardware utilizado.

El uso de HealthCard puede ser un gran aporte a los problemas de interoperabilidad y seguridad en los sistemas de información médicos, así como en los servicios de emergencia médica móvil.

En el proyecto marco se está planificando el desarrollo de una aplicación basada en tarjetas inteligentes para el departamento de hemoterapia en el Hospital de Clínicas, en colaboración con la Asociación de Hemofílicos del Uruguay.

Agradecimientos

Agradecemos especialmente a Gustavo Betarte por brindar su conocimiento, apoyo y entusiasmo durante la tutoría del proyecto. A Regina Motz, a Carlos Luna y al Ing. Franco Simini por su aporte en temas relacionados en Informática y Salud. Al Dr. Roberto Rocha del Hospital de Clínicas de la Universidad Federal de Panamá por compartir sus experiencias en el área de historias clínicas resumidas.

Referencias

[1] Sun Microsystems Inc., *JavaCard™ Technology*, http://java.sun.com/products/javacard

[2] Chen, Z., Di Giorgio R., Sun Microsystems Inc., *Understanding Java Card 2.0*, 1998, http://www.javaworld.com/javaworld/jw-03-1998/jw-03-javadev.html

[3] Chan, Y. L., Chan, H. Y., *What is a Java Smart Card?*, SURPRISE 98 Report

[4] ISO, *ISO7816 (part 1-3) Asynchronous SmartCard Information*, http://www.scia.org/aboutSmartCards/iso7816_wimages.htm

[5] Di Giorgio, R., Sun Microsystems Inc., *Smart Cards: a premier*, 1997, http://www.javaworld.com/jw-12-1997/jw-12-javadev.html

[6] Sun Microsystems Inc., *Java Card Virtual Machine Specification v2.1*, 1999, http://java.sun.com/products/javacard/javacard21.html

[7] Sun Microsystems Inc., *Java Card 2.1 Runtime Environment (JCRE) Specification*, 1999, http://java.sun.com/products/javacard/javacard21.html

[8] Sun Microsystems Inc., *Java Card 2.1 API Specification*, 1999, http://java.sun.com/products/javacard/javacard21.html

[9] Anderson, R. Kuhn, M., *Tamper Resistance - a Cautionary Note*, The Second USENIX Workshop on Electronic Commerce – Proceedings, 1996

[10] Anderson, R., Kuhn, M., *Low Cost Attacks on Tamper Resistant Devices*, Security Protocols, 5th International Workshop – Proceedings, 1997

[11] Sun Microsystems Inc., *JavaCard 2.1 Development Kit Release Notes*, 1999, http://java.sun.com/products/javacard/javacard21.html

[12] Montgomery, M., Krishna, K., *Secure Object Sharing in JavaCard*, Workshop on SmartCard Technology (Smartcard '99), USENIX, mayo 1999.

[13] Perovich, D., Rodríguez, L., Varela, M., *A Simple Methodology for Secure Object Sharing*, Gemplus Developer Conference 2001 - Proceedings, junio 2001.

[14] *Franklin Community Health Network HealthCard*, http://www.fchn.org/card/cardhome.htm

[15] *Conjunto Essencial de Informaçôes do Prontuário para Integraçâo da Informaçâo em Saúde*, http://www.datasus.gov.br/prc/datasus.htm

[16] *Cartão Nacional de Saúde*, http://www.datasus.gov.br/cartao/datasus.htm

[17] La Quey, R. E., *SML: Simplifying XML*, noviembre 1999, http://www.xml.com/pub/a/1999/11/sml/index.html

[18] Foro de discusión de SML, http://groups.yahoo.com/group/sml-dev/

[19] Perovich, D., Rodriguez, L., Varela, M., *Programación de JavaCards*, Reporte Técnico, In.Co., Facultad de Ingeniería, Universidad de la República, mayo 2001

Domicilio para correspondencia

Ing. Leonardo Rodríguez[a], lrodrigu@fing.edu.uy

Ing. Daniel Perovich [a], perovich@fing.edu.uy

Ing. Martín Varela[a], mvarela@fing.edu.uy

Dr. Luis Martínez[b]

[a] Instituto de Computación, Facultad de Ingeniería, Universidad de la Repúlica

Julio Herrera y Reissig 565, piso 5
Montevideo 11300
Uruguay
Tel: +598 (2) 711 4244
Fax: +598 (2) 711 0469

[b] Hospital de Clínicas "Dr. Manuel Quintela", Facultad de Medicina, Universidad de la República

Avenida Italia s/n
Montevideo
Uruguay
Tel.: +598 (2) 480 1222
Fax: +598 (2) 487 3182