(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 376 191**

(51) Int. CI.:
G06F 9/42 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Número de solicitud europea: **01939519 .3**
(96) Fecha de presentación: **25.05.2001**
(97) Número de publicación de la solicitud: **1301854**
(97) Fecha de publicación de la solicitud: **16.04.2003**

(54) Título: **MÉTODO Y APARATO PARA CREAR MÉTODOS NATIVOS EFICACES QUE EXTIENDEN UN INTÉRPRETE DE CÓDIGOS DE BYTE.**

(30) Prioridad:
25.05.2000 US 207482 P

(45) Fecha de publicación de la mención BOPI:
09.03.2012

(45) Fecha de la publicación del folleto de la patente:
09.03.2012

(73) Titular/es:
Oracle America, Inc.
500 Oracle Parkway
Redwood City, CA 94065, US

(72) Inventor/es:
LONG, Dean, R., E.;
PLUMMER, Christopher, J. y
FRESKO, Nedim

(74) Agente/Representante:
Curell Aguilá, Mireia

DESCRIPCIÓN

Método y aparato para crear métodos nativos eficaces que extienden un intérprete de códigos de byte.

Referencia cruzada a solicitudes relacionadas

La presente solicitud reivindica prioridad con respecto a la solicitud de patente provisional presentada el 25 de mayo de 2000, número de solicitud 60/207.482, titulada “Method and Apparatus for Writing Time and Space Efficient Native Methods that Extend A Byte-Code Interpreter.”

Antecedentes de la invención

1. Campo de la invención

La presente invención se refiere en general a software informático y sistemas operativos. Más específicamente, se refiere a una interfaz para una máquina virtual Java que posibilita un uso más eficaz de componentes en la máquina virtual y permite una invocación eficaz de métodos nativos.

2. Descripción de la técnica relacionada

La necesidad de instalar un sistema Java o máquina virtual Java (“JVM”) en sistemas de consumo e integrados está creciendo. En el futuro, es más probable que los dispositivos, electrodomésticos, cajas de adaptación de televisores y similares contengan algún tipo de implementación del lenguaje Java. Como es sabido en el sector, una implementación típica del lenguaje Java es una JVM, que contiene un bucle intérprete (al que se hace referencia también como intérprete de códigos de bytes) que ejecuta repetidamente códigos de byte. El bucle intérprete se escribe típicamente en un lenguaje de bajo nivel, tal como el lenguaje de programación C, y ejecuta una representación intermedia, basada en pilas, del lenguaje Java denominada códigos de byte Java. Un sistema Java, tal como el presente en un dispositivo integrado o de consumo, contiene típicamente un conjunto de bibliotecas escritas en Java y un conjunto de métodos nativos, escritos en un lenguaje tal como el C. Para que los códigos de byte Java llamen a métodos nativos, la máquina virtual proporciona una interfaz de métodos nativos. Esta interfaz nativa es responsable de localizar un método nativo y de transferir un conjunto de argumentos de método desde la pila de códigos de byte Java a una pila nativa (a la que se hace referencia también como pila de C) antes de la ejecución del método nativo. La interfaz es responsable también de tomar un valor de retorno de un método nativo y de devolverlo a la pila de Java para su uso posterior por códigos de byte Java. Esencialmente, la interfaz de métodos nativos toma argumentos de la pila de Java y los coloca en la pila de C. Una interfaz de métodos nativos común para el Java es la Interfaz Nativa de Java o JNI.

Cuando se usan interfaces actuales de métodos nativos en sistemas con recursos limitados de CPU y de memoria surgen otros problemas. Uno de los problemas es que muchos métodos nativos críticos en cuanto al rendimiento se deben ejecutar frecuentemente. No obstante, el “protocolo” de la interfaz de métodos nativos consume un tiempo excesivo en la ejecución. Adicionalmente, con los retornos del método nativo para el bucle intérprete, es necesario desapilar un marco de la pila de Java. Otro problema es la cantidad de espacio utilizado por la pila nativa o pila de C. Para llamadas de métodos especiales, a saber, invocaciones de método debidas a la reflexión de Java (Method.invoke() y ejecución de constructores para Class.newInstance()) y la ejecución del inicializador estático <clinit> de una clase en el primer uso de la clase, se produce un problema de uso de la pila nativa. Las funciones C que gestionan la invocación del método de reflexión y la invocación del método <clinit> llaman de manera recursiva al bucle intérprete (Java) para ejecutar el método objetivo especial. Esto significa que la pila nativa tiene un marco de intérprete nuevo apilado en la misma. Este proceso puede continuar de manera indefinida, desde el bucle intérprete al código nativo, de vuelta al bucle intérprete, y así sucesivamente. Este ciclo de llamadas recursivas puede consumir potencialmente excesivos recursos de la pila de C y ciclos del reloj del procesador puesto que la pila de C está típicamente pre-asignada y se hace suficientemente grande como para evitar el desbordamiento en un escenario del peor caso. Esta tara designada a pre-asignar una memoria de pila de C para adecuarse a un escenario del peor caso del uso de la pila es significativa para dispositivos de consumo e integrados que ejecutan una JVM que tiene recursos restringidos de memoria y de procesador. Por lo tanto, si al uso de la pila de C del peor caso contribuyen llamadas C recursivas, y dichas llamadas recursivas se pueden reducir o eliminar, entonces se puede reducir potencialmente el uso de la pila C del peor caso, permitiendo de este modo la reducción del tamaño de pilas C pre-asignadas.

Se requiere una interfaz nativa de función especializada que permita que una JVM minimice la cantidad de recursos de memoria y de procesador que consume la JVM. En ciertos casos, la interfaz nativa de función especializada usada conjuntamente con el bucle intérprete puede eliminar potencialmente la recursividad de C. Adicionalmente, resultaría deseable extender de forma eficaz el bucle intérprete en una JVM sin añadir uno o más códigos de byte nuevos y permitiendo que ciertos métodos nativos manipulen o accedan directamente el estado de la JVM. Más específicamente, se requiere una interfaz nativa que no necesite apilamiento o desapilamiento de marcos Java, que no necesite la ordenación de argumentos y resultados de método entre las pilas de Java y nativa, y que no necesite retrollamadas de funciones caras con el fin de permitir que el método nativo acceda a datos internos de la JVM.

"Compiling Java just in time", Micro, IEEE, v17n13, junio de 1997, páginas 36 a 43, describe la llamada a métodos nativos en un entorno Java. Entre el intérprete y el código nativo se inserta una rutina intermediaria (*stub routine*). Un método nativo lee de forma intermediaria los argumentos entrantes de la pila de Java y los coloca en registros o en la pila del hilo de acuerdo con la convención de llamada nativa. Cuando el método nativo devuelve valores de retorno, la rutina intermediaria almacena el valor de retorno de vuelta en la pila de Java.

Se describen métodos, sistemas y soportes legibles por ordenador para ejecutar un método nativo en un entorno virtual Java.

En un aspecto de la invención, se proporciona un método de ejecución de un método nativo en una máquina virtual Java que incluye una pila de Java, comprendiendo el método: determinar si un método nativo va a ser gestionado por una primera interfaz nativa o una de una pluralidad de otras interfaces nativas; si el método nativo va a ser gestionado por la primera interfaz nativa, invocar al método nativo y posibilitar que el método nativo acceda a un estado interno de la máquina virtual Java mediante la obtención de un puntero de función desde un bloque de método, la invocación a la función del método nativo y el traslado, a la función del método nativo, de datos que permiten el acceso a una máquina virtual Java, incluyendo dichos datos un puntero a argumentos en la pila de Java; ejecutar el método nativo en la máquina virtual Java; y almacenar, por medio del método nativo, sus resultados en la pila de Java, y modificar un puntero de pila de Java sobre la base de un código de retorno, ajustando de este modo el estado de la máquina virtual Java sobre la base de la ejecución del método nativo, con lo cual se minimiza la transición entre un bucle intérprete y el método nativo a través de la primera interfaz nativa.

Los métodos nativos se pueden clasificar de manera que queden capacitados para ser gestionados por una interfaz nativa especial. Al método nativo especial se le pueden pasar un puntero a la parte superior de la pila de Java y un puntero a un puntero del bloque de método. El método nativo especial puede colocar un marco de transición correspondiente a un método nuevo en una pila en la máquina virtual. El método nativo especial también puede apilar argumentos asociados al marco de transición en la pila, y se puede retornar un código de resultado. La pila puede ser una pila de Java y el código de resultado indica que se ha apilado un marco de transición nuevo en la pila. Se puede minimizar la recursividad de la pila en la máquina virtual y se puede reducir la memoria utilizada por una pila mientras se está ejecutando la máquina virtual.

Otros aspectos de la invención proporcionan un sistema según la reivindicación 6 y un soporte legible por ordenador según la reivindicación 7.

Breve descripción de los dibujos

La invención se pondrá más claramente de manifiesto a partir de la siguiente descripción considerada conjuntamente con los dibujos adjuntos, en los cuales:

la figura 1 es un diagrama de bloques de un bucle intérprete en una máquina virtual Java de acuerdo con una forma de realización de la presente invención.

La figura 2 es un diagrama de flujo de un proceso de establecimiento y llamada en el bucle intérprete de acuerdo con una forma de realización de la presente invención.

La figura 3 es un diagrama de flujo que describe un escenario de un proceso para la ejecución de un método de SNI de acuerdo con una forma de realización de la presente invención.

La figura 4 es un diagrama de flujo de un proceso para un marco de transición nuevo de la SNI de acuerdo con una forma de realización de la presente invención.

Las figuras 5A y 5B son diagramas de flujo de un proceso para un escenario de un bloque de método nuevo de la SNI de acuerdo con una forma de realización de la presente invención.

La figura 6 es una representación esquemática de la máquina virtual de acuerdo con una forma de realización de la presente invención.

La figura 7 es un diagrama de bloques de un sistema de ordenador típico, adecuado para implementar una forma de realización de la presente invención.

Descripción detallada

A continuación se hará referencia detalladamente a una forma de realización preferida de la invención. En los dibujos adjuntos se ilustra un ejemplo de la forma de realización preferida. Aunque la invención se describirá en combinación con una forma de realización preferida, se entenderá que no se pretende limitar la invención a una forma de realización preferida. Por el contrario, se pretende abarcar alternativas, modificaciones y equivalentes en la

medida que puedan estar incluidos dentro del alcance de la invención según se define por medio de las reivindicaciones adjuntas.

En las diversas figuras se describe un método de invocación de un método nativo en una máquina virtual Java ("JVM"). En ciertos casos, una interfaz rápida de función especializada para métodos nativos usada en combinación con el bucle intérprete puede eliminar potencialmente la recursividad de la pila de C en la JVM. La interfaz se comporta como una extensión para el componente del bucle intérprete en la JVM en la medida en la que un método nativo, invocado a través de la interfaz de función especializada, puede modificar el estado del bucle intérprete, si así fuera necesario. Esto se realiza sin añadir instrucciones de códigos de byte nuevas a la JVM.

En una forma de realización específica, el conjunto de interfaces nativas usadas por la JVM se extiende o modifica para incluir una interfaz nativa especial a la que se hace referencia como SNI con fines ilustrativos. La figura 1 es un diagrama de bloques de un bucle intérprete en una máquina virtual Java de acuerdo con una forma de realización de la presente invención. Un bucle intérprete 102 es un componente de la máquina virtual Java que se describe más detalladamente en la figura 6 como componente 617 posteriormente. Un módulo de establecimiento y de llamadas de métodos nativos 104 es un módulo responsable de reaccionar a y comunicarse con métodos nativos especiales con los que se encuentra la máquina virtual Java. El módulo 104 inicia la comunicación con un método nativo, tal como los métodos 106 y 108. Un método nativo especial 106 ó 108 retorna ciertos datos a un módulo de retorno de estado 110. El método 110 acepta un valor de retorno de un método nativo especial y permite que el bucle intérprete 102 continúe con la ejecución.

La figura 2 es un diagrama de flujo de un proceso de establecimiento y de llamada en el bucle intérprete de acuerdo con una forma de realización de la presente invención. En la etapa 202, el bucle intérprete, específicamente el módulo 104, examina un bloque de método correspondiente al método con el que se encuentra la JVM en relación con un tipo de método tal como Java, JNI, o, en la presente invención, SNI. Tal como es sabido en el campo de la programación de máquinas virtuales Java, un bloque de método es una estructura de datos que representa el método y contiene punteros, número de argumentos y otros datos referentes al método. En la etapa 204, el bucle intérprete obtiene un puntero de función a partir del bloque del método.

En la etapa 206, el bucle intérprete llama al método usando el puntero de función y pasa ciertos datos. Un tipo de datos que se pasan es un puntero a argumentos en una pila de Java. Otro tipo de datos que se pasan es un puntero a un puntero de bloque de método. El uso de estos punteros se describe de forma más detallada posteriormente. Todavía otro tipo de datos que se pasan es un puntero de entorno de ejecución tal como se realiza típicamente para acceder a información referente al hilo que se está ejecutando actualmente. En la etapa 208, el método SNI da comienzo a la ejecución.

La figura 3 es un diagrama de flujo que describe un escenario de un proceso para la ejecución de un método SNI de acuerdo con una forma de realización de la presente invención. En la forma de realización descrita, el proceso muestra la instancia en la que el método SNI retorna un resultado que indica el tamaño del resultado del método. El método SNI, que se ejecuta conjuntamente con el bucle intérprete, ha almacenado su resultado en una pila de Java en la JVM. A continuación necesita informar al bucle intérprete sobre cuánto ajustar el puntero de la parte superior de la pila Java sobre la base de un código de retorno en la etapa 302. En la forma de realización descrita, los códigos de retorno posibles son nulo, sencillo o doble. El ajuste de puntero de la parte superior de la pila de Java es una etapa típica realizada en la JVM cuando se ejecutan métodos. En la etapa 304, tal como es sabido en el sector, un contador de programa en el bucle intérprete se ajusta al siguiente código de byte a ejecutar. En la etapa 306, el bucle intérprete reanuda la ejecución del código de byte.

La figura 4 es un diagrama de flujo de un proceso para un escenario de un marco de transición nuevo de la SNI de acuerdo con una forma de realización de la presente invención. En este escenario, el método SNI apila un marco de transición en una pila de Java. Los marcos de transición se usan para invocar cualquier tipo de método. En la etapa 404, el método SNI apila argumentos asociados al marco de transición en la pila de Java. En la etapa 406, el método SNI retorna al bucle intérprete un resultado al que se hace referencia, en la forma de realización descrita, como "marco de transición nuevo de la SNI". En la forma de realización descrita, el "marco de transición nuevo de la SNI" es el código de resultado retornado al bucle intérprete. En la etapa 408, la JVM da comienzo a la ejecución del método de transición al que se hace referencia en el marco de transición, fase en la cual se completa el procesado de un resultado de retorno de un "marco de transición nuevo de la SNI".

Las figuras 5A y 5B son diagramas de flujo de un proceso para un escenario de un bloque de método nuevo de la SNI de acuerdo con una forma de realización de la presente invención. En la etapa 502 de la figura 5A, un método SNI determina el bloque de método apropiado que le será indicado al bucle intérprete para que sea invocado. En la etapa 504, la interfaz almacena el puntero al bloque de método determinado en la etapa 502, que puede ser cualquier tipo de bloque de método, incluyendo un método SNI, en el argumento de puntero designado descrito anteriormente en la etapa 206. En la etapa 506, el método nativo SNI apila argumentos nuevos correspondientes a ese método en la pila de Java. Al realizar esto, se pueden sobrescribir argumentos previos en la pila de Java para el método SNI. En la etapa 508, el bloque de método nuevo y el control se retornan al bucle intérprete. En la etapa 510 de la figura 5B, el bucle intérprete comprueba el código de retorno del método nativo especial y ajusta el puntero de

la parte superior de la pila de Java sobre la base del tamaño de los argumentos en el método nuevo. En la etapa 512, el bucle intérprete se ramifica al componente que gestiona la invocación de métodos nuevos y realiza esto sobre la base del bloque de método nuevo retornado. En esta fase se completa el proceso.

La figura 6 es una representación esquemática de una máquina virtual 611 tal como la JVM 607, con la que puede trabajar el sistema de ordenador 700 de la figura 7 que se describe posteriormente. Tal como se ha mencionado anteriormente, cuando un programa de ordenador, por ejemplo, un programa escrito en el lenguaje de programación Java™, se traduce de la fuente a códigos de byte, el código fuente 601 se proporciona a un compilador de códigos de bytes 603 dentro de un entorno de tiempo de compilación 603. El compilador de códigos de bytes 609 traduce el código fuente 601 a códigos de bytes 605. En general, el código fuente 601 se traduce a códigos de bytes 605 en el momento en el que un desarrollador de software crea el código fuente 601.

Los códigos de bytes 605 en general se pueden reproducir, descargar, o distribuir de otra manera a través de una red, por ejemplo, a través de la interfaz de red 1024 de la figura 10, o se pueden almacenar en un dispositivo de almacenamiento tal como los medios de almacenamiento principales 1004 de la figura 10. En la forma de realización descrita, los códigos de bytes 603 son independientes de la plataforma. Es decir, los códigos de bytes 603 se pueden ejecutar sustancialmente en cualquier sistema de ordenador que esté haciendo funcionar una máquina virtual adecuada 611. Las instrucciones nativas formadas mediante la compilación de códigos de byte se pueden conservar para un uso posterior por la JVM. De esta manera, el coste de la traducción se amortiza durante múltiples ejecuciones para proporcionar una ventaja de velocidad del código nativo con respecto al código interpretado. A título de ejemplo, en un entorno Java™, se pueden ejecutar códigos de byte 605 en un sistema de ordenador que esté haciendo funcionar una JVM.

Los códigos de bytes 605 se suministran a un entorno de tiempo de ejecución 613 que incluye la máquina virtual 611. El entorno de tiempo de ejecución 613 se puede ejecutar en general usando un procesador tal como la CPU 1002 de la figura 10. La máquina virtual 611 incluye un compilador 615, un intérprete 617 que implementa el bucle intérprete 102 descrito en la figura 1, y un sistema de tiempo de ejecución 619. Los códigos de bytes 605 se pueden proporcionar en general o bien al compilador 615 ó bien al intérprete 617.

Cuando se proporcionan códigos de bytes 605 al compilador 615, los métodos contenidos en los códigos de bytes 605 se compilan en instrucciones máquina nativos (no mostradas). Por otro lado, cuando se proporcionan códigos de byte 605 al intérprete 617, los códigos de bytes 605 se leen hacia el intérprete 617 de uno en uno cada vez. A continuación, el intérprete 617 ejecuta la operación definida por cada código de byte a medida que cada código de byte es leído hacia el intérprete 617. En general, el intérprete 617 procesa códigos de byte 605 y realiza operaciones asociadas a códigos de byte 605 de forma sustancialmente continua.

Cuando se llama a un método, si se determina que el método se va a invocar como un método interpretado, el sistema de tiempo de ejecución 619 puede obtener el método a partir del intérprete 617. Por otro lado, si se determina que el método se va a invocar como un método compilado, el sistema de tiempo de ejecución 619 activa el compilador 615. A continuación, el compilador 615 genera instrucciones máquina nativas a partir de códigos de byte 605, y ejecuta las instrucciones de lenguaje máquina. En general, las instrucciones de lenguaje máquina se descartan cuando finaliza la máquina virtual 611. El funcionamiento de las máquinas virtuales o, más particularmente, las máquinas virtuales Java™, se describe más detalladamente en *The Java™ Virtual Machine Specification* de Tim Lindholm y Frank Yellin (ISBN 0-201-63452-X), que se incorpora en su totalidad a la presente descripción a título de referencia.

La presente invención utiliza varias operaciones implementadas por ordenador que involucran datos almacenados en sistemas de ordenador. Estas operaciones incluyen, aunque sin limitarse a las mismas, aquellas que requieren manipulación física de cantidades físicas. Habitualmente, aunque no de forma necesaria, estas cantidades adoptan la forma de señales eléctricas o magnéticas capaces de ser almacenadas, transferidas, combinadas, comparadas, y manipuladas de otra manera. Las operaciones descritas en el presente documento que forman parte de la invención son operaciones máquina útiles. A las manipulaciones realizadas se les hace referencia frecuentemente en términos tales como producir, identificar, hacer funcionar, determinar, comparar, ejecutar, descargar, o detectar. En ocasiones resulta conveniente, principalmente por motivos de uso común, referirse a estas señales eléctricas o magnéticas como bits, valores, elementos, variables, caracteres, datos, o similares. No obstante, debería recordarse que todos estos términos y otros similares se deben asociar a las cantidades físicas apropiadas y son simplemente etiquetas convenientes aplicadas a estas cantidades.

La presente invención se refiere asimismo a un dispositivo, sistema o aparato para realizar las operaciones mencionadas anteriormente. El sistema se puede construir especialmente para los fines requeridos, o puede ser un ordenador de propósito general selectivamente activado o configurado por un programa de ordenador almacenado en este último. Los procesos presentados anteriormente no están relacionados de forma inherente con ningún ordenador particular u otro aparato informático. En particular, se pueden usar varios ordenadores de propósito general con programas escritos de acuerdo con los aspectos dados a conocer en el presente documento, o, alternativamente, puede que resulte más conveniente construir un sistema de ordenador más especializado para realizar las operaciones requeridas.

La figura 7 es un diagrama de bloques de un sistema de ordenador de propósito general 700 adecuado para llevar a cabo el procesado según una forma de realización de la presente invención. La Figura 7 ilustra una forma de realización de un sistema de ordenador de propósito general. Se pueden usar otras arquitecturas y configuraciones de sistemas de ordenador para llevar a cabo el procesado de la presente invención. El sistema de ordenador 700, constituido por varios subsistemas que se describen posteriormente, incluye por lo menos un subsistema de microprocesador (al que se hace referencia también como unidad de procesado central, o CPU) 702. Es decir, la CPU 702 se puede implementar mediante un procesador de un solo chip o con múltiples procesadores. La CPU 702 es un procesador digital de propósito general que controla el funcionamiento del sistema de ordenador 700. Usando instrucciones recuperadas de memoria, la CPU 702 controla la recepción y manipulación de datos de entrada, y la salida y visualización de datos en dispositivos de salida.

La CPU 702 está acoplada bidireccionalmente a unos primeros medios de almacenamiento principales 704, típicamente una memoria de acceso aleatorio (RAM), y unidireccionalmente a una segunda área de almacenamiento principal 706, típicamente una memoria de solo lectura (ROM), a través de un bus de memoria 708. Tal como es bien sabido en la técnica, los medios de almacenamiento principales 704 se pueden usar como un área de almacenamiento general y como memoria temporal de tipo *scratch-pad*, y también se puede usar para almacenar datos de entrada y datos procesados. También puede almacenar instrucciones y datos de programación, en forma de una pila de memoria además de otros datos e instrucciones para procesos que estén funcionando sobre la CPU 702, y se usa típicamente para la transferencia rápida de datos de instrucciones, de una manera bidireccional, a través del bus de memoria 708. También tal como es bien sabido en la técnica, los medios de almacenamiento principales 706 incluyen típicamente instrucciones de funcionamiento básicas, código de programa, datos y objetos usados por la CPU 702 para realizar estas funciones. Los dispositivos de almacenamiento principales 704 y 706 pueden incluir cualesquiera soportes de almacenamiento legibles por ordenador adecuados, que se describen posteriormente, dependiendo de si, por ejemplo, el acceso a los datos necesita ser bidireccional o unidireccional. La CPU 702 también puede recuperar y almacenar de forma directa y muy rápidamente datos necesitados frecuentemente en una memoria caché 710.

Un dispositivo extraíble de almacenamiento de gran capacidad 712 proporciona una capacidad adicional de almacenamiento de datos para el sistema de ordenador 700, y el mismo está acoplado o bien bidireccional o bien unidireccionalmente a la CPU 702 a través de un bus periférico 714. Por ejemplo, un dispositivo extraíble de almacenamiento de gran capacidad, específico, conocido comúnmente como CD-ROM, pasa típicamente datos de forma unidireccional a la CPU 702, mientras que un disco flexible puede pasar datos bidireccionalmente a la CPU 702. Los medios de almacenamiento 712 también pueden incluir soportes legibles por ordenador tales como cinta magnética, memoria flash, señales materializadas en una onda portadora, TARJETAS DE PC, dispositivos portátiles de almacenamiento de gran capacidad, dispositivos de almacenamiento holográfico, y otros dispositivos de almacenamiento. Unos medios fijos de almacenamiento de gran capacidad 716 proporcionan también una capacidad adicional de almacenamiento de datos y están acoplados bidireccionalmente a la CPU 702 a través del bus periférico 714. El ejemplo más común de medios de almacenamiento de gran capacidad 716 es una unidad controladora de disco duro. En general, el acceso a estos soportes es más lento que el acceso a medios de almacenamiento principales 704 y 706. Los medios de almacenamiento de gran capacidad 712 y 716 en general almacenan instrucciones de programación adicionales, datos, y similares que, típicamente, no están siendo usados activamente por la CPU 702. Se apreciará que la información conservada dentro de los medios de almacenamiento de gran capacidad 712 y 716 se puede incorporar, si fuera necesario, de una manera normalizada como parte de los medios de almacenamiento principales 704 (por ejemplo, RAM) en forma de memoria virtual.

Además de proporcionar a la CPU 702 acceso a subsistemas de almacenamiento, el bus periférico 714 se usa para proporcionar acceso también a otros subsistemas y dispositivos. En la forma de realización descrita, los mismos incluyen un monitor de visualización 718 y un adaptador 720, un dispositivo de impresora 722, una interfaz de red 724, una interfaz de dispositivos auxiliares de entrada/salida 726, una tarjeta de sonido 728 y altavoces 730, y otros subsistemas según sea necesario.

La interfaz de red 724 permite que la CPU 702 se acople a otro ordenador, red de ordenadores, o red de telecomunicaciones usando una conexión de red según se muestra. A través de la interfaz de red 724, se contempla que la CPU 702 pudiera recibir información, por ejemplo, objetos de datos o instrucciones de programas, desde otra red, o que pudiera dar salida a información hacia otra red en el transcurso de la ejecución de las etapas de método descritas anteriormente. Se puede recibir información, frecuentemente representada como una secuencia de instrucciones a ejecutar en una CPU, desde otra red, y se puede dar salida a la misma hacia otra red, por ejemplo, en forma de una señal de datos de ordenador materializada en una onda portadora. Se puede usar una tarjeta de interfaz o un dispositivo similar y software apropiado implementado mediante la CPU 702, para conectar el sistema de ordenador 700 a una red externa y transferir datos según protocolos normalizados. Es decir, formas de realización de métodos de la presente invención se pueden ejecutar únicamente sobre la CPU 702, o se pueden realizar a través de una red tal como Internet, redes de intranet, o redes de área local, conjuntamente con una CPU remota que comparta una parte del procesado. A la CPU 702 también se pueden conectar dispositivos adicionales de almacenamiento de gran capacidad (no mostrados) a través de la interfaz de red 724.

La interfaz de dispositivos de I/O auxiliares 726 representa interfaces generales y personalizadas que permiten que la CPU 702 envíe y, más típicamente, reciba datos de otros dispositivos tales como micrófonos, pantallas sensibles al tacto, lectores de tarjetas transductoras, lectores de cintas, reconocedores de voz o de escritura, lectores biométricos, cámaras, dispositivos portátiles de almacenamiento de gran capacidad, y otros ordenadores.

Acoplado también a la CPU 702 se encuentra un controlador de teclado 732 a través de un bus local 734 para recibir entradas desde un teclado 736 ó un dispositivo de puntero 738, y enviar símbolos decodificados desde el teclado 736 ó dispositivo de puntero 738 a la CPU 702. El dispositivo de puntero puede ser un ratón, un lápiz óptico, un control de *track ball*, o una tablilla, y es útil para interaccionar con una interfaz gráfica de usuario.

Adicionalmente, formas de realización de la presente invención se refieren además a productos de almacenamiento de ordenador con un soporte legible por ordenador que contienen código de programa para realizar varias operaciones implementadas por ordenador. El soporte legible por ordenador es cualquier dispositivo de almacenamiento de datos que pueda almacenar datos que seguidamente puedan ser leídos por un sistema de ordenador. Los soportes y el código de programa pueden ser aquellos especialmente diseñados y construidos para los fines de la presente invención, o pueden ser del tipo bien conocido para aquellos con conocimientos habituales en las técnicas de software informático. Los ejemplos de soportes legibles por ordenador incluyen, aunque sin limitarse a los mismos, todos los soportes mencionados anteriormente: soportes magnéticos tales como discos duros, discos flexibles, y cinta magnética; soportes ópticos tales como discos de CD-ROM; soportes magneto-ópticos tales como discos *floptical*; y dispositivos de hardware especialmente configurados tales como circuitos integrados de aplicación específica (ASICs), dispositivos lógicos programables (PLDs), y dispositivos de ROM y RAM. El soporte legible por ordenador también se puede distribuir como una señal de datos materializada en una onda portadora a través de una red de sistemas de ordenador acoplados, de manera que el código legible por ordenador se almacena y ejecuta de una forma distribuida. Los ejemplos de código de programa incluyen tanto código máquina, en la medida en la que sea producido, por ejemplo, por un compilador, como archivos que contengan código de nivel superior que se pueda ejecutar usando un intérprete.

Los expertos en la materia apreciarán que los elementos de hardware y software descritos anteriormente tienen un diseño y una construcción normalizados. Otros sistemas de ordenador adecuados para su uso con la invención pueden incluir subsistemas adicionales o un número menor de subsistemas. Adicionalmente, el bus de memoria 708, el bus periférico 714, y el bus local 734 son ilustrativos de cualquier esquema de interconexión que sirva para enlazar los subsistemas. Por ejemplo, se podría usar un bus local para conectar la CPU a unos medios fijos de almacenamiento de gran capacidad 716 y un adaptador de visualización 720. El sistema de ordenador mostrado en la figura 7 no es más que un ejemplo de un sistema de ordenador adecuado para ser usado con la invención. También se pueden utilizar otras arquitecturas de ordenador que presenten diferentes configuraciones de subsistemas.

Aunque la invención anterior se ha descrito con cierto detalle con el fin de clarificar su compresión, resultará evidente que se pueden llevar a la práctica ciertos cambios y modificaciones dentro del alcance de las reivindicaciones adjuntas. Además, debería observarse que existen formas alternativas de implementar tanto el proceso como el aparato de la presente invención. Por consiguiente, las presentes formas de realización deben considerarse como ilustrativas y no limitativas, y la invención no debe limitarse a los detalles ofrecidos en la presente, sino que se puede modificar dentro del alcance y los equivalentes de las reivindicaciones adjuntas.

REIVINDICACIONES

1. Método de ejecución de un método nativo en una máquina virtual Java que incluye una pila de Java, comprendiendo el método:

determinar (202) si un método nativo (106, 108) va a ser gestionado por una primera interfaz nativa o una de una pluralidad de otras interfaces nativas;

si el método nativo va a ser gestionado por la primera interfaz nativa, invocar al método nativo y posibilitar que el método nativo acceda a un estado interno de la máquina virtual Java mediante la obtención (204) de un puntero de función desde un bloque de método, la invocación a la función del método nativo y el traslado (206), a la función del método nativo, de datos que permiten el acceso a un estado de la máquina virtual Java que va a ser usado por el método nativo sin realizar retrollamadas a la máquina virtual Java, incluyendo dichos datos un puntero a argumentos en la pila de Java;

ejecutar (208) el método nativo en la máquina virtual Java; y

almacenar, por medio del método nativo, sus resultados en la pila de Java, y modificar (302) un puntero de pila de Java sobre la base de un código de retorno, ajustando (302, 304) de este modo el estado de la máquina virtual Java sobre la base de la ejecución del método nativo, con lo cual se minimiza la transición entre un bucle intérprete y el método nativo a través de la primera interfaz nativa.

2. Método según la reivindicación 1, que comprende además clasificar uno o más métodos nativos de manera que el método o métodos nativos quedan capacitados para ser gestionados por la primera interfaz nativa.

3. Método según la reivindicación 1 ó 2, en el que la determinación de si un método nativo va a ser gestionado por una primera interfaz nativa o una de una pluralidad de otras interfaces nativas comprende además examinar (202) un bloque de método del método nativo para determinar un tipo de método.

4. Método según cualquiera de las reivindicaciones 1 a 3, en el que la ejecución del método en un bucle intérprete comprende además:

apilar (402) un marco de transición correspondiente a un método particular en una primera pila en la máquina virtual Java;

apilando (404) el método nativo una pluralidad de argumentos asociados al marco de transición en la primera pila; y

retornar (406) un código de resultado al bucle intérprete.

5. Método según la reivindicación 4, en el que la primera pila es una pila de Java y el código de resultado indica que se ha apilado un marco de transición nuevo en la pila de Java.

6. Sistema (700) para ejecutar un método nativo en una máquina virtual Java que incluye una pila de Java, comprendiendo el sistema:

un procesador (702); y

un soporte legible por ordenador, que almacena un programa para su ejecución por parte del procesador, comprendiendo el programa:

un código de ordenador que determina si un método nativo va a ser gestionado por una primera interfaz nativa o una de una pluralidad de otras interfaces nativas;

un código de ordenador que invoca al método nativo y posibilita que el método nativo acceda a un estado de la máquina virtual Java obteniendo (204) un puntero de función a partir de un bloque de método, invocando a la función de método nativo y pasando (206), a la función de método nativo, datos que permiten el acceso a un estado de la máquina virtual Java a usar por el método nativo sin realizar retrollamadas a la máquina virtual Java, incluyendo dichos datos un puntero a argumentos en la pila de Java, si el método nativo va a ser gestionado por la primera interfaz nativa; y

un código de ordenador que ejecuta el método en la Máquina Virtual Java; y

un código de ordenador que provoca que el método nativo almacene sus resultados en la pila de Java, y que modifique un puntero de pila de Java sobre la base de un código de retorno, ajustando de este modo el estado de la Máquina Virtual Java sobre la base de la ejecución del método con lo cual se minimiza la transición entre un bucle intérprete y el método nativo a través de la primera interfaz nativa.

7. Soporte legible por ordenador, que contiene instrucciones programadas dispuestas para ejecutar un método nativo en una máquina virtual Java que incluye una pila de Java, incluyendo, el soporte legible por ordenador, instrucciones programadas, para:

determinar si un método nativo va a ser gestionado por una primera interfaz nativa o una de una pluralidad de otras interfaces nativas;

si el método va a ser gestionado por la primera interfaz nativa, invocar al método nativo y posibilitar que el método nativo acceda a un estado de la máquina virtual Java mediante la obtención de un puntero de función desde un bloque de método, la invocación a la función del método nativo y el traslado, a la función del método nativo, de datos que permiten el acceso a un estado de la máquina virtual Java que va a ser usado por el método nativo sin realizar retrollamadas a la máquina virtual Java, incluyendo dichos datos un puntero a argumentos en la pila de Java;

ejecutar el método en la Máquina Virtual Java; y

almacenar, por medio del método nativo, sus resultados en la pila de Java, y modificar un puntero de pila de Java sobre la base de un código de retorno, ajustando de este modo el estado de la JVM sobre la base de la ejecución del método, con lo cual se minimiza la transición entre un bucle intérprete y el método nativo a través de la primera interfaz nativa.

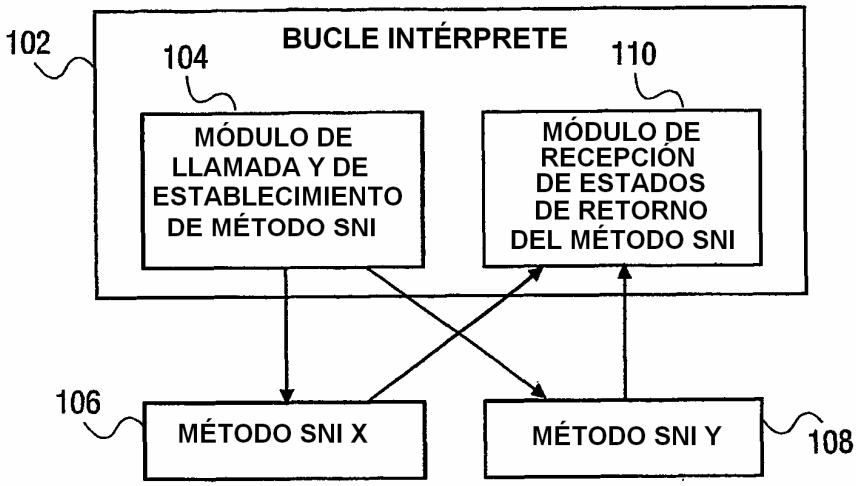
102
BUCLE INTÉRPRETE
104
110
MÓDULO DE LLAMADA Y DE ESTABLECIMIENTO DE MÉTODO SNI
MÓDULO DE RECEPCIÓN DE ESTADOS DE RETORNO DEL MÉTODO SNI
106
MÉTODO SNI X
MÉTODO SNI Y
108

FIG. 1

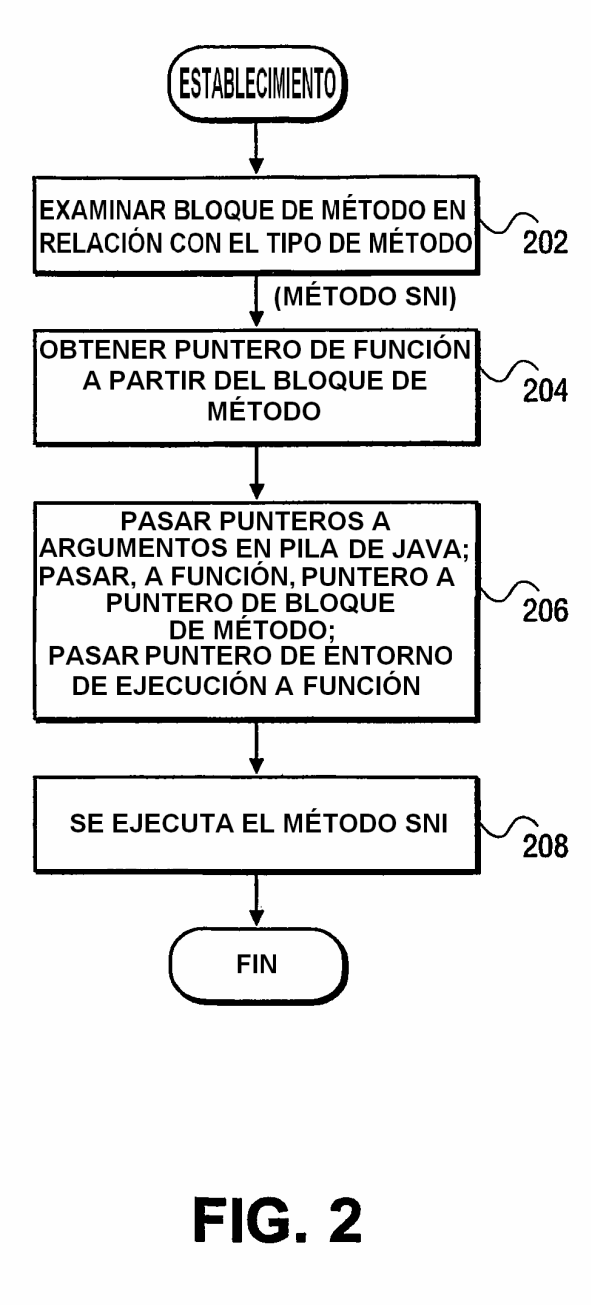
ESTABLECIMIENTO
EXAMINAR BLOQUE DE MÉTODO EN RELACIÓN CON EL TIPO DE MÉTODO
202
(MÉTODO SNI)
OBTENER PUNTERO DE FUNCIÓN A PARTIR DEL BLOQUE DE MÉTODO
204
PASAR PUNTEROS A ARGUMENTOS EN PILA DE JAVA; PASAR, A FUNCIÓN, PUNTERO A PUNTERO DE BLOQUE DE MÉTODO; PASAR PUNTERO DE ENTORNO DE EJECUCIÓN A FUNCIÓN
206
SE EJECUTA EL MÉTODO SNI
208
FIN

FIG. 2

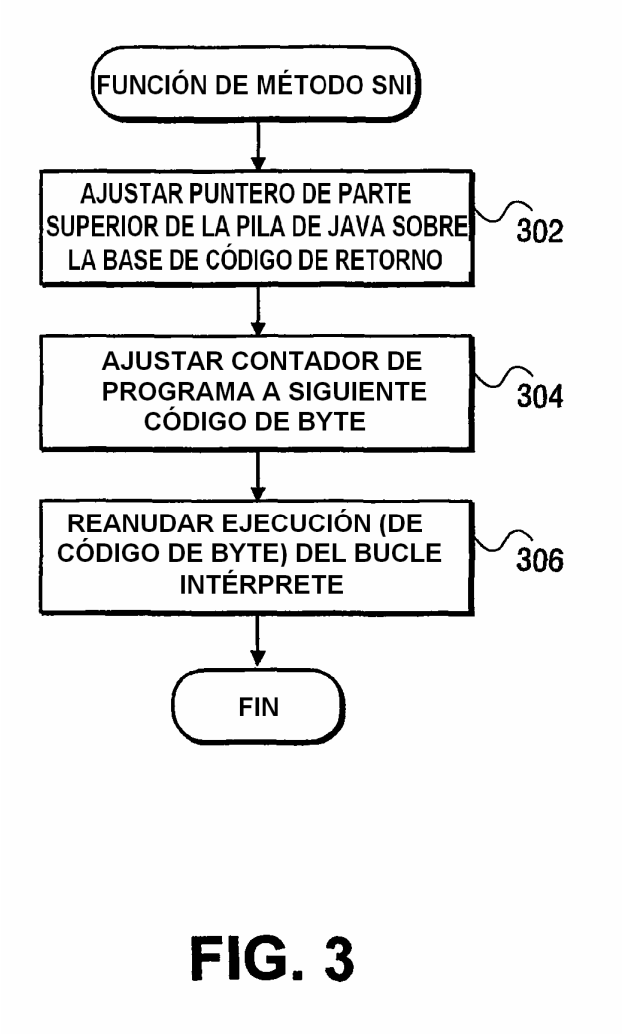
FUNCIÓN DE MÉTODO SNI
AJUSTAR PUNTERO DE PARTE SUPERIOR DE LA PILA DE JAVA SOBRE LA BASE DE CÓDIGO DE RETORNO
302
AJUSTAR CONTADOR DE PROGRAMA A SIGUIENTE CÓDIGO DE BYTE
304
REANUDAR EJECUCIÓN (DE CÓDIGO DE BYTE) DEL BUCLE INTÉRPRETE
306
FIN

FIG. 3

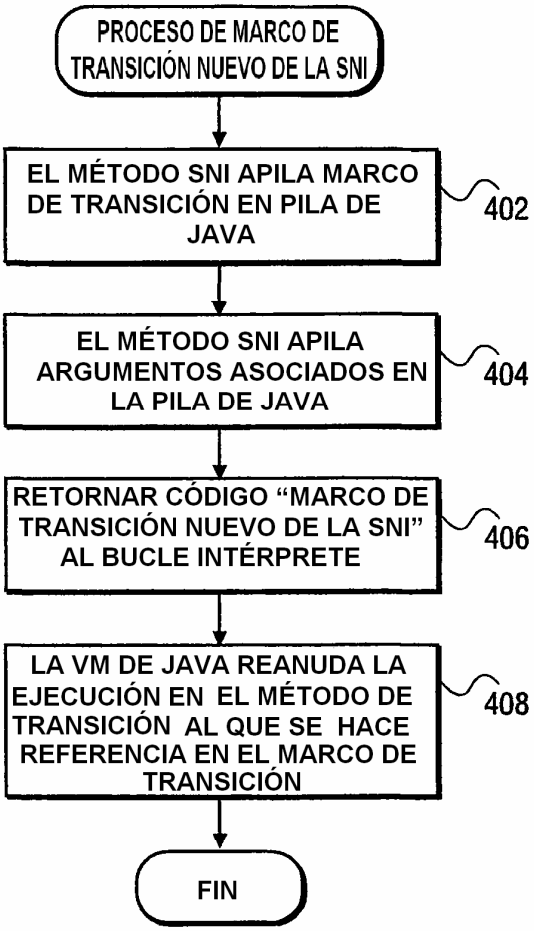
PROCESO DE MARCO DE TRANSICIÓN NUEVO DE LA SNI
EL MÉTODO SNI APILA MARCO DE TRANSICIÓN EN PILA DE JAVA
402
EL MÉTODO SNI APILA ARGUMENTOS ASOCIADOS EN LA PILA DE JAVA
404
RETORNAR CÓDIGO "MARCO DE TRANSICIÓN NUEVO DE LA SNI" AL BUCLE INTÉRPRETE
406
LA VM DE JAVA REANUDA LA EJECUCIÓN EN EL MÉTODO DE TRANSICIÓN AL QUE SE HACE REFERENCIA EN EL MARCO DE TRANSICIÓN
408
FIN

FIG. 4

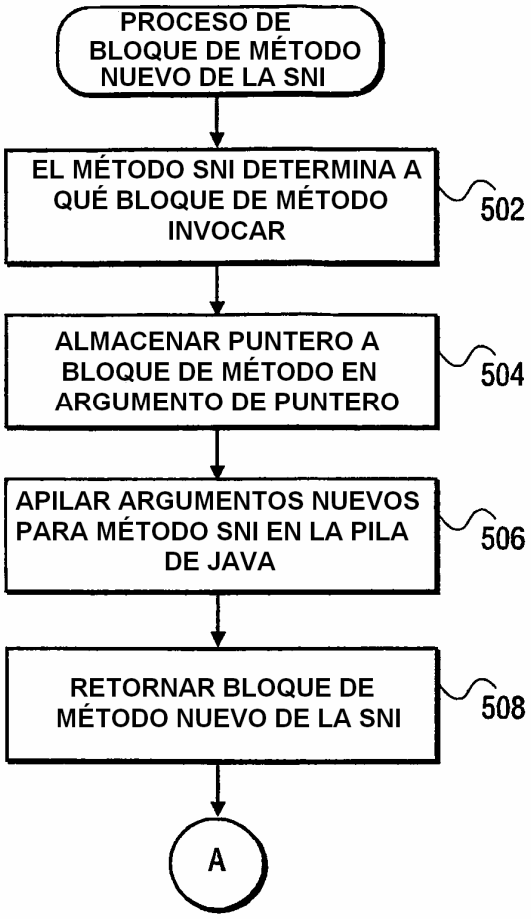
PROCESO DE BLOQUE DE MÉTODO NUEVO DE LA SNI
EL MÉTODO SNI DETERMINA A QUÉ BLOQUE DE MÉTODO INVOCAR
502
ALMACENAR PUNTERO A BLOQUE DE MÉTODO EN ARGUMENTO DE PUNTERO
504
APILAR ARGUMENTOS NUEVOS PARA MÉTODO SNI EN LA PILA DE JAVA
506
RETORNAR BLOQUE DE MÉTODO NUEVO DE LA SNI
508
A

FIG. 5A

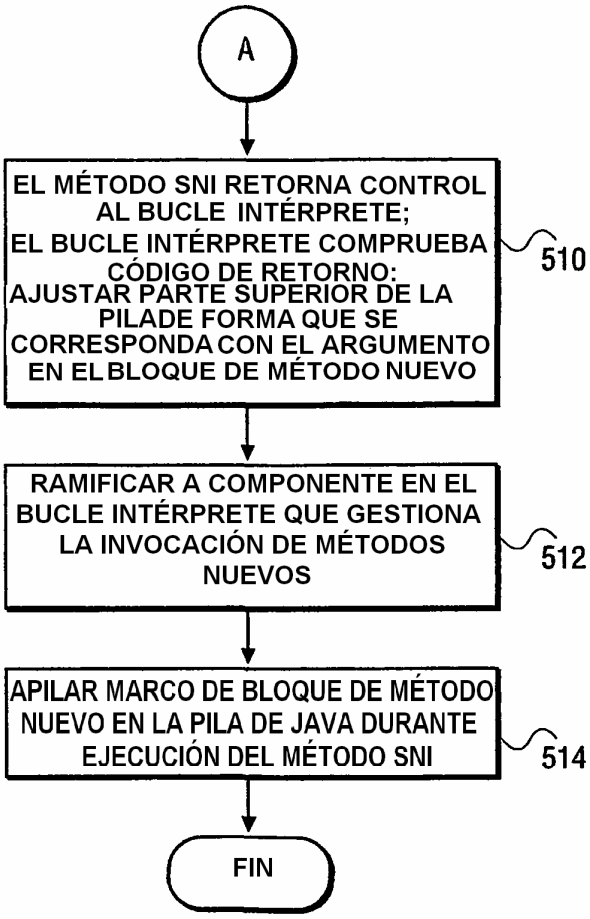
A
EL MÉTODO SNI RETORNA CONTROL AL BUCLE INTÉRPRETE;
EL BUCLE INTÉRPRETE COMPRUEBA CÓDIGO DE RETORNO:
AJUSTAR PARTE SUPERIOR DE LA PILADE FORMA QUE SE CORRESPONDA CON EL ARGUMENTO EN EL BLOQUE DE MÉTODO NUEVO
510
RAMIFICAR A COMPONENTE EN EL BUCLE INTÉRPRETE QUE GESTIONA LA INVOCACIÓN DE MÉTODOS NUEVOS
512
APILAR MARCO DE BLOQUE DE MÉTODO NUEVO EN LA PILA DE JAVA DURANTE EJECUCIÓN DEL MÉTODO SNI
514
FIN

FIG. 5B

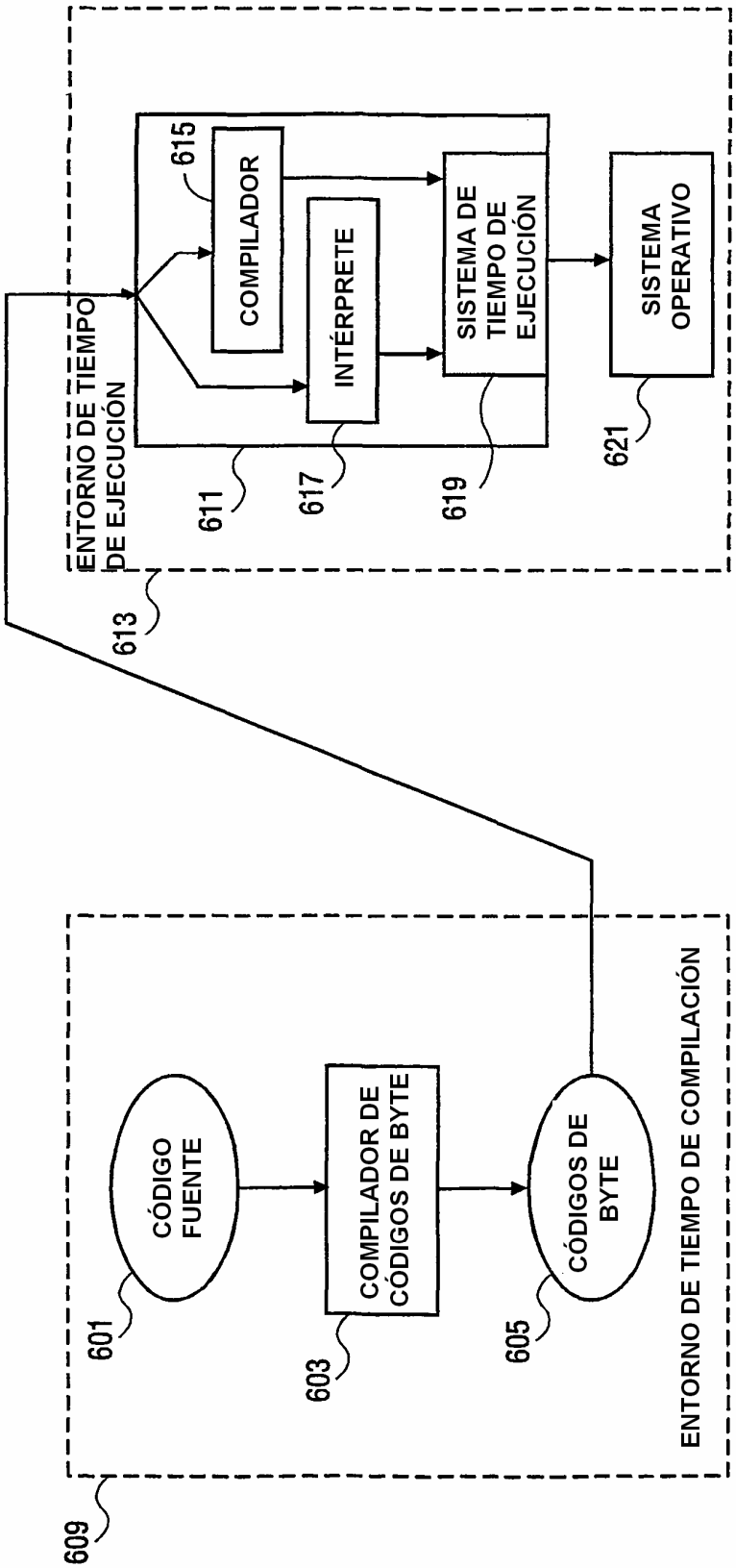
ENTORNO DE TIEMPO DE COMPILACIÓN
609
CÓDIGO FUENTE
601
COMPILADOR DE CÓDIGOS DE BYTE
603
CÓDIGOS DE BYTE
605
613
ENTORNO DE TIEMPO DE EJECUCIÓN
615
COMPILADOR
INTÉRPRETE
617
SISTEMA DE TIEMPO DE EJECUCIÓN
619
SISTEMA OPERATIVO
621
611

FIG. 6

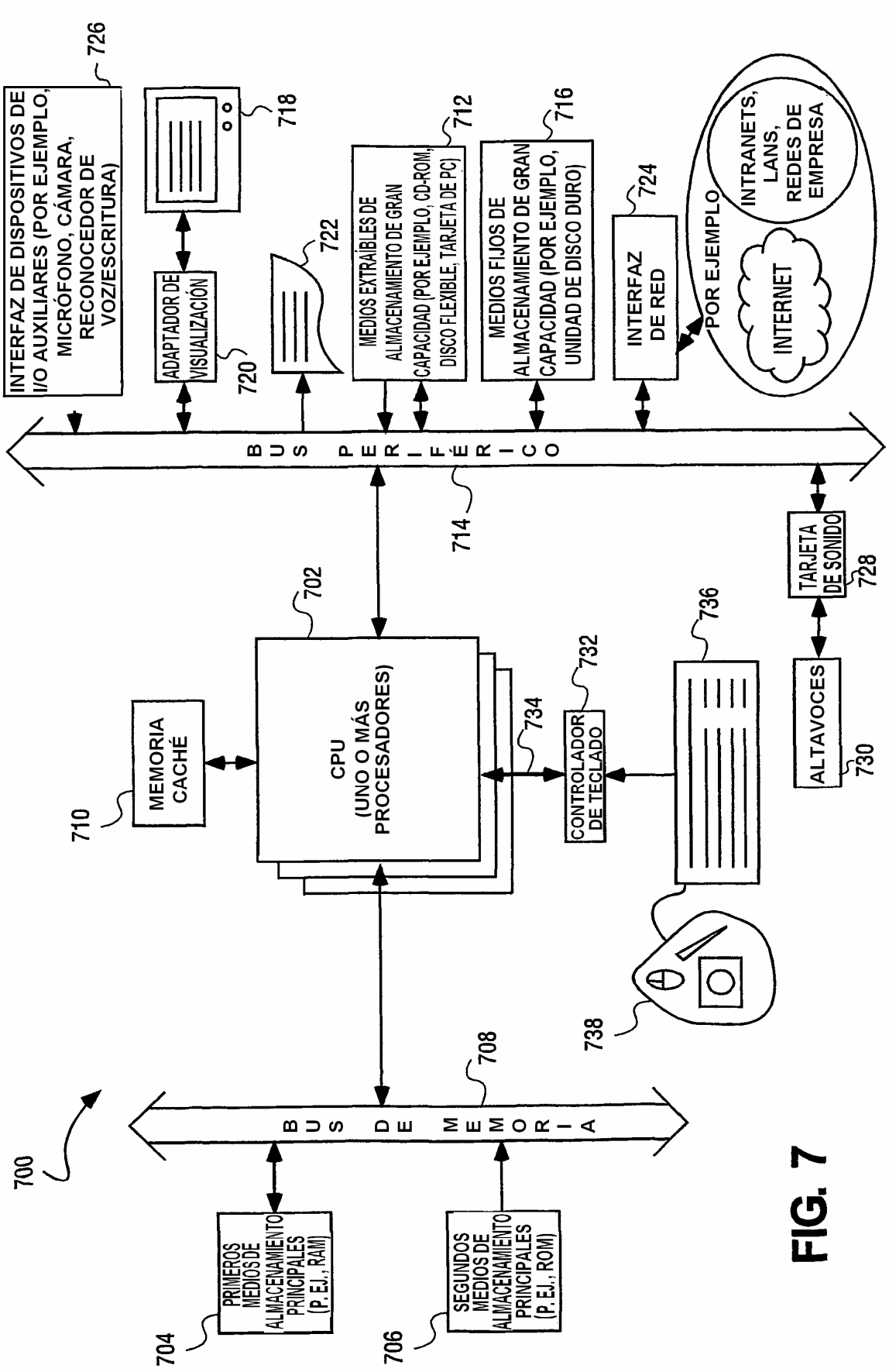
700
BUS DE MEMORIA
708
PRIMEROS MEDIOS DE ALMACENAMIENTO PRINCIPALES (P. EJ., RAM)
704
SEGUNDOS MEDIOS DE ALMACENAMIENTO PRINCIPALES (P. EJ., ROM)
706
MEMORIA CACHÉ
710
CPU (UNO O MÁS PROCESADORES)
702
734
CONTROLADOR DE TECLADO
732
736
738
BUS PERIFÉRICO
714
TARJETA DE SONIDO
728
ALTAVOCES
730
INTERFAZ DE RED
724
POR EJEMPLO
INTERNET
INTRANETS, LANS, REDES DE EMPRESA
MEDIOS FIJOS DE ALMACENAMIENTO DE GRAN CAPACIDAD (POR EJEMPLO, UNIDAD DE DISCO DURO)
716
MEDIOS EXTRAÍBLES DE ALMACENAMIENTO DE GRAN CAPACIDAD (POR EJEMPLO, CD-ROM, DISCO FLEXIBLE, TARJETA DE PC)
712
722
ADAPTADOR DE VISUALIZACIÓN
720
718
INTERFAZ DE DISPOSITIVOS DE I/O AUXILIARES (POR EJEMPLO, MICRÓFONO, CÁMARA, RECONOCEDOR DE VOZ/ESCRITURA)
726

FIG. 7