(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 927 748**

(51) Int. CI.:

G06F 9/445 (2008.01)
G06F 9/46 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **04.09.2017 PCT/EP2017/072068**

(87) Fecha y número de publicación internacional: **08.03.2018 WO18042029**

(96) Fecha de presentación y número de la solicitud europea: **04.09.2017 E 17758567 (6)**

(97) Fecha y número de publicación de la concesión europea: **31.08.2022 EP 3507690**

(54) Título: **Optimización de huella de memoria de aplicación de tarjeta java**

(30) Prioridad:

02.09.2016 EP 16306105

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
10.11.2022

(73) Titular/es:

THALES DIS FRANCE SAS (100.0%)
6, rue de la Verrerie
92190 Meudon, FR

(72) Inventor/es:

CHAFER, SYLVAIN;
FAVREAU, VALENTIN;
GONDOWASITO, CHANDRA y
PHAN, GUILLAUME

(74) Agente/Representante:

DEL VALLE VALIENTE, Sonia

DESCRIPCIÓN

Optimización de huella de memoria de aplicación de tarjeta java

Campo técnico

La presente invención se refiere a un método de optimización de la huella de memoria de aplicación de tarjeta Java.

Encuentra aplicaciones, en particular, en productos de tarjeta inteligente para los cuales existe la tendencia de almacenar cada vez más números de aplicaciones (subprogramas) a pesar de los recursos de memoria limitados.

Técnica relacionada

Se podrían seguir los enfoques descritos en esta sección, pero no son necesariamente enfoques que se hayan concebido o seguido previamente. Por lo tanto, salvo que se indique lo contrario en la presente memoria, los enfoques descritos en esta sección no constituyen el estado de la técnica de las reivindicaciones de esta solicitud y no se admiten como estado de la técnica mediante su inclusión en esta sección.

La tecnología de tarjetas Java se introdujo en 1996 y ahora se utiliza ampliamente en el dominio de las tarjetas inteligentes (notablemente para tarjetas SIM o tarjetas de ATM). Permite que las aplicaciones (subprogramas) basadas en java se ejecuten en tarjetas inteligentes y otros dispositivos similares que tienen recursos de memoria limitados. Estas aplicaciones usualmente no tienen una gran huella de memoria cuando se toman individualmente. Sin embargo, dado que el mercado tiende demandar el almacenamiento de más y más de estas aplicaciones en tarjetas de java individuales, apremia a los investigadores a buscar soluciones para reducir el espacio de memoria consumido por cada aplicación.

Los diferentes enfoques considerados hasta ahora implican en su totalidad modificar el propio código de aplicación para reducir su huella de memoria. La optimización del código puede realizarse a mano, refactorizando el código de bytes o utilizando un código de bytes propietario para reducir la huella de memoria. Sin embargo, se establece que dicha optimización esté limitada en términos de eficiencia, y la reducción de la huella de memoria que actualmente se logra de esta manera no es sustancial. Esto exige nuevas formas de implementar una reducción del espacio de memoria requerido para almacenar aplicaciones.

Un planteamiento alternativo puede basarse, según las realizaciones de la presente invención, en la observación de que una vez que se ha configurado una aplicación, el código binario que implementa la personalización de esta aplicación se vuelve inútil y, sin embargo, sigue usando algo de espacio de memoria en la tarjeta. La idea detrás de la invención es finalmente eliminar a continuación, al menos parte del código binario asociado con la personalización de un subprograma de tarjeta Java para liberar espacio de memoria.

La solicitud de patente US-2011/126183 describe la carga de una aplicación personalizada en una tarjeta inteligente. La aplicación y los datos de personalización se proporcionan en un servidor que está conectado a un lector de tarjetas. Dicha aplicación se cargará desde el servidor en la tarjeta inteligente, mientras que dicha aplicación se personalizará con dichos datos de personalización. Además, dichos datos de personalización se almacenarán en una memoria no volátil de respaldo dentro de la tarjeta inteligente o dentro del lector de tarjetas.

La patente europea EP-1936574 describe un método de personalización para una aplicación en una tarjeta inteligente, respetando el método la especificación de tarjeta Java. Según la tecnología de tarjeta Java, implementar una aplicación en una tarjeta inteligente pasa por un estado de carga, un estado de instalación y un estado de personalización. Normalmente, la personalización se produce cuando la aplicación ya está instalada. En la solicitud se propone tener un componente personalizado en el archivo de CAP, que contiene la información de personalización. En cuanto se instala la aplicación, también se personaliza.

El documento US-2008/268811 describe un método para personalizar un dispositivo a través de una red inalámbrica que comprende: recibir una solicitud para personalizar el dispositivo con información específica de usuario; preparar un archivo de personalización único para su uso con el dispositivo en respuesta a la solicitud de personalizar el dispositivo; proporcionar un archivo de descripción al dispositivo a través de la red inalámbrica, comprendiendo el archivo de descripción un URL que apunta a un entorno de alojamiento web que contiene una aplicación; proporcionar la aplicación al dispositivo a través de la red inalámbrica en respuesta a un usuario que contacta con el entorno de alojamiento web usando el URL; y proporcionar el archivo de personalización único al dispositivo a través de la red inalámbrica en respuesta a una solicitud del dispositivo para recuperar el archivo de personalización.

Sumario

Para lograr este objetivo, la presente invención propone un método para la optimización de la huella de memoria de una aplicación de tarjeta Java que permite reducir una huella de memoria de subprograma eliminando parte o la totalidad del código asociado a su personalización una vez que se ha completado dicha personalización.

Más específicamente, según un primer aspecto, se propone un método para reducir la huella de memoria de una aplicación de tarjeta Java según la reivindicación independiente 1.

El método anterior permite finalmente eliminar el código binario asociado con la personalización de un subprograma de tarjeta Java una vez que se han completado todas las etapas asociadas con su personalización. Por consiguiente, permite reducir la huella de memoria asociada con esta aplicación en la tarjeta.

Por lo tanto, el método propuesto se basa en la separación por adelantado del código relacionado con la personalización del resto del código. Permite que este código realice la personalización de una aplicación instalada desde un paquete principal mientras que el mismo se incluye e instala desde un paquete separado especializado para la personalización, especialmente el paquete de Card Personalization Specifications (especificaciones de personalización de tarjeta - CPS), por ejemplo. De esta manera, el paquete de CPS y todo el código inherente a la personalización (denominado “el código de personalización” en lo sucesivo) se pueden eliminar una vez que se han completado todas las etapas de personalización.

Dicho de otro modo, el método hace posible superar las limitaciones establecidas por las reglas de tiempo de ejecución de tarjeta Java que se implementan mediante un servidor de seguridad para proteger los subprogramas evitando una instancia de subprograma de acceso a objetos creados en un paquete de subprograma diferente. Para ello, las realizaciones del método de la presente invención pueden comprender modificar el contexto de servidor de seguridad de un objeto creado por una instancia, especialmente el subprograma de CPS, contenida en el paquete de CPS. Asignando al objeto creado a partir del subprograma de CPS (llamado objeto de CPS en adelante) el mismo contexto de servidor de seguridad que el subprograma principal, las realizaciones del método pueden permitir que el código de personalización contenido en el objeto de CPS acceda al subprograma principal y ejecute el código requerido por la personalización de la aplicación sin ser bloqueado por el servidor de seguridad.

Además de esto, las realizaciones del método pueden comprender asimismo hacer que los objetos creados durante el proceso de personalización pertenezcan al subprograma principal. Estos objetos pertenecerían, de cualquier otra manera, al subprograma de CPS, dado que se han creado a partir del mismo. En tal configuración, se eliminarían, a continuación, junto con dicho paquete de CPS si tuviera que eliminarse este paquete. Introduciendo algunas modificaciones de la propiedad del objeto de CPS creado a partir del subprograma de CPS del paquete de CPS, las realizaciones del método superan esta limitación. La propiedad del objeto de CPS puede cambiarse hacia adelante y hacia atrás desde el subprograma de CPS al subprograma principal y recíprocamente, de modo que una vez que se ha completado la personalización, el objeto de CPS pertenece a la instancia de subprograma de CPS y puede eliminarse junto con dicho paquete de CPS, mientras que todos los objetos personalizados dentro del subprograma principal siguen perteneciendo al subprograma principal.

En realizaciones del método, tomadas solas o en combinación:

- el paquete principal y el segundo paquete pueden utilizar una interfaz compartible de tarjeta Java para ser capaces de compartir recursos entre sí;

- el paquete principal puede asociarse a una fase de transacción y el segundo paquete se asocia a una fase de personalización, respectivamente;

- el segundo paquete es un paquete de especificación de personalización de tarjeta, CPS, asociado a la fase de personalización, conteniendo dicho paquete de CPS un subprograma de CPS especializado para la creación de un objeto de CPS en la memoria de la tarjeta Java,

- el paquete de CPS puede tener un contexto de servidor de seguridad y una propiedad, y configurarse para ejecutar un código de personalización utilizado para la personalización de la aplicación principal;

- el método además puede comprender cambiar el contexto de servidor de seguridad del objeto de CPS asignando al mismo un contexto de protección de seguridad del subprograma principal y cambiar la propiedad del objeto de CPS desde una propiedad de instancia de subprograma de CPS a una propiedad de instancia de subprograma principal;

- el método además puede comprender gestionar la personalización de la aplicación principal teniendo la aplicación principal acceso al objeto de CPS y ejecutar el código de personalización que inicializa los datos de aplicación principal.

- el método además puede comprender:

- cambiar de vuelta la propiedad del objeto de CPS desde la propiedad de subprograma principal a la propiedad de subprograma de CPS; y,

- borrar el paquete de CPS;

- el contexto de servidor de seguridad del objeto de CPS se cambia mediante una primera interfaz de programación de aplicación, API;

- la propiedad del objeto de CPS se cambia y/o cambia de vuelta mediante la primera API, o mediante una segunda API distinta de dicha primera API;

- la primera API y/o la segunda API, se utilizan para cambiar la propiedad y/o el contexto de servidor de seguridad del objeto de CPS, respectivamente, son proporcionadas por la plataforma de tarjeta Java o se crean específicamente;

- la primera API y/o la segunda API utilizadas para cambiar la propiedad y el contexto de servidor de seguridad del objeto de CPS pueden borrarse de la memoria de la tarjeta Java después de que se haya realizado el cambio de vuelta de propiedad del objeto de CPS desde el subprograma principal al subprograma de CPS;

- en una variante, la primera API y/o la segunda API utilizadas para cambiar la propiedad y el contexto de servidor de seguridad del objeto de CPS pueden quedar inutilizables después de haber realizado el cambio de vuelta de la propiedad del objeto de CPS desde el subprograma principal al subprograma de CPS;

- el contexto de servidor de seguridad del objeto de CPS se cambia implementando un parámetro específico en el comando de instalación estándar utilizado para instalar el subprograma de CPS y definido por la especificación de la tarjeta de plataforma global;

- la aplicación principal que se instala desde el paquete principal es una aplicación de Europay Mastercard Visa (EMV);

- el paquete principal se carga en la memoria de solo lectura (ROM) de la tarjeta Java;

- en una variante, el paquete de CPS se carga en la memoria no volátil (NVM) de la tarjeta Java y, por último,

- el método además puede comprender dividir la aplicación de tarjeta Java en el paquete principal y el segundo paquete.

Un segundo aspecto de la invención se refiere a un producto de programa informático que comprende según la reivindicación independiente 15.

Por último, un tercer aspecto de la invención propone un dispositivo para reducir la huella de memoria de una aplicación de tarjeta Java, que comprende medios para llevar a cabo todas las etapas del método según el primer aspecto o una cualquiera de las realizaciones anteriores de dicho método.

Breve descripción de los dibujos

Las realizaciones de la presente invención se ilustran a manera de ejemplo y, no de forma excluyente, en las figuras de los dibujos adjuntos, en las que los números de referencia similares se refieren a elementos similares, y en las que:

- La Figura 1 es un diagrama esquemático que muestra una configuración básica para la instalación de una estructura de subprograma de tarjeta Java según la técnica anterior;

- La Figura 2 es un diagrama esquemático que ilustra el problema técnico resuelto por la invención; y,

- La Figura 3 es un diagrama de bloques compuesto y un diagrama de flujo que ilustran las etapas llevadas a cabo para la implementación de realizaciones del método propuesto para la optimización de la huella de memoria de una aplicación de tarjeta Java.

Descripción de realizaciones preferidas

Haciendo referencia en primer lugar a la Figura 1, se muestra en la misma una descripción esquemática de una configuración básica para la instalación de una estructura de subprograma de tarjeta Java según la técnica anterior.

Un único paquete, que se denominará el paquete principal 11 en lo sucesivo, contiene todos los elementos requeridos para la instalación y las funcionalidades de una aplicación, de aquí en adelante denominado el subprograma principal 12. Este subprograma principal 12 puede ser, por ejemplo, una aplicación de Europay Mastercard Visa (EMV). En el contexto de la invención, una aplicación de este tipo está concebida para crearse/instalarse en la tarjeta Java. Para este fin, el paquete principal 11 se carga en la tarjeta Java y se ejecuta en la misma.

Algún código 13 contenido en el subprograma principal 12 está especializado para la personalización de dicho subprograma principal 12. Este código 13 de personalización, entonces, también está contenido en el paquete

principal. Dicho código 13 está adaptado para crear los objetos de datos de subprograma principal (tales como instancias de clases o tipos de matrices) y además está adaptado para inicializar los valores de cada campo de cada objeto de datos durante el proceso de personalización del subprograma principal. En lo sucesivo, esto se denomina "fase de personalización".

Haciendo ahora referencia a la Figura 2, se muestra en la misma una descripción esquemática de una configuración según realizaciones, en donde el paquete para la instalación del subprograma de tarjeta Java se divide en dos paquetes de subprogramas distintos: un paquete principal 11 y un paquete 21 de especificación de personalización de tarjeta (CPS).

Las realizaciones pueden, por ejemplo, basarse en la implementación de un diseño de múltiples paquetes con diferentes paquetes capaces de compartir y enlazar sus recursos. Una forma de lograr esta asociación funcional entre el paquete principal 11 que contiene el subprograma principal 12 y el paquete 21 de CPS que contiene un subprograma 25 de CPS para permitir la compartición de recursos anterior, puede ser utilizar una interfaz compartible de tarjeta Java. De esta manera, se puede acceder a los objetos de interfaz compartibles (SIO) de una instancia de subprograma en un paquete desde otra instancia de subprograma en otro paquete. En particular, una vez que se ha creado dicha instancia 12 de subprograma principal, la instancia 12 de subprograma principal puede solicitar una instancia 32 de objeto de CPS creada por el subprograma 25 de CPS y que contiene el código 13 de personalización.

La flecha 22 que se ve en la Figura 2 ilustra ese último hecho. La instancia 12 de subprograma principal creada a partir del paquete principal llama al código 13 de personalización incluido en el objeto 32 de CPS en el paquete de CPS. La flecha 23 y la cruz 24 ilustran que el código de personalización no puede acceder a la instancia 12 de subprograma principal debido a un servidor de seguridad de la tarjeta Java. De hecho, el servidor de seguridad de la tarjeta Java prohíbe que los objetos creados en un paquete de subprograma accedan a los objetos creados en otro paquete de subprograma. Por lo tanto, prohíbe que cualquier objeto creado en el paquete de CPS acceda a cualquier objeto del paquete principal. Un experto en la técnica apreciará que, en esa situación, el código de personalización no puede ejecutarse y la fase de personalización no puede lograrse correctamente.

Según las reglas reales de la tarjeta Java, por defecto, las instancias de objetos creadas a partir de una instancia de subprograma pertenecen a esta instancia de subprograma, lo que significa que la propiedad de estos objetos es la de la instancia de subprograma. Como consecuencia, los objetos personalizados al ejecutar el código de personalización incluido en el objeto de CPS creado a partir del subprograma de CPS, tienen la misma propiedad que la del subprograma de CPS. Dado que el subprograma de CPS está en el paquete de CPS, se deduce que, si el paquete de CPS se eliminara, todos los objetos personalizados que se crearon a partir de la instancia de objeto de CPS se eliminarían por completo. Este efecto se acumula con el hecho de que, en ese caso, el subprograma principal no es capaz de acceder a los objetos propiedad del subprograma de CPS debido al servidor de seguridad.

El método propone una solución para superar estas limitaciones cambiando no solo el contexto de servidor de seguridad, sino también la propiedad del objeto de CPS creado por el subprograma de CPS contenido en el paquete de CPS.

Las realizaciones del método propuesto se describirán ahora con referencia a la Figura 3.

Haciendo referencia a la Figura 3, se muestra en la misma un diagrama de bloques compuesto y un diagrama de flujo que ilustran diferentes tareas realizadas según realizaciones del método propuesto para la optimización de la huella de memoria de una aplicación de tarjeta Java. El método se basa básicamente en una separación previa del subprograma de tarjeta Java en un paquete principal y un paquete de CPS.

Más específicamente, el método comprende lo siguiente.

En 31a, en la memoria de una tarjeta Java se carga un paquete principal 11 que contiene un subprograma principal 12, que está concebido para instalarse en la tarjeta y personalizarse. En paralelo, en 31b, en la memoria de la misma tarjeta Java además se carga un paquete 21 de especificación de personalización de tarjeta (CPS), que contiene el subprograma 25 de CPS, dentro del cual se incluye el código 13 de personalización requerido por la personalización de dicha aplicación principal. En la Figura 3, estas etapas de carga se ilustran mediante las flechas 31a y 31b, respectivamente. En un ejemplo, el paquete principal 11 puede cargarse en la memoria de solo lectura (ROM) de la tarjeta Java y el paquete de CPS puede cargarse en la memoria no volátil (NVM) de dicha tarjeta Java. Este ejemplo es ventajoso dado que, como se describirá con más detalle a continuación, el código del paquete de CPS puede borrarse en última instancia, pero no pretende limitar el alcance de la invención. En otros ejemplos, ambos paquetes pueden cargarse en el espacio de memoria respectivo de la misma memoria física.

La creación del objeto 32 de CPS mediante el subprograma 25 de CPS en el paquete 21 de CPS cargado en la memoria de la tarjeta se ilustra mediante la flecha 33 en la Figura 3. Debe observarse que el objeto 32 de CPS tiene la propiedad del subprograma 25 de CPS y el contexto de seguridad del paquete 21 de CPS. Esto es así porque el objeto 32 de CPS se ha creado a partir de la instancia 25 de subprograma de CPS que, en sí misma, se creó a partir del paquete 21 de CPS. Una vez creado, el subprograma principal 12 llama al objeto de CPS para gestionar su

personalización. Esto es posible dado que, como se ha mencionado anteriormente, el paquete principal 11 y el paquete 21 de CPS están asociados (por ejemplo, a través de la utilización apropiada de la interfaz compartible de tarjeta Java) y, por lo tanto, pueden compartir recursos.

Sin embargo, el código 13 de personalización dentro del objeto 32 de CPS no puede acceder al subprograma principal 12. De hecho, como se describió previamente con referencia a la Figura 2, si el objeto 32 de CPS intentase acceder al subprograma principal 12, como se ilustra mediante la flecha 37, a continuación sería bloqueado por el servidor 41 de seguridad como se ilustra mediante la cruz 24.

Para evitar esta situación, el método prosigue, a continuación, con el cambio del contexto de servidor de seguridad y de la propiedad del objeto 32 CPS como se ilustra mediante la flecha 34. El cambio de propiedad puede realizarse, por ejemplo, en el subprograma de CPS antes de que el subprograma principal obtenga el objeto de CPS o en el subprograma principal una vez que dicho subprograma principal obtenga el objeto de CPS. En ambos casos, este cambio se produce antes de la ejecución del código de personalización mencionado de aquí en adelante.

En la Figura 3, el objeto de CPS con contexto de servidor de seguridad y propiedad cambiados está representado por el bloque 32'. Este cambio puede lograrse, por ejemplo, ejecutando una o más interfaces de programación de aplicación (API) que se especializa o especializan para cambiar el contexto de servidor de seguridad y/o la propiedad de un objeto dentro de la tarjeta Java. Por ejemplo, el contexto de servidor de seguridad puede cambiarse mediante una primera API, y la propiedad puede cambiarse mediante esta primera API, o mediante una segunda API distinta de dicha primera API. En un ejemplo, esta o estas API pueden ser proporcionadas, por ejemplo, por la plataforma de tarjeta Java o pueden crearse específicamente.

A continuación, el proceso continúa con la ejecución del código de personalización, del objeto 32' de CPS, para personalizar los objetos de datos de la aplicación principal 12, como se ilustra mediante la flecha 35. Debe observarse que el objeto 32' de CPS puede acceder al subprograma principal 12 para ese propósito, porque se ha producido el cambio de contexto de servidor de seguridad ilustrado mediante la flecha 34. En la Figura 3, la aplicación principal 12 con objetos de datos personalizados se representa mediante el bloque 12'.

Después de haberse completado la personalización del subprograma principal, la propiedad del objeto 32' de CPS se cambia de vuelta a la instancia de subprograma de CPS como se ilustra mediante la flecha 39. En la Figura 3, el objeto de CPS con propiedad cambiada de vuelta a la instancia de subprograma de CPS está representado por el bloque 32".

Finalmente, una vez que la propiedad del objeto de CPS ha cambiado de vuelta al subprograma de CPS, el paquete 21 de CPS puede borrarse como se ilustra mediante la flecha 40. Como resultado, también se borra el código de personalización, dado que es parte del código de paquete de CPS.

Para resumir, las realizaciones del método como se ha descrito anteriormente permiten personalizar una aplicación de tarjeta Java mientras se elimina finalmente todos los elementos de código relacionados con su personalización. Por consiguiente, esto proporciona de forma ventajosa una reducción de la huella de memoria para el subprograma principal.

Por tanto, el método permite realizar la personalización de una aplicación creada/instalada desde un paquete a partir del objeto creado/instalado desde un paquete diferente. Además, el paquete posterior puede eliminarse después de haberse completado el proceso de personalización sin eliminar ninguno de los elementos requeridos por el subprograma principal para su funcionamiento.

Se espera que el método permita reducir la huella de memoria de aplicación significativamente y presenta varias ventajas adicionales:

- Es fácil de implementar en aplicaciones ya existentes;
- En términos de seguridad de la tarjeta, ya no es posible ejecutar código de personalización “parasitario” para modificar la aplicación porque el código de personalización ya no existe;
- La reducción del espacio de memoria requerido para una aplicación ofrece la posibilidad de utilizar tarjetas de java con menos memoria, en consecuencia, productos más baratos;
- Recíprocamente, es posible instalar más aplicaciones en la memoria de una única tarjeta de java, aumentando, por tanto, la productividad de una única tarjeta.

La presente invención también puede integrarse en un producto de programa informático, que comprenda todas las características que habilitan la implementación de los métodos descritos en la presente memoria, y que, cuando se carga en un sistema de procesamiento de información, sea capaz de llevar a cabo estos métodos. Los medios de programa informático o el programa informático en el presente contexto significan cualquier expresión, en cualquier lenguaje, código o notación, de un conjunto de instrucciones concebidas para hacer que un sistema tenga una capacidad de procesamiento

de información para realizar una función particular, ya sea directamente o después de la conversión a otro lenguaje. Dicho programa informático puede almacenarse en un soporte legible por ordenador o máquina que permite que los datos, instrucciones, mensajes o paquetes de mensajes y otra información legible por máquina se lean desde el soporte. El soporte legible por ordenador o máquina puede incluir memoria no volátil, tal como ROM, memoria flash, memoria de unidad de disco, CD-ROM y otro almacenamiento permanente. De forma adicional, un soporte legible por ordenador o máquina puede incluir, por ejemplo, almacenamiento volátil, tal como RAM, memorias intermedias, memoria caché y circuitos de red. Además, el soporte legible por ordenador o máquina puede comprender información legible por ordenador o máquina en un soporte de estado transitorio, tal como un enlace de red y/o una interfaz de red, incluyendo una red por cable o una red inalámbrica, que permiten que un dispositivo lea tal información legible por ordenador o máquina.

Las expresiones tales como ‘‘comprender’’, ‘‘incluir’’, ‘‘incorporar’’, ‘‘contener’’, ‘‘ser’’, ‘‘estar’’ y ‘‘tener’’ deben entenderse de manera no exclusiva cuando se interpreta la descripción y sus reivindicaciones asociadas, especialmente entendidas para permitir que también estén presentes otros artículos o componentes que no se definen explícitamente. La referencia al singular también debe entenderse como una referencia al plural y viceversa.

Se estipula que los signos de referencia en las reivindicaciones no limitan el alcance de las reivindicaciones, sino que simplemente se insertan para mejorar la legibilidad de las reivindicaciones.

REIVINDICACIONES

1. Un método para reducir la huella de memoria de una aplicación de tarjeta Java, comprendiendo el método cargar (31a, 31b) dos paquetes distintos en la memoria de una tarjeta Java, especialmente un primer paquete principal (11) que contiene una aplicación principal y especializado para la instalación y la fase de transacción de dicha aplicación principal en la tarjeta Java, y un segundo paquete, capaz de compartir recursos con el paquete principal y destinado a utilizarse durante la ejecución de la fase de personalización de la aplicación principal y a borrarse después de dicha ejecución de dicha aplicación principal, estando asociado el paquete principal a una fase de transacción y estando asociado el segundo paquete a una fase de personalización, respectivamente, siendo el segundo paquete un paquete (21) de especificación de personalización de tarjeta, CPS, asociado a la fase de personalización, conteniendo dicho paquete (21) de CPS un subprograma (25) de CPS especializado para la creación de un objeto (32) de CPS en la memoria de la tarjeta Java, en donde el método además comprende cambiar (34) el contexto de servidor de seguridad del objeto de CPS asignando al mismo un contexto de servidor de seguridad del subprograma principal, cambiar la propiedad del objeto de CPS desde la propiedad de instancia de subprograma de CPS a una propiedad de instancia de subprograma principal, cambiar de vuelta (39) la propiedad del objeto de CPS desde la propiedad de subprograma principal a la propiedad de subprograma de CPS; y, borrar (40) el paquete de CPS.

2. El método de la reivindicación 1, en donde el paquete principal y el segundo paquete utilizan una interfaz compartible de tarjeta Java para ser capaces de compartir recursos entre sí.

3. El método de la reivindicación 1, en donde el paquete de CPS tiene un contexto de servidor de seguridad y una propiedad, y está configurado para ejecutar un código de personalización utilizado para la personalización de la aplicación principal.

4. El método de una cualquiera de las reivindicaciones 1 a 3, que además comprende gestionar (35) la personalización de la aplicación principal teniendo la aplicación principal acceso al objeto de CPS y ejecutar el código de personalización que inicializa datos de aplicación principal.

5. El método de una cualquiera de las reivindicaciones 1 a 4, en donde el contexto de servidor de seguridad del objeto de CPS se cambia mediante una primera interfaz de programación de aplicación, API.

6. El método de una cualquiera de las reivindicaciones 1 a 5, en donde la propiedad del objeto de CPS se cambia y/o cambia de vuelta mediante la primera API o mediante una segunda API distinta de dicha primera API.

7. El método de una cualquiera de las reivindicaciones 5 y 6, en donde la primera API y/o la segunda API, utilizadas para cambiar la propiedad y/o el contexto de servidor de seguridad del objeto de CPS, respectivamente, se proporcionan mediante la plataforma de tarjeta Java o se crean específicamente.

8. El método de una cualquiera de las reivindicaciones en donde la primera API y/o la segunda API utilizadas para cambiar la propiedad y el contexto de servidor de seguridad del objeto de CPS se borra o borran de la memoria de tarjeta Java después de que se haya realizado el cambio de vuelta de la propiedad de objeto de CPS desde el subprograma principal al subprograma de CPS.

9. El método de una cualquiera de las reivindicaciones 5 a 7, en donde la primera API y/o la segunda API utilizadas para cambiar la propiedad y el contexto de servidor de seguridad del objeto de CPS queda o quedan inutilizables después de haber realizado el cambio de vuelta de la propiedad de objeto de CPS desde el subprograma principal al subprograma de CPS.

10. El método de una cualquiera de las reivindicaciones 1 a 9, en donde el contexto de servidor de seguridad del objeto de CPS se cambia implementando un parámetro específico en el comando de instalación estándar utilizado para instalar el subprograma de CPS y definido por la especificación de la tarjeta de plataforma global.

11. El método de una cualquiera de las reivindicaciones 1 a 10, en donde la aplicación principal instalada desde el paquete principal es una aplicación de Europay Mastercard Visa (EMV).

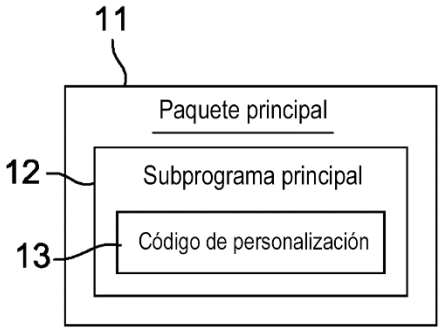
12. El método de una cualquiera de las reivindicaciones 1 a 11, en donde el paquete principal se carga en la memoria de solo lectura (ROM) de la tarjeta Java.

13. El método de una cualquiera de las reivindicaciones 1 a 11, en donde el paquete de CPS se carga en la memoria no volátil (NVM) de la tarjeta Java.

14. El método de una cualquiera de las reivindicaciones 1 a 13, que además comprende dividir la aplicación de tarjeta Java en el paquete principal y el segundo paquete.

15. Un producto de programa informático que comprende una o más secuencias de instrucciones almacenadas que son accesibles para un procesador y que, cuando son ejecutadas por el procesador, hacen que el procesador realice las etapas de cualquiera de las reivindicaciones 1 a 14.

16. Un dispositivo para reducir la huella de memoria de una aplicación de tarjeta Java, que comprende medios para llevar a cabo todas las etapas del método de una cualquiera de las reivindicaciones 1 a 14.



ESTADO DE LA TÉCNICA

Fig. 1

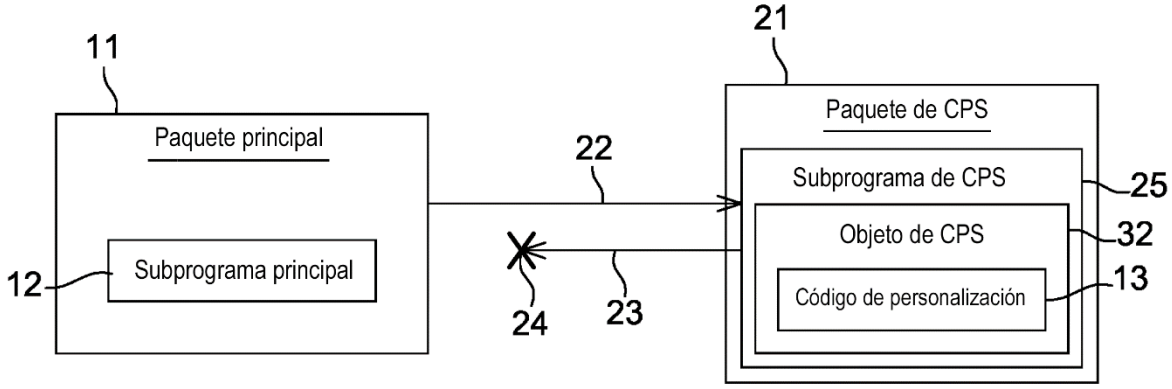


Fig. 2

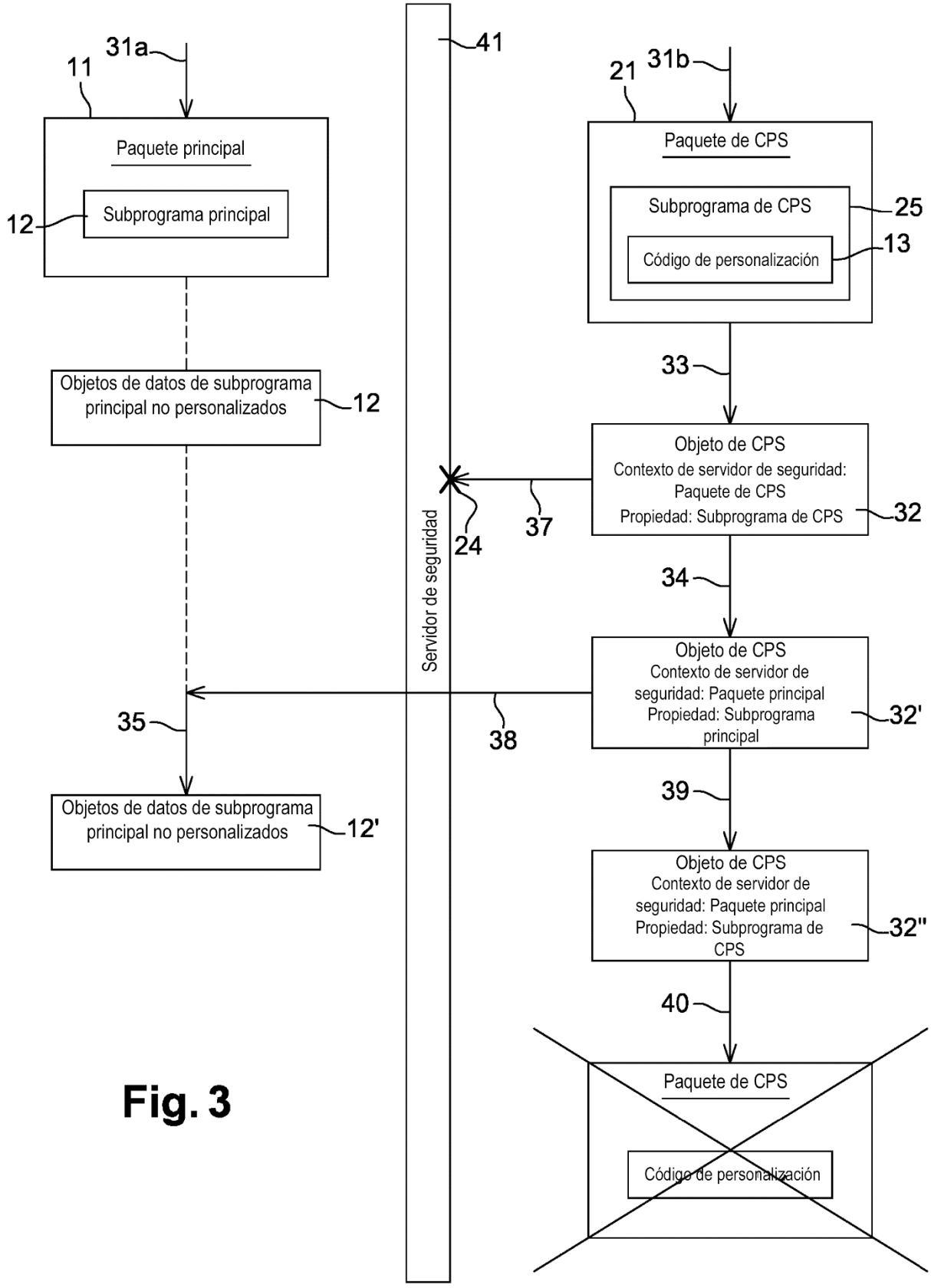

31a
11
Paquete principal
12
Subprograma principal
Objetos de datos de subprograma principal no personalizados
12
35
Objetos de datos de subprograma principal no personalizados
12'
41
Servidor de seguridad
24
37
38
31b
21
Paquete de CPS
Subprograma de CPS
25
Código de personalización
13
33
Objeto de CPS
Contexto de servidor de seguridad: Paquete de CPS
Propiedad: Subprograma de CPS
32
34
Objeto de CPS
Contexto de servidor de seguridad: Paquete principal
Propiedad: Subprograma principal
32'
39
Objeto de CPS
Contexto de servidor de seguridad: Paquete principal
Propiedad: Subprograma de CPS
32"
40
Paquete de CPS
Código de personalización

Fig. 3