(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 645 010**

(51) Int. CI.:

G06F 9/45 (2006.01)
G06F 9/455 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **28.10.2011 PCT/EP2011/069044**

(87) Fecha y número de publicación internacional: **03.05.2012 WO12056023**

(96) Fecha de presentación y número de la solicitud europea: **28.10.2011 E 11807873 (2)**

(97) Fecha y número de publicación de la concesión europea: **26.07.2017 EP 2633399**

(54) Título: **Método y sistema para generar un código**

(30) Prioridad:

28.10.2010 WO PCT/EP2010/066414

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
01.12.2017

(73) Titular/es:

DELOITTE INNOWAKE GMBH (100.0%)
Robert-Bosch-Straße 1 IT-Tower
89250 Senden, DE

(72) Inventor/es:

BERNECKER, THORSTEN

(74) Agente/Representante:

TEMIÑO CENICEROS, Ignacio

DESCRIPCIÓN

Método y sistema para generar un código

Área de la invención

La invención se refiere a un método implementado por ordenador para generar un código para su ejecución en un entorno de tiempo de ejecución. Además, la invención se refiere a una unidad de manipulación de código intermedio para generar un código dentro de un entorno de tiempo de ejecución, así como a un sistema para generar un código para su ejecución en un entorno de tiempo de ejecución.

Antecedentes y estado de la técnica

En el área de los sistemas y aplicaciones de software operativo, se ha deseado desde hace mucho tiempo portar y ejecutar aplicaciones creadas originalmente para sistemas anfitrión u ordenadores centrales en sistemas informáticos más modernos y potentes. Esto a menudo implica ofrecer la misma funcionalidad previamente disponible en la aplicación de software en el sistema de destino. Sin embargo, a menudo es imposible simplemente recompilar el código fuente de la aplicación en el nuevo sistema porque los compiladores no están disponibles en la plataforma de destino.

Además de eso, un objetivo concurrente al mover aplicaciones a una nueva plataforma es hacer que el código fuente esté disponible en un lenguaje de programación de módem, por ejemplo, un lenguaje de programación orientado a objetos. Tal objetivo puede manifestarse en un requisito para transformar el código fuente de una aplicación de COBOL, Natural o PL/1 en código fuente de Java.

La transformación del código fuente original en un lenguaje de programación de módem permite la compilación en muchas plataformas diferentes ya que dichos compiladores están disponibles en una gran variedad de plataformas de hardware y sistemas operativos. El código fuente transformado puede compilarse fácilmente en el sistema de destino deseado.

Además, los programas transformados son más fáciles y rentables de mantener y modernizar.

Existen formas conocidas de transformar el código fuente de un lenguaje de programación en otro de forma automatizada. Las construcciones de lenguaje de un lenguaje original se asignan a construcciones de lenguaje en un lenguaje de destino. El objetivo de esto es crear la menor diferencia posible entre la estructura del programa de la aplicación original y la estructura del programa de destino, es decir, la creación de una transformación de declaración a declaración del código fuente original. De esta manera, el código fuente original y el código fuente de destino se ven muy similares, de hecho, familiares para cualquiera que conozca el código de la aplicación original. El alto parecido garantiza un mantenimiento óptimo.

Sin embargo, los diferentes paradigmas de diseño en el lenguaje original y de destino a menudo hacen imposible tal transformación de declaración a declaración. El código fuente original y el destino exhiben enormes diferencias. Esto es exacerbado por el hecho de que el lenguaje de programación original contiene construcciones de lenguaje que no están disponibles en el lenguaje de programación de destino.

Tales construcciones de lenguaje son, por ejemplo, instrucciones de salto. Los lenguajes de procedimiento, como Natural y COBOL, a menudo son usados para controlar el flujo del programa. La ejecución del programa se detiene en un punto y continúa en otro. El lenguaje de programación Natural tiene el comando “REINPUT” que obliga a la ejecución del programa a continuar en el punto del programa que recibió la última entrada del usuario. “REINPUT” puede usarse en cualquier lugar del código Natural.

El uso intensivo de las instrucciones de salto en los lenguajes de programación de procedimientos conduce a grandes partes del código fuente original que requieren una transformación que no emplea una transformación de declaración a declaración. En cambio, estas partes del código deben reestructurarse de forma masiva o incluso requieren una reimplementación manual. Incluso si el código transformado es semánticamente correcto, el código transformado puede no ser sintácticamente equivalente al código original. Una transformación de este tipo tiene la desventaja de no ser reconocible en comparación con la estructura original del programa.

David Doolin, Jack Dongarra, Keith Seymour, "JLAPACK - Compiling LAPACK Fortran to Java", (20000526), Citeseer, URL: http://citeseerx ist.psu.edu/viewdoc/summary?doi=10.1.1.33.6090 , XP002630913 se refiere a subrutinas numéricas traducidas de su subconjunto fuente Fortran 77 en archivos de clase, ejecutables por la Máquina Virtual de Java (JVM) y adecuadas para su uso por los programadores de Java. Esto hace posible que las aplicaciones o applets Java, distribuidos en World Wide Web (WWW), utilicen un código numérico heredado establecido originalmente en Fortran. La traducción se realiza utilizando un compilador Fortran-a-Java (fuente-a-fuente) de propósito especial.

El documento US 2006031820 A1 se refiere a un método para la transformación de programas y un aparato para la transformación de programas COBOL a Java. El método consiste en: (1) un enfoque para la transformación del programa de declaración a declaración, facilitado por una biblioteca de lenguaje objetivo predefinido, que mantiene los comentarios originales, el flujo de control del programa, la funcionalidad y la complejidad del tiempo; (2) un enfoque para ir a la eliminación de declaraciones, que utiliza el mecanismo existente de manejo de excepciones en el lenguaje objetivo y su implementación se oculta en una superclase en una biblioteca; (3) un BNF extendido para distinguir diferentes apariciones del mismo término en una producción de BNF; (4) un nuevo enfoque para la declaración incorporada como una declaración de marcador especial y un comentario, (5) en la descripción de lo anterior, se define un lenguaje de especificación de transformación de programa para describir la relación entre los comentarios en dos lenguajes. (6) un aparato, como la realización preferida del método, es un sistema de transformación de programa COBOL a Java Cobol2Java; Se proporciona una muestra de la aplicación COBOL y su traducción a Cobol2Java.

Objeto de la invención

El objeto de la invención es proporcionar un método y un sistema que permitan transformar el código fuente original en un lenguaje de programación original en un código fuente de destino semánticamente idéntico y sintácticamente casi idéntico en un lenguaje de programación de destino, mediante el cual la transformación de construcciones de lenguaje en la programación original el lenguaje que no está disponible en el lenguaje de programación de destino aún es posible, manteniendo un parecido más cercano con la estructura original. Un objeto particular de la invención es transformar cualquier código fuente original en un lenguaje de programación original en código fuente en un lenguaje de programación de destino donde el código fuente original contiene instrucciones de salto que no están disponibles en el lenguaje de programación de destino

Solución de acuerdo con la invención

Este objeto se resuelve mediante los métodos, las unidades de manipulación de código, los sistemas y los productos de programas informáticos de acuerdo con las reivindicaciones independientes respectivas.

La solución de acuerdo con la invención permite transformar programas en código fuente original en código fuente de destino que tiene una estructura similar incluso si el lenguaje de programación de destino no permite instrucciones de salto.

De este modo, la invención permite retener la estructura del programa original cuando se ha creado en un lenguaje de programación original y se ha transformado en un nuevo programa en un lenguaje de destino de módem creando las instrucciones de salto que no están disponibles en la fuente del lenguaje de destino del módem en su código ejecutable (objeto). Esto permite retener estructuras de programación bien conocidas y entendidas de los programas existentes en el lenguaje de programación de destino. El código resultante en el lenguaje de destino sigue siendo tan reconocible y sostenible como el código original. Esto también aumenta la probabilidad de que la aplicación en el lenguaje de programación de destino se ejecute correctamente.

Además, esto permite una transformación de declaración a declaración del código fuente original en el código fuente de destino.

Las realizaciones preferidas comprenden las siguientes características.

Al menos un fragmento de código fuente insertado en el código fuente de destino se adapta para realizar declaraciones de programa que son funcionalmente equivalentes a una instrucción de salto estática colocada en el código fuente original, si el destino de la instrucción de salto está dentro del marco de pila PARR18 de la instrucción de salto.

Al menos un fragmento de código fuente insertado en el código fuente de destino se adapta para realizar declaraciones de programa que son funcionalmente equivalentes a una instrucción de salto dinámica colocada en el código fuente original si el destino de la instrucción de salto está fuera del marco de pila de la instrucción de salto.

Al menos un fragmento de código fuente insertado en el código fuente de destino se adapta para realizar declaraciones de programa que son funcionalmente equivalentes a una instrucción de salto dinámico colocada en el código fuente original, si el destino de la instrucción de salto está dentro del marco de pila de la instrucción de salto.

El fragmento de código fuente insertado en el código fuente de destino comprende además una variable auxiliar que contiene el destino del salto y que se evalúa mediante la declaración de cambio.

El lenguaje de programación de destino comprende un lenguaje de programación que no permite declaraciones de salto del tipo “GOTO” en su código fuente.

El lenguaje de programación de destino comprende al menos uno de Java y un lenguaje de programación que es ejecutable en el entorno de ejecución “.NET”.

El lenguaje de programación fuente comprende un lenguaje de programación que permite instrucciones de salto en su código fuente.

El lenguaje de programación fuente es al menos uno de Cobol, Natural y PL/1.

El método comprende además un paso de crear código final a partir del código fuente de destino, por lo que el código final comprende código independiente de hardware, el código final para ejecución en un entorno de tiempo de ejecución.

El entorno de tiempo de ejecucióncomprende una Máquina Virtual de Java.

Al menos un fragmento de código fuente insertado en el código fuente de destino puede ser adaptado para realizar declaraciones de programa que son funcionalmente equivalentes a una instrucción de salto dinámica colocada en el código fuente original, si el destino de la instrucción de salto está fuera del marco de pila de la instrucción de salto.

La unidad de manipulación de código puede adaptarse adicionalmente para insertar en la fuente de destino al menos una variable auxiliar que contiene el destino del salto y que es evaluada por la declaración condicional.

El sistema de acuerdo puede comprender además medios para crear un código final a partir del código fuente de destino, por lo que el código final comprende un código independiente de hardware, estando el código final para su ejecución en un entorno de tiempo de ejecución.

En otras realizaciones preferidas de la invención, una o más de las siguientes características pueden estar comprendidas:

- Al menos una instrucción de salto insertada en el código final puede contener manejo de excepciones.

Al menos una instrucción de salto insertada en el código final puede comprender un salto directo dentro del código final, si el destino del salto se encuentra dentro del área válida del salto directo.

El área válida es el marco de pila de una subrutina de la pila de llamadas, por lo que la pila de llamadas está compuesta por marcos de pila. Cada marco de pila corresponde a una llamada a una subrutina que aún no ha terminado con un retorno. El marco de pila en la parte superior de la pila de llamadas es para la rutina que se está ejecutando actualmente.

La declaración de manejo de excepciones se puede adaptar para aceptar y manejar al menos una excepción de declaración de señalización.

La declaración de gestión de excepciones puede adaptarse para indicar, al aceptar al menos la excepción de la declaración de señalización, una excepción si la declaración de gestión de excepciones no está dentro del área válida del destino de salto del salto directo.

La declaración de señalización se puede insertar en el código final a través de la unidad de manipulación de código intermedio.

Durante la compilación del código fuente original en el código fuente de destino, se pueden crear datos de control que alimentan a la unidad de manipulación de código intermedio y que contiene información sobre cómo deben insertarse las declaraciones de señalización en el código final.

El código intermedio generado puede contener al menos una declaración de señalización.

El código fuente de destino puede contener al menos una declaración de señalización.

El salto directo puede ser del tipo “GOTO”.

El código intermedio puede comprender código de byte para ser interpretado por un intérprete de una máquina virtual.

El lenguaje de programación original puede comprender un lenguaje de programación que permite instrucciones de salto en su código fuente. El lenguaje de programación original puede ser al menos un Cobol, Natural y PL/1.

Es particularmente beneficioso que el código fuente original esté en el lenguaje de programación Natural y el código fuente de destino sea Java. Natural ofrece un sistema muy fácil para manejar instrucciones de salto lo que ha llevado a un uso generalizado del lenguaje en escenarios de negocios. Java, por otro lado, ha encontrado un uso generalizado debido a su independencia de plataforma. Mediante el uso de la invención, el programa Natural se puede transformar en programas Java y ejecutarse en plataformas arbitrarias y se pueden mantener y mejorar allí.

La máquina virtual puede comprender una Máquina Virtual de Java y la unidad de manipulación de código intermedio comprende un cargador de clase Java de la Máquina Virtual de Java.

La unidad de manipulación de código intermedio puede derivarse del cargador de clases Java de la Máquina Virtual de Java, es decir, el cargador de clase de la Máquina Virtual de Java puede usarse como clase base a partir de la cual se puede derivar la clase de la unidad de manipulación de código intermedio.

El código final generado por la unidad de manipulación del código intermedio se puede entregar para su ejecución en la máquina virtual.

La máquina virtual puede comprender una Máquina Virtual de Java.

El código final puede comprender código independiente del hardware.

La unidad de manipulación de código intermedio de acuerdo con esta invención es la herramienta que permite implementar el método descrito en esta invención.

Hacer que la unidad de manipulación de código intermedio sea independiente y desacoplada del cargador de clases de Máquina Virtual de Java (JVM) e implementarla de manera que produzca un código de objeto ejecutable tal que el cargador de clase de JVM pueda cargar y ejecutar mediante la interpretación de JVM permite crear Código de Byte Java para JVM que, por un lado, es una representación semántica completa del código fuente Natural y, por otro lado, no necesita ser adaptado por el cargador de clase de JVM para insertar instrucciones de salto en el Código Byte Java. Una ventaja adicional es que el código objeto es ejecutable en cualquier entorno JVM sin requerir un cargador de clase especialmente adaptado o una unidad de manipulación en tiempo de ejecución para el entorno de destino.

El código intermedio puede haberse creado a partir del código fuente de destino en un lenguaje de programación de destino, mediante el cual el código fuente de destino se compiló del código fuente original que contiene al menos una instrucción de salto, al omitir al menos una instrucción de salto.

La unidad de manipulación de código intermedio puede comprender un cargador de clase Java.

La unidad de manipulación de código intermedio puede haberse derivado de un cargador de clase Java.

El código final puede comprender código independiente del hardware.

Si la unidad de manipulación de código intermedio se deriva del cargador de clase de la Máquina Virtual de Java (JVM) y la comprende, entonces la unidad de manipulación de código intermedio es parte de la JVM.

La unidad de manipulación de código intermedio puede derivarse de un cargador de clase Java.

La unidad de manipulación de código intermedio se puede adaptar para verificar antes de insertar una instrucción de salto en el código final que el objetivo para la instrucción de salto se encuentra dentro del área válida de la instrucción de salto que se debe insertar.

La unidad de manipulación de código intermedio puede adaptarse adicionalmente para insertar una instrucción de salto en el código final que activa un salto directo al objetivo de salto en el código final durante la ejecución del código final si el área válida de la instrucción de salto a insertar está dentro del área válida del objetivo del salto.

La unidad de manipulación de código intermedio puede adaptarse adicionalmente para insertar una declaración de señalización en el código final que indica una excepción durante la ejecución del código final, si el área válida de la instrucción de salto que se va a insertar queda fuera del área válida del objetivo del salto.

La unidad de manipulación del código intermedio puede adaptarse adicionalmente para insertar una declaración de manejo de excepciones en el código final que captura y maneja la excepción señalada.

La declaración de manejo de excepción puede contener el salto directo al objetivo en el código final, si la declaración de manejo de excepción se encuentra dentro del área válida del objetivo del salto.

El código final puede comprender código independiente del hardware.

Un producto de programa de ordenador con código de aplicación que, cuando se carga en una unidad de procesamiento de datos, ejecuta el método descrito anteriormente.

Breve descripción de figuras.

Más detalles y propiedades de la invención se dan en la siguiente descripción de los dibujos en los que:

La figura 1a ilustra un diagrama de bloques para delinear un método de acuerdo con el segundo aspecto de la invención;

La figura 1b ilustra un diagrama de bloques para delinear un segundo ejemplo del método de acuerdo con el segundo aspecto de la invención;

La figura 2 ilustra la ejecución de una condición de excepción de múltiples etapas que representa una instrucción de salto en el código fuente original como se ejemplifica en el Código Byte Java;

Las figuras 3a-3b ilustran dos posibles formas de realización de un sistema de acuerdo con el segundo aspecto de la invención;

La figura 4 ilustra un diagrama de bloques para delinear el método de acuerdo con el primer aspecto de la invención;

Las figuras 5a-5b ilustran ejemplos de código fuente en un lenguaje de programación de destino; y

La figura 6 representa una realización de un sistema de acuerdo con la invención para ejecutar el método de acuerdo con el primer aspecto de la invención.

Descripción detallada

Lo siguiente describirá un ejemplo que existe como código fuente original en el lenguaje de programación natural que se ejecutará en un entorno de tiempo de ejecución de Máquina Virtual de Java (JVM) después de la transformación. Para permitir ejecutar el código fuente Natural original en la JVM, el código fuente Natural primero debe transferirse al código fuente en el lenguaje de programación Java. Un compilador de Java compila este código fuente de Java en un código de objeto ejecutable que pueda leer el cargador de clase de la JVM. El código objeto se conoce como Código Byte Java y puede ser ejecutado por el entorno de ejecución de Java.

La invención también es aplicable a otros lenguajes de programación y entornos de tiempo de ejecución. La invención permite, por ejemplo, el código fuente escrito en el lenguaje de programación COBOL para transformarse en el lenguaje de programación, ”VB.NET” que a su vez se puede ejecutar en el entorno de tiempo de ejecución “.NET”.

Descripción detallada del segundo aspecto de la invención

La figura 1a muestra la ejecución de ejemplo para el método propuesto por esta invención que permite transferir el código fuente en el lenguaje de programación original en código que es ejecutable por un entorno de tiempo de ejecución. El ejemplo transforma el código fuente en el lenguaje de programación Natural en código que es ejecutable por el JVM. El lenguaje de programación Java no permite instrucciones de salto.

El código fuente Natural 1 contiene una o más instrucciones de salto. Por ejemplo, el código fuente Natural puede contener una o varias instrucciones “REINPUT” que provocan un salto de regreso a un destino en el código en el que se le solicitó al usuario la última entrada”. En Natural, se puede solicitar al usuario la instrucción INPUT. La instrucción “REINPUT” se puede colocar arbitrariamente dentro del código fuente Natural.

El código fuente Natural 1 puede contener varias subrutinas que pueden anidarse. La instrucción “REINPUT” puede provocar un salto a un punto de ejecución fuera de la subrutina que ejecuta la instrucción “REINPUT”.

En un primer paso S1, el código fuente Natural 1 que contiene una o más instrucciones de salto se transforma en el código 2 fuente Java, por lo que la fuente resultante Java no contiene ninguna instrucción de salto correspondiente. La transformación del código fuente Natural 1 en el código 2 fuente Java se puede lograr a través de un compilador que, por ejemplo, convierte las declaraciones o funciones Naturales en declaraciones o funciones Java correspondientes.

Como Java es un lenguaje orientado a objetos, tiene sentido transformar el código fuente Natural en código fuente Java que consiste en clases Java. Las subrutinas en Natural se transforman preferentemente en métodos de Java.

La traducción del código fuente en el paso S 1 se realiza mejor mediante la creación del código 2 fuente Java que es sintácticamente similar al código fuente Natural 1. Esto se logra haciendo una transformación línea por línea o de declaración a declaración. Se crea así un alto grado de código fuente reutilizable y sostenible que muestra un fuerte parecido entre el código fuente 1 y el código fuente 2.

Como Java no tiene instrucciones de salto que permitan la transformación directa de la declaración “REINPUT”, el código fuente 2 se crea sin las instrucciones de salto correspondientes. El Código de Bytes Java se extiende insertando fragmentos de código que mostrarán el comportamiento de la instrucción “REINPUT” una vez ejecutado (ver el paso S5).

Durante la transformación del código fuente Natural 1 en el código 2 fuente Java, un paso S3 conduce a la creación de datos de control para cada instrucción de salto identificada en el código fuente Natural 2 que permite que la unidad de manipulación de código intermedio inserte fragmentos de código en el Código de Bytes Java que son funcionalmente equivalentes a las instrucciones de salto encontradas en el código fuente Natural 1. Los datos de control identifican en qué parte del Código Byte Java deben insertarse los fragmentos de código. Puede guardarse en un archivo de control que utiliza la unidad de manipulación de códigos intermedia cuando cambia el Código de Bytes Java, que se muestra en el paso S6. Sin embargo, los datos de control del paso S3 también pueden almacenarse en la memoria principal del sistema informático.

Después de transformar el código fuente Natural 1 en el código 2 fuente Java mientras se omiten las instrucciones de salto, el código 2 fuente Java se compila con un compilador Java. El resultado es uno o más archivos .class 3 que contienen Código de Bytes. Como el código fuente 2 Java no contenía ninguna instrucción de salto, el Código 3 de byte Java tampoco contiene ninguna instrucción de salto equivalente a las encontradas en el código fuente Natural 1.

En el siguiente paso S4, los archivos .class, es decir, Código 3 de byte Java, se introducen en una unidad de manipulación de código 5 intermedio que cambia el Código 3 de byte Java e inserta fragmentos de código en él como el paso S5 que son funcionalmente equivalentes a las instrucciones de salto que se encuentran en el código fuente Natural 1. La unidad de manipulación de código 5 intermedio en un paso S6 pasa por los datos de control alcanzados en el paso S3. La unidad de manipulación de código 5 intermedio luego inserta instrucciones de salto en el Código 3 de bytes de Java de acuerdo con estos datos.

El método exacto para insertar fragmentos de código y/o instrucciones de salto en el Código 3 de bytes de Java se explicará ahora haciendo referencia a la Fig. 2.

Después de insertar instrucciones de salto en el Código 3 de byte Java, está disponible un Código Byte Java 4 que es semánticamente por completo equivalente al Código fuente Natural 1. Este Código 4 de Byte Java puede ser ejecutado por la JVM.

En una realización de la invención, la unidad de manipulación de código 5 intermedio que inserta las instrucciones de salto en el Código 3 de byte Java puede codificarse para que sea una unidad de manipulación que sea independiente y desacoplada del Cargador de clase de JVM. Esto se logra mediante una unidad de manipulación del código 5 intermedio que crea archivos “.class” que se pasan al Cargador de clase de JVM y, a su vez, se entregan al intérprete de JVM. La ventaja de dicha implementación es que el Código Byte Java es semánticamente idéntico al código fuente Natural 1 y no necesita ser manipulado más por el cargador de clase de JVM para insertar las instrucciones de salto respectivas en el Código Byte Java. Además, los archivos “.class” creados por la unidad de manipulación desacoplada o el Código Byte Java creado por la unidad de manipulación de código 5 intermedio pueden ejecutarse en entornos de tiempo de ejecución de Java arbitrarios sin requerir un Cargador de Clase específicamente modificado o una unidad de manipulación de tiempo de ejecución.

Otra realización de la invención puede usar una unidad de manipulación de código 5 intermedio que se ha derivado del cargador de clase de JVM y, si es necesario, sobrescribe y reemplaza. Esta unidad de manipulación de tiempo de ejecución derivada comprende la funcionalidad del cargador de clase de JVM y la funcionalidad requerida para insertar fragmentos de código y/o instrucciones de salto en el Código 3 de bytes de Java. Por lo tanto, la unidad de manipulación de código intermedio se convierte en un componente de la JVM tal que las instrucciones de salto respectivas se insertan en Código 3 de byte Java después de que la unidad de manipulación de código intermedio derivada del cargador de clase de JVM haya cargado el Código 3 de byte Java. Los diversos entornos de tiempo de ejecución de Java para diferentes plataformas requieren cada uno la misma unidad de manipulación de tiempo de ejecución.

La Fig. 1b muestra un ejemplo de ejecución de un método de acuerdo con la invención. La diferencia con el método mostrado en la figura 1a es que no se generan datos de control para alimentar a la unidad de manipulación de código 5 intermedio para insertar fragmentos de código y/o instrucciones de salto.

Al igual que en la ejecución del método de acuerdo con la Fig. 1a en un paso S1, inicialmente el código fuente Natural que contiene instrucciones de salto se transforma en código fuente Java. Para cada instrucción de salto encontrada durante la transformación del código fuente Natural 1 en el código 2 fuente Java, se inserta un marcador de salto en el código 2 fuente Java que representa una instrucción de salto. Los fragmentos de código que son funcionalmente equivalentes a la instrucción de salto en el código fuente Natural 1 serán insertados en los marcadores de salto en el Código 3 de bytes de Java por la unidad de manipulación de código 5 intermedio. Una implementación de la invención puede insertar una función de Java como el marcador de salto que codifica para lanzar una excepción, como se describe en la Fig. 2 con mayor detalle. Alternativamente, el marcador de salto se puede insertar como un autentificador, tal como un comentario de Java.

El código 2 fuente de Java creado de esta manera en un próximo paso S2 se compila a Código de Byte Java mediante un compilador de Java. Este código compilado también contiene los marcadores insertados en el paso S1.

En el siguiente paso S4, el Código 3 de byte Java se pasa a la unidad de manipulación de código 5 intermedio. Se determina en el siguiente paso S5 los marcadores de salto contenidos en el Código Byte Java. La manipulación del código intermedio procede a reemplazar los marcadores de salto por fragmentos de código que son funcionalmente equivalentes a las instrucciones de salto presentes en el código fuente Natural 1. La sustitución es específicamente necesaria si los marcadores de salto se insertaron como autentificadores o comentarios en el código 2 fuente Java. Si los marcadores de salto se insertaron como funciones de Java, la manipulación de código 5 intermedio puede modificar adicionalmente el Código 3 de byte Java mediante fragmentos de código adicionales tales que éstos, en combinación con la función de Java, se vuelven funcionalmente equivalentes a las instrucciones de salto en el código fuente Natural 1. El Código de Bytes Java resultante puede ser ejecutado por la JVM.

El marcador de salto insertado en el paso S1 puede ser un “REINPUT” () de función java que básicamente lanza una excepción que lo hace sintácticamente cercano a una instrucción “REINPUT” en el código fuente Natural 1. La unidad de manipulación de código 5 intermedio agrega el manejo de excepción correspondiente para capturar y tratar la excepción lanzada por la función “REINPUT” () al Código 3 de byte Java. Además, la unidad de manipulación de código 5 intermedio inserta las instrucciones de salto correspondientes y sus destinos al Código 3 de bytes de Java, como se muestra con más detalle en la Fig. 2.

En el caso que los marcadores de salto se hayan insertado en forma de autenticadores o comentarios en el paso S1, la unidad de manipulación de código intermedio los reemplaza por una función que arroja una excepción similar a la “reinput” () de la función Java.

La Fig. 2 es un ejemplo de cómo los fragmentos de código se insertan en el Código de Bytes Java para obtener instrucciones de salto que son semánticamente similares a las del Código fuente Natural1. Para una mejor legibilidad, esto se muestra en la sintaxis de Java. Los fragmentos de código se insertan dentro de S5 en Código 3 de byte Java mediante la unidad de manipulación de código 5 intermedio de acuerdo con la invención como se describe haciendo referencia a la Fig. 1a y Fig. 1b.

El fragmento de código MAIN corresponde a un programa principal en el código fuente Natural 1. La función FUNCTION A corresponde a una subrutina en el código fuente Natural 1. El programa Principal MAIN llama (C1) a la función FUNCTION A implementada como un método de la clase Java que representa el programa principal. Una función adicional FUNCTION B también corresponde a una subrutina en el código fuente Natural 1. FUNCTION A llamada FUNCTION B, C2. Dentro del código fuente natural original, la subrutina representada por FUNCTION A llama a la subrutina representada por FUNCTION B. FUNCTION B también se implementa como un método de la clase Java que representa el programa principal. La función FUNCTION B puede ser la función de “reinput” de Java () mencionada anteriormente.

El código fuente natural original 1 contiene una declaración de “reinput” (en la subrutina que corresponde a FUNCTION A) que salta a un punto en el programa principal que está fuera de la subrutina. Este punto se marca como "Target" en la Fig. 2.

A diferencia del código fuente de Java, el Código de Bytes Java permite que las declaraciones de salto salten de un punto a otro. Sin embargo, estos saltos solo son válidos dentro de un bloque de ejecución o el alcance del código del programa Java. Este alcance o bloque puede estar limitado, por ejemplo, a un método. Por lo tanto, el salto solo es válido dentro de este método. Tales saltos se ejecutan utilizando la instrucción “GOTO” en el Código Byte Java.

Para permitir todas las instrucciones de salto posibles en el código fuente Natural 1 en el Código de Bytes Java, esta instrucción “GOTO” por sí sola no es suficiente. No permite saltar desde las llamadas de método o los métodos complicados hacia el exterior.

Para permitir saltos fuera de tales ámbitos o bloques, la invención utiliza el manejo de excepciones.

Dentro de la función en el código fuente Natural representado por la función FUNCTION A se encuentra una declaración “REINPUT” que se ejecutará para saltar a un destino en el código hasta la última declaración INPUT ejecutada. En el punto de la declaración “REINPUT” dentro de la función FUNCTION A en el Código Byte Java se lanza una excepción. La excepción se lanza al llamar a la función FUNCTION B.

La función FUNCTION A tiene un manejador de excepciones que captura la excepción lanzada por la función FUNCTION B y la maneja. Las excepciones de supervisión lanzadas cuando se llama a la función FUNCTION A se realizan con un bloque TRY. Si la función FUNCTION B provoca una excepción, el bloque “CATCH” correspondiente de función FUNCTION A lo atrapa y lo maneja. El manejo de acuerdo con la invención, la excepción se pasa al que llama de la función FUNCTION A. Este paso de la excepción se logra lanzando una excepción dentro del bloque “CATCH”. El ejemplo de la Fig. 2 muestra cómo la función FUNCTION A captura la excepción causada por la función FUNCTION B y pasa al programa principal MAIN, J2.

Si la excepción causada por la función FUNCTION B es una función de excepción en tiempo de ejecución, FUNCTION A puede no requerir un controlador de excepciones porque la excepción será capturada y manejada, a más tardar, por el manejador de excepciones del programa principal MAIN, tal como se describe a continuación. Sin embargo, si se requiere un manejo de excepciones de tiempo de ejecución específico, un controlador de excepciones en la función FUNCTION A aún puede ser ventajoso.

El programa principal MAIN también contiene un manejador de excepciones que captura y maneja la excepción lanzada por la función FUNCTION A o lanzada por la función FUNCTION B y no capturada/manejada por la función FUNCTION A. La captura y manejo de la excepción causada por la función FUNCTION A se realiza en el bloque CATCH del programa principal MAIN. Dado que la ejecución del programa está dentro del bloque de ejecución o del alcance del objetivo de salto, el manejo de excepciones en el programa principal MAIN no tiene que arrojar una excepción adicional, pero puede usar la instrucción “GOTO” para saltar al objetivo "Target".

Los manejadores de excepciones, las declaraciones para lanzar excepciones y las declaraciones “GOTO” mostradas en la Fig. 2 se insertan como Código Byte Java en el Código 3 de byte Java por la unidad de manipulación de código 5 en el paso S5. El método que se muestra en la Fig. 1b solo requiere insertar el controlador de excepciones, las declaraciones “GOTO” y los objetivos "Target". Lanzar las excepciones, es decir, la función “reinput” () ya se ha insertado en el código del código fuente Java S1.

El Código Byte Java modificado de esta manera ahora contiene una serie de manejadores de excepciones y declaraciones “GOTO” que corresponden a las instrucciones “REINPUT” en el código fuente Natural 1, incluidos los saltos más allá de los bloques de ejecución o áreas de alcance.

Por supuesto, el manejo de excepciones no siempre requiere volver a saltar la función principal MAIN. Si el objetivo de la instrucción de salto está dentro de la función FUNCTION A, es suficiente volver a la función FUNCTION A de la función FUNCTION B utilizando una excepción y luego ejecutar una instrucción “GOTO” dentro del bloque de “CATCH” respectivo en lugar de lanzar una excepción.

Si el código fuente Natural 1 (o el código fuente original en un lenguaje de programación diferente tal como COBOL o PL/1) contiene diferentes tipos de instrucciones de salto, puede ser ventajoso proporcionar varios bloques “CATCH” para el manejo de excepciones. La excepción puede manejarse de manera diferente dependiendo del tipo de instrucción de salto.

La Fig. 3a muestra un primer sistema que ha sido modificado para ejecutar el método de acuerdo con la invención. Desde dentro de un entorno de desarrollo, el código fuente Natural que contiene instrucciones de salto se transforma en el código 2 fuente Java, si corresponde, sin instrucciones de salto. Si se crea el código fuente de Java eliminando las instrucciones de salto, también se generan datos de control, como se menciona en la figura 1a. El código fuente de Java se compila con un compilador para crear el Código Byte Java, como uno o varios archivos “.class”. El entorno de ejecución comprende Máquina Virtual de Java JVM. Los archivos “.class” creados por el compilador se cargan en una unidad de manipulación de código intermedio (desacoplada) que inserta los fragmentos de código respectivos y/o las instrucciones de salto en el Código Byte Java. Los fragmentos de código corresponden funcionalmente a las instrucciones de salto en el código fuente Natural y pueden implementarse como manipuladores de excepción, como se muestra al hacer referencia a la Fig. 2. Si el código fuente de Java se generó omitiendo las instrucciones de salto como en la Fig. 1a, la unidad de manipulación de código desacoplada también recibe los datos de control creados anteriormente. De lo contrario, como en la Fig. 1b, la información requerida está contenida en el Código 3 de byte Java. El resultado del proceso de la unidad de manipulación desacoplada son los archivos ".class" que se han complementado con instrucciones de salto. Los archivos “.class” luego son leídos por el cargador de clase de JVM y pasados al intérprete de la JVM para su ejecución.

La Fig. 3b muestra un segundo sistema que se ha modificado para ejecutar el método dado por la invención. La diferencia con el sistema dado en la Fig. 3a es básicamente que la unidad de manipulación de código intermedio (la unidad de manipulación desacoplada en la Fig. 3a) es un reemplazo de la unidad de manipulación en tiempo de ejecución para el cargador de clase JVM. En el caso que se muestra aquí, la unidad de manipulación de tiempo de ejecución es un derivado del cargador de clase JVM, es decir, además de la funcionalidad proporcionada por el cargador de clase JVM, contiene funcionalidad adicional para modificar el Código Byte Java que se le ha pasado. En una forma concreta, la funcionalidad del cargador de clase JVM de cargar el código de byte puede sobrescribirse, de modo que cargar el Código Byte Java mediante la unidad de manipulación de tiempo de ejecución ya modifica el Código Byte Java.

La ventaja de derivar de la carga de clase JVM es el poco esfuerzo requerido para crear la unidad de manipulación de tiempo de ejecución. En una forma diferente, la unidad de manipulación de tiempo de ejecución se crea de forma completamente independiente, pero tiene la misma funcionalidad y objeto que se explica en la Fig. 3b.

La compilación del código 2 fuente Java por un compilador en uno o más archivos 3 “.class”, así como la modificación del Código 3 de byte Java a través de la unidad de manipulación en tiempo de ejecución, se realiza como se muestra en la Fig. 3a. Aquí también, siempre que el código 2 fuente Java se haya creado a partir del código fuente Natural 1

omitiendo las instrucciones de salto como se muestra en la figura 1a, se pueden crear datos de control que se dan a la unidad de manipulación de tiempo de ejecución. La unidad de manipulación de tiempo de ejecución puede insertar fragmentos de código en Código 3 de byte Java mediante el uso de los datos de control que son funcionalmente equivalentes a las instrucciones de salto en el código fuente Natural 1. De lo contrario, como en la Fig. 1b, toda la información necesaria está presente en el Código 3 de byte Java.

Descripción detallada del primer aspecto de la invención.

La Fig. 4 muestra la ejecución de ejemplo para el método propuesto por el primer aspecto de la invención que transfiere el código fuente en el lenguaje de programación original a un código que es ejecutable por un entorno de tiempo de ejecución. En el ejemplo, el código fuente en el lenguaje de programación Natural se transforma en código que es ejecutable por la JVM. El lenguaje de programación Java, como se describe anteriormente, no permite instrucciones de salto.

El código fuente Natural 1 contiene una o más instrucciones de salto. Por ejemplo, el código fuente Natural 1 puede contener una o varias instrucciones “REINPUT” que provocan un salto de regreso a un destino en el código donde se le solicitó al usuario la última entrada. En Natural, se puede solicitar al usuario la instrucción “INPUT”. La instrucción “REINPUT” se puede colocar arbitrariamente dentro del código fuente Natural.

El código fuente Natural 1 puede contener varias subrutinas que pueden anidarse. La instrucción “REINPUT” puede provocar un salto a un punto de ejecución fuera de la subrutina que ejecuta la instrucción “REINPUT”.

En un primer paso S1, el código fuente Natural 1 que contiene una o más instrucciones de salto se transforma en el código 2 fuente Java, por lo que el código 2 fuente Java resultante no contiene ninguna instrucción de salto correspondiente. La transformación del código fuente Natural 1 en el código 2 fuente Java se puede lograr a través de un compilador que, por ejemplo, convierte declaraciones o funciones Natural en declaraciones o funciones Java correspondientes.

La traducción del código fuente en el paso S 1 se realiza mejor mediante la creación del código 2 fuente Java que es sintácticamente similar al código fuente Natural 1. Esto se logra haciendo una transformación línea por línea o una de estado a estado. Por lo tanto, se crea un alto grado de código fuente reutilizable y sostenible que muestra una fuerte semejanza entre el código 1 fuente y el código fuente 2.

Como Java no tiene instrucciones de salto que permitan la transformación directa de la instrucción “REINPUT”, el código fuente 2 se crea sin las instrucciones de salto correspondientes. El código fuente de Java 2 se extiende insertando fragmentos de código que mostrarán el comportamiento de la instrucción “REINPUT” una vez ejecutado (ver el paso S5).

En el siguiente paso S2, el código 2 fuente Java se alimenta en una unidad 3 de manipulación de código que cambia el código 2 fuente Java al insertar fragmentos de código en él (como paso S3) que son funcionalmente equivalentes a las instrucciones de salto encontradas en el código fuente Natural 1.

Después de manipular el código fuente de Java 2, está disponible un código fuente Java 4 que es semánticamente por completo equivalente al código fuente Natural 1. Este código fuente Java 4 puede compilarse y ejecutarse por la JVM

Con más detalle, en un paso S1, el código fuente Natural 1 que contiene las instrucciones de salto se transforma en el código 2 fuente Java. Para cada instrucción de salto encontrada durante la transformación del código fuente Natural 1 en el código 2 fuente Java, se inserta un marcador de salto en el código 2 fuente Java que representa una instrucción de salto. Los fragmentos de código que son funcionalmente equivalentes a las instrucciones de salto en el código fuente Natural 1 se insertarán en los marcadores de salto en el código fuente Java 4 mediante la unidad de manipulación de códigos 3. Una implementación de la invención puede insertar una función de Java como marcador de salto que codifica para lanzar una excepción, como se describe con referencia a la figura 5a con mayor detalle. Alternativamente, el marcador de salto se puede insertar como un autentificador, como un comentario de Java. Además de la excepción, se insertan las declaraciones de bucle y las declaraciones de conmutación para seguir manejando la excepción.

El marcador de salto insertado en el paso S1 puede ser una función de Java que básicamente arroja una excepción que lo hace sintácticamente cercano a una instrucción “REINPUT” en el código fuente Natural 1. La unidad de manipulación de código 3 agrega el manejo de excepciones correspondiente para capturar y tratar la excepción lanzada por dicha función Java. Además, la unidad de manipulación de código 3 inserta las instrucciones de salto correspondientes y sus destinos al código 4 fuente de Java, como se muestra con más detalle en la Fig. 5a y Fig. 5b.

El método exacto para insertar fragmentos de código que son funcionalmente equivalentes al código fuente original también se explicará haciendo referencia a las Figuras 5a y 5b. Las Figuras 5a y 5b son ejemplos de cómo los fragmentos de código se insertan en el código 4 fuente de Java para obtener instrucciones de salto que son

semánticamente similares a las del Código fuente Natural 1. Los fragmentos de código se insertan dentro de S3 mediante la unidad de manipulación de códigos 3 como se describe con referencia a la Fig. 4.

El fragmento de código A2 en la Fig. 5a muestra un código fuente de Java que corresponde a un código fuente Cobol1 (o PL/1 y que incluye marcadores de salto). El código fuente Cobol original 1 incluye una Instrucción “GOTO” para una posición específica. Esta instrucción “GOTO” se transforma en un Método "gotoLabel", que representa la declaración “GOTO” original y un Método "Label", que marca el destino del salto.

El fragmento de código A3 en la Fig. 5a muestra un código fuente de Java que incluye el fragmento de código equivalente a la instrucción de salto en el código fuente Cobol original 1.

La instrucción "throw new GotoException" arroja una excepción que detiene el procesamiento. El bloque “catch” correspondiente captura la excepción y la maneja. El código para el manejo de excepciones (“try and catch”) encierra el código fuente de Java. Para continuar el procesamiento en el destino de la instrucción se inserta una declaración de bucle (“while (*true*) and continue”) dentro del código para el manejo de excepciones. Además de la declaración de bucle, se inserta una declaración condicional (“switch case”) dentro del bucle. El código que pertenece a las ramas del caso de las declaraciones de cambio maneja el procesamiento en el destino del salto. El fragmento de código B2 en la Fig. 5b muestra un código 2 fuente Java que corresponde a un código fuente Natural 1 y que incluye marcadores de salto. El código fuente natural original 1 incluye una declaración de “Reinput” que provoca un salto a la última declaración de ”input”. Los “Reinput” tienen la posibilidad de saltar a una posición dinámica (a la última declaración de “input”) y tienen la posibilidad de saltar a un marco de pila anterior, es decir, saltar a un destino fuera de la función actual.

La declaración “Reinput” del código 1 fuente original se transforma en un método " REINPUT" (dentro de la función B2-FUNCTION A) y el “INPUT-Statement” del código 1 fuente original se transforma en un método "INPUT" (dentro de la función B2-MAIN), que marca el destino del salto.

La instrucción "throw new GOTOException" (dentro de la FUNCTION A B3) arroja una excepción que detiene el procesamiento. El bloque “catch” correspondiente (dentro de FUNCTION B3 MAIN) captura la excepción y la maneja. El código para el manejo de excepciones (“try and catch”) encierra el código fuente de Java. Para continuar el procesamiento en el destino del salto, se inserta una instrucción de bucle (“while (true) y continue”) dentro del código para el manejo de excepciones. Además de la instrucción de bucle, se inserta una instrucción condicional (condicional “CASE”) dentro del bucle. El código que pertenece a las ramas “CASE” de la instrucción de interrupción maneja el procesamiento en el destino del salto. El destino del salto, es decir, la última instrucción de ”input” se almacena con una variable auxiliar "__iw$jump$1". La variable auxiliar se evalúa mediante la instrucción del interruptor (interruptor (__iw$jump$1)).

Para activar un salto, la declaración de “Reinput” del código 1 fuente original se reemplaza por una excepción (lanzar nueva “ReinputException”). La excepción lanzada es atrapada por el manejador de excepciones (“Catch (ReinputException)”). En el ejemplo de código B3-FUNCTION A, el destino del salto está fuera de la función, es decir, fuera del marco de pila de la función. Por lo tanto, cuando se lanza la excepción dentro de la función B3-FUNCTION A, la pila se desenrolla (es decir, las ”input” se eliminan de la pila) hasta que se encuentra un manejador de excepciones (en el ejemplo, el manejador de excepciones respectivas se encuentra en la función B3-MAIN) que está preparado para manejar la excepción (Catch). La excepción es capturada por el manejador de excepciones manejado como se describe con referencia a la Fig. 5a.

La Fig. 6 muestra un sistema que ha sido modificado para ejecutar el método de acuerdo con el primer aspecto de la invención. Desde un entorno de desarrollo, el código fuente Natural que contiene instrucciones de salto se transforma en el código 2 fuente Java que incluye marcadores de salto. El código fuente de Java se manipula con una unidad de manipulación (precompilador) para crear el código 4 fuente de Java que contiene fragmentos de código funcionalmente equivalentes a las instrucciones de salto en el código fuente natural. El código 4 de fuente manipulado es compilado por un compilador para crear Código Byte Java que es ejecutable por una Máquina Virtual de Java.

El entorno de tiempo de ejecución (ejecución) comprende la Máquina Virtual de Java JVM o un entorno de ejecución “NET”. Los archivos “.class” creados por el compilador se cargan en Máquina Virtual de Java y se ejecutan mediante Máquina Virtual de Java.

Las técnicas actuales pueden implementarse en circuitos electrónicos digitales, o en hardware de ordenador, firmware, software o en combinaciones de ellos. El aparato de la invención puede implementarse en un producto de programa informático tangiblemente incorporado en un dispositivo de almacenamiento legible por máquina para su ejecución mediante un procesador programable. Los pasos del método de acuerdo con la invención pueden ser realizados por un procesador programable que ejecuta un programa de instrucciones para realizar funciones de la invención operando sobre la base de datos de entrada y generando datos de salida. La invención puede implementarse en uno o varios programas informáticos que son ejecutables en un sistema programable, que incluye al menos un procesador programable acoplado para recibir datos de y transmitir datos a un sistema de almacenamiento, al menos un dispositivo de entrada y al menos un dispositivo de salida, respectivamente. Los programas de ordenador pueden implementarse

en un lenguaje de programación de alto nivel u orientado a objetos, y/o en un ensamblador o código de máquina. El lenguaje o el código pueden ser un lenguaje o código compilado o interpretado. Los procesadores pueden incluir microprocesadores de propósito general y especial. Un procesador recibe declaraciones y datos de memorias, en particular de memorias de solo lectura y/o memorias de acceso aleatorio. Un ordenador puede incluir uno o más dispositivos de almacenamiento masivo para almacenar datos; tales dispositivos pueden incluir discos magnéticos, como discos duros internos y discos extraíbles; discos magnetoópticos; y discos ópticos. Los dispositivos de almacenamiento adecuados para incorporar tangiblemente las declaraciones y los datos del programa informático incluyen todas las formas de memoria no volátil, incluyendo a modo de ejemplo dispositivos de memoria de semiconductores, tales como EPROM, EEPROM y dispositivos de memoria “flash”; discos magnéticos tales como discos duros internos y discos extraíbles; discos magneto-ópticos; y discos de CD-ROM. Cualquiera de los anteriores puede complementarse o incorporarse en ASICs (circuitos integrados específicos de la aplicación).

Los sistemas informáticos o las redes informáticas distribuidas, como se menciona anteriormente, pueden usarse, por ejemplo, para producir bienes, entregar piezas para ensamblar productos, controlar procesos técnicos o económicos o implementar actividades de telecomunicaciones.

Para proporcionar la interacción con un usuario, la invención puede implementarse en un sistema informático que tiene un dispositivo de visualización tal como un monitor o pantalla LCD para mostrar información al usuario y un teclado y un dispositivo señalador tal como un mouse o una bola de seguimiento para que el usuario puede proporcionar ”input” al sistema informático. El sistema informático puede programarse para proporcionar una interfaz de usuario gráfica o de texto a través de la cual los programas interactúan con los usuarios.

Un ordenador puede incluir un procesador, una memoria acoplada al procesador, un controlador de disco duro, un controlador de video y un controlador de entrada/salida acoplado al procesador por un bus de procesador. El controlador del disco duro está acoplado a una unidad de disco duro adecuado para almacenar programas de ordenador ejecutables, incluidos los programas que incorporan la técnica actual. El controlador de I/O se acopla mediante bus I/O a una interfaz de I/O. La interfaz de I/O recibe y transmite en forma analógica o digital por al menos un enlace de comunicación. Dicho enlace de comunicación puede ser un enlace en serie, un enlace paralelo, una red de área local o un enlace inalámbrico (por ejemplo, un enlace de comunicación de RF). Un enlace de comunicación de RF). Una pantalla está acoplada a una interfaz, que está acoplada a un bus de I/O. Un teclado y un dispositivo señalador también están acoplados al bus I/O. Alternativamente, se pueden usar buses separados para el dispositivo señalador de teclado y la interfaz de I/O.

REIVINDICACIONES

1. Un método implementado por ordenador para generar un código a partir del código (1) fuente original, mediante el cual el código (1) fuente original incluye al menos una instrucción de salto y existe en un lenguaje de programación de fuente, comprendiendo el método:

- crear un código (2) intermedio a partir del código (1) fuente original omitiendo al menos una instrucción de salto, existiendo el código (2) intermedio en un lenguaje de programación de destino,

- cargar (S2) el código (2) intermedio en una unidad (3) de manipulación de código, y

- crear (S3), a través de la unidad (3) de manipulación de código, del código (4) fuente de destino desde el código (2) intermedio, en donde la creación comprende una inserción de al menos un fragmento de código fuente en el código (4) fuente de destino, por lo que el al menos un fragmento de código fuente insertado es funcionalmente equivalente a al menos una instrucción de salto omitida, y por lo que al menos un fragmento de código fuente insertado comprende

- una instrucción que arroja una excepción,

- al menos una declaración de bucle dentro del código fuente para el manejo de excepciones,

- al menos una declaración condicional dentro de un bloque “Try” dentro de la declaración de bucle,

- un bloque de captura correspondiente, que captura la excepción, y vuelve a la declaración de bucle, y

- una variable auxiliar que contiene el destino del salto y que es evaluada por la declaración condicional.

2. Un método implementado por ordenador para generar un código a partir del código (1) fuente original, mediante el cual el código (1) fuente original incluye al menos una instrucción de salto y existe en un lenguaje de programación de origen, comprendiendo el método:

- crear un código (2) intermedio a partir del código (1) fuente original mediante la sustitución de al menos una instrucción de salto por al menos un marcador de salto, el código (2) intermedio existente en un lenguaje de programación de destino,

- cargar en (S2) el código (2) intermedio en una unidad (3) de manipulación de código, y

- crear en (S3), a través de la unidad (3) de manipulación de código, el código (4) fuente de destino del código (2) intermedio, donde la creación comprende una inserción de al menos un fragmento de código ~~1~~ fuente en el código (4) fuente de destino, por lo que el al menos un fragmento de código fuente insertado es funcionalmente equivalente a la al menos una instrucción de salto reemplazada, y por lo cual el al menos un fragmento de código fuente insertado comprende

- una instrucción que arroja una excepción,

- al menos una declaración de bucle dentro del código fuente para el manejo de excepciones,

- al menos una declaración de cambio dentro de un bloque “Try” dentro de la declaración de bucle,

- un bloque de captura correspondiente que captura la excepción, y vuelve a la declaración de bucle, y

- una variable auxiliar que contiene el destino del salto y que se evalúa mediante la declaración condicional.

3. Un método de acuerdo con la reivindicación 1 o 2, en el que el al menos un fragmento de código fuente insertado en el código (4) fuente de destino realiza declaraciones de programa que son funcionalmente equivalentes:

- a una instrucción de salto estática colocada en el código fuente original, si el destino de la instrucción de salto está dentro del marco de pila de la instrucción de salto, y/o

- a una instrucción de salto dinámica colocada en el código fuente original, si el destino de la instrucción de salto está fuera del cuadro de pila de la instrucción de salto, y/o

- a una instrucción de salto dinámica colocada en el código fuente original, si el destino de la instrucción de salto está dentro del marco de pila de la instrucción de salto.

4. Un método de acuerdo con una de las reivindicaciones anteriores, en el que el lenguaje de programación de destino comprende:

- un lenguaje de programación que no permite instrucciones de salto del tipo “GOTO” en su código fuente, y/o

- al menos uno de Java, y un lenguaje de programación que es ejecutable en el entorno de tiempo de ejecución .NET y/o

por lo que el lenguaje de programación fuente comprende

- un lenguaje de programación que permite instrucciones de salto en su código fuente, y/o

- al menos uno de Cobol, Natural y PL/1.

5. Un método de acuerdo con una de las reivindicaciones anteriores, comprendiendo además el método un paso de crear el código (5) final a partir del código (4) fuente de destino, por lo que el código (5) final comprende un código independiente de hardware, estando el código (5) final para su ejecución en un entorno de tiempo de ejecución.

6. Una unidad (3) de manipulación de código para generar un código, por lo que la unidad (3) de manipulación de código está adaptada para cargar el código (2) intermedio que se ha creado a partir del código (1) fuente original, por lo que el código (1) fuente original incluye en al menos una instrucción de salto, omitir al menos una instrucción de salto y generar un código (4) fuente de destino a partir del código (2) intermedio, por lo que la generación del código (4) fuente de destino comprende la inserción de al menos un fragmento de código fuente en el código (4) fuente de destino, en el que al menos un fragmento de código fuente insertado es funcionalmente equivalente a al menos una instrucción de salto omitido, y por lo que al menos un fragmento de código fuente insertado comprende una instrucción que arroja una excepción, en al menos una instrucción de bucle dentro del código fuente para el manejo de excepciones, al menos una declaración condicional dentro de un bloque de prueba dentro de la instrucción de bucle, un bloque de captura correspondiente que captura la excepción, y regresa al bloque de prueba, y una variable auxiliar que contiene el destino del salto y que es evaluada por la instrucción condicional.

7. Una unidad (3) de manipulación de código para generar un código, por lo que la unidad (3) de manipulación de código está adaptada para cargar el código (2) intermedio que se ha creado a partir del código (1) fuente original, por lo que el código (1) fuente original incluye en al menos una instrucción de salto, al reemplazar en al menos una instrucción de salto por al menos un marcador de salto, y para generar un código (4) fuente de destino del código (2) intermedio, por lo que la generación del código (4) fuente de destino comprende la inserción de al menos un fragmento de código fuente en el código (4) fuente de destino, por lo que el al menos un fragmento de código fuente insertado es funcionalmente equivalente a al menos una instrucción de salto reemplazada, y por lo que el al menos un fragmento de código fuente insertado comprende una declaración que arroja una excepción, al menos una declaración de bucle dentro del código fuente para el manejo de excepciones, al menos una declaración condicional dentro de un bloque de prueba dentro de la instrucción de bucle, un bloque de captura correspondiente que captura la excepción, y regresa al bloque de prueba, y una variable auxiliar que contiene el destino del salto y que es evaluada por la instrucción condicional.

8. Un sistema para generar un código, que comprende:

- un compilador para la creación (S1) del código (2) intermedio del código (1) fuente original, mediante el cual el código (1) fuente original incluye al menos una instrucción de salto, omitiendo al menos una instrucción de salto o reemplazando a en al menos una instrucción de salto por un marcador de salto, existiendo el código fuente original en un lenguaje de programación de fuente y

- una unidad (3) de manipulación de código de acuerdo con la reivindicación 6 o la reivindicación 7.

9. Un sistema de acuerdo con la reivindicación 8, en el que el al menos un fragmento de código fuente insertado en el código (4) fuente de destino realiza declaraciones de programa que son funcionalmente equivalentes a una instrucción de salto dinámica colocada en el código fuente original, si el destino de la instrucción de salto está fuera del marco de pila de las instrucciones de salto.

10. Un sistema de acuerdo con una de las reivindicaciones 8 a 9, que comprende además medios para crear el código (5) final del código (4) fuente de destino, por lo que el código (5) final comprende un código independiente de hardware, estando el código (5) final para su ejecución en un entorno de tiempo de ejecución.

11. Un producto de programa informático con un código de aplicación que, cuando se carga en una unidad de procesamiento de datos, ejecuta el método de acuerdo con una de las reivindicaciones 1 a 5.

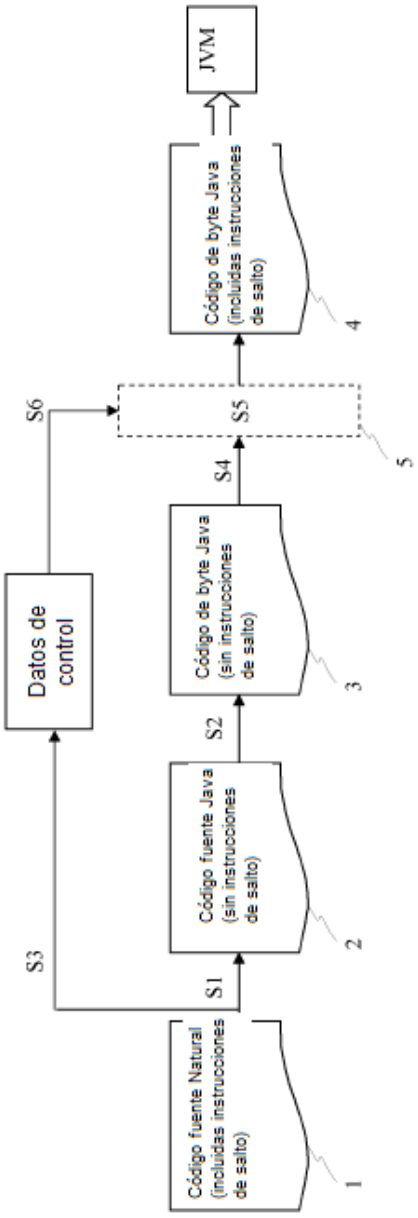
JVM
Código de byte Java (incluidas instrucciones de salto)
4
S6
S5
5
S4
Código de byte Java (sin instrucciones de salto)
3
Datos de control
S2
Código fuente Java (sin instrucciones de salto)
2
S3
S1
Código fuente Natural (incluidas instrucciones de salto)
1

Fig. 1a

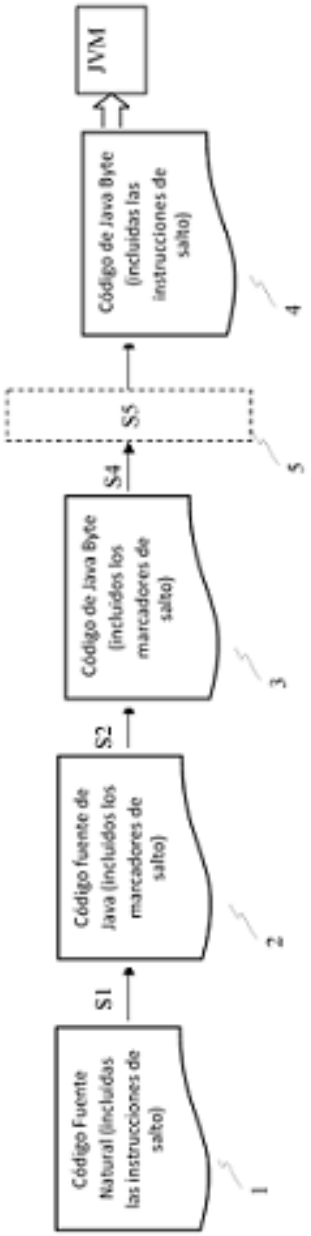
Código Fuente Natural (incluidas las instrucciones de salto)
1
S1
Código fuente de Java (incluidos los marcadores de salto)
2
S2
Código de Java Byte (incluidos los marcadores de salto)
3
S4
S5
5
Código de Java Byte (incluidas las instrucciones de salto)
4
JVM

Fig. 1b

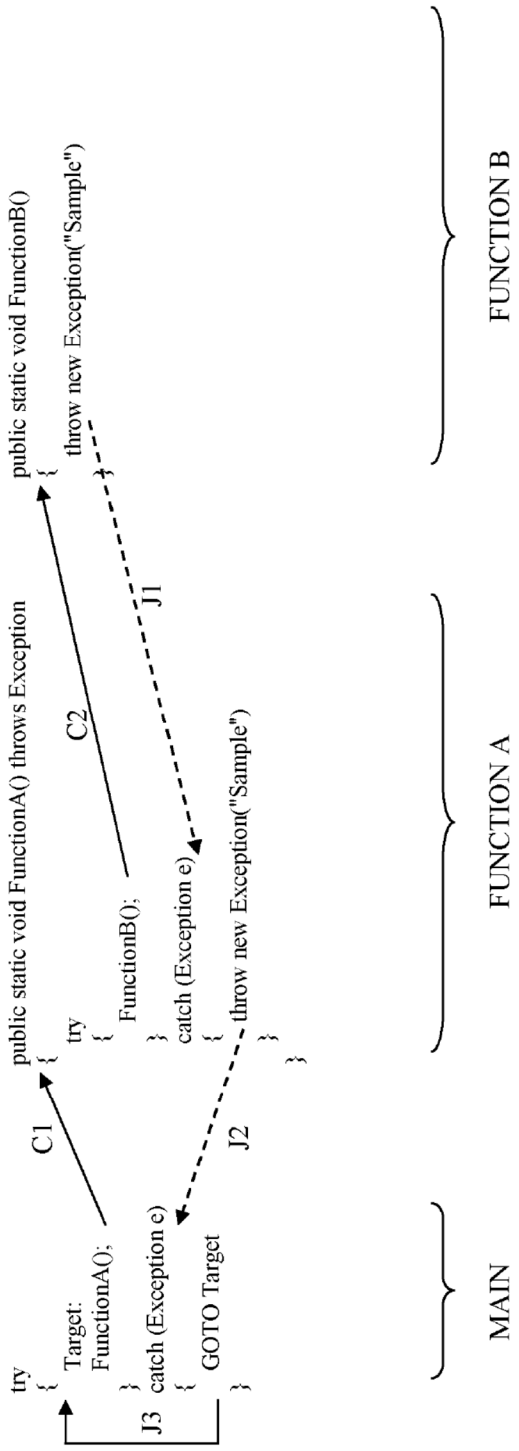
try
{
Target:
FunctionA();
}
J3 catch (Exception e)
{
GOTO Target
}
C1
public static void FunctionA() throws Exception
{
try
{
FunctionB();
}
catch (Exception e)
{
throw new Exception("Sample")
}
}
C2
J2
public static void FunctionB()
{
throw new Exception("Sample")
}
J1
MAIN
FUNCTION A
FUNCTION B

Fig. 2

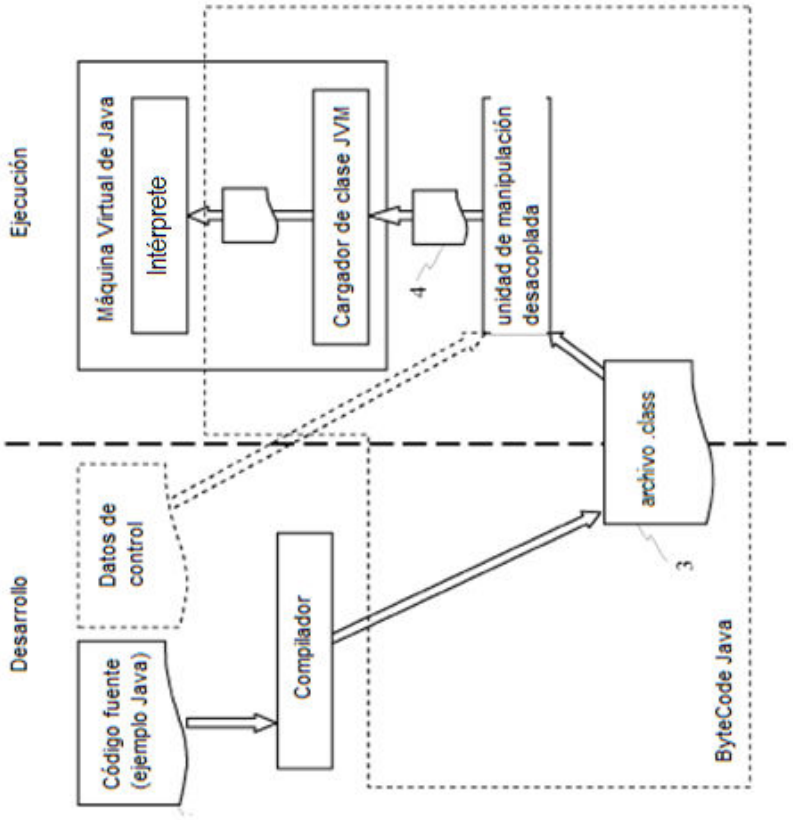
Ejecución
Máquina Virtual de Java
Intérprete
Cargador de clase JVM
4
unidad de manipulación desacoplada
archivo .class
3
Desarrollo
Datos de control
Compilador
Código fuente (ejemplo Java)
ByteCode Java

Fig. 3a

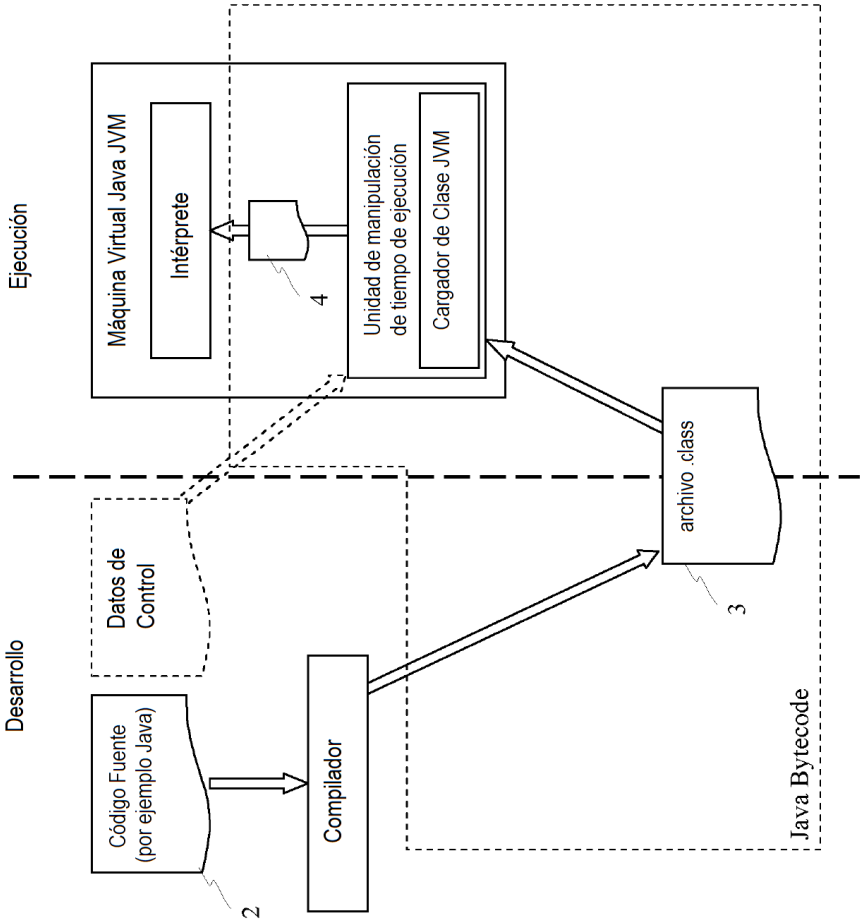
Ejecución
Desarrollo
Máquina Virtual Java JVM
Intérprete
4
Unidad de manipulación de tiempo de ejecución
Cargador de Clase JVM
archivo .class
3
Datos de Control
Código Fuente (por ejemplo Java)
2
Compilador
Java Bytecode

Fig. 3b

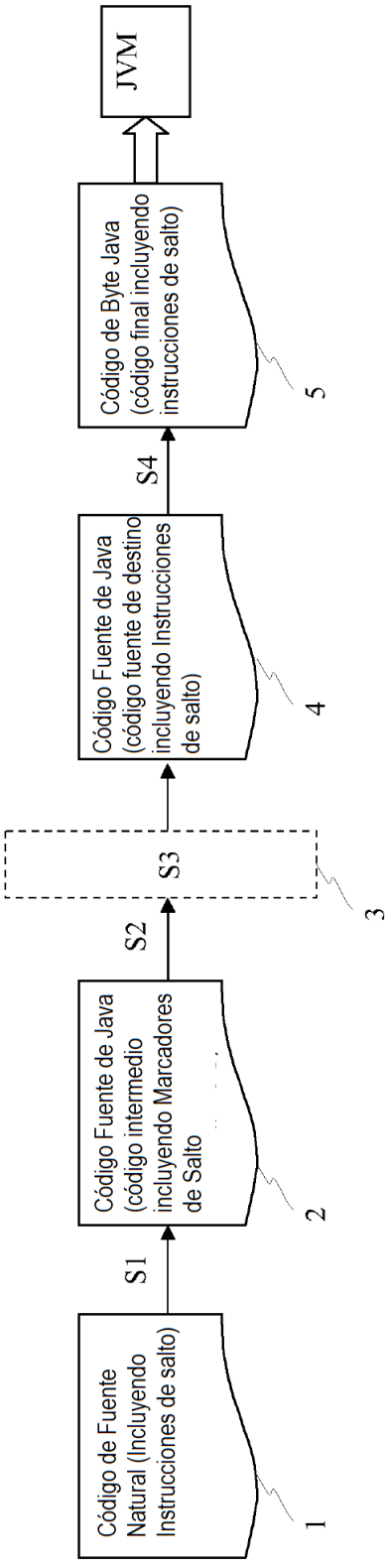
Código de Fuente Natural (Incluyendo Instrucciones de salto)
1
S1
Código Fuente de Java (código intermedio incluyendo Marcadores de Salto
2
S2
S3
3
Código Fuente de Java (código fuente de destino incluyendo Instrucciones de salto)
4
S4
Código de Byte Java (código final incluyendo instrucciones de salto)
5
JVM

Fig. 4

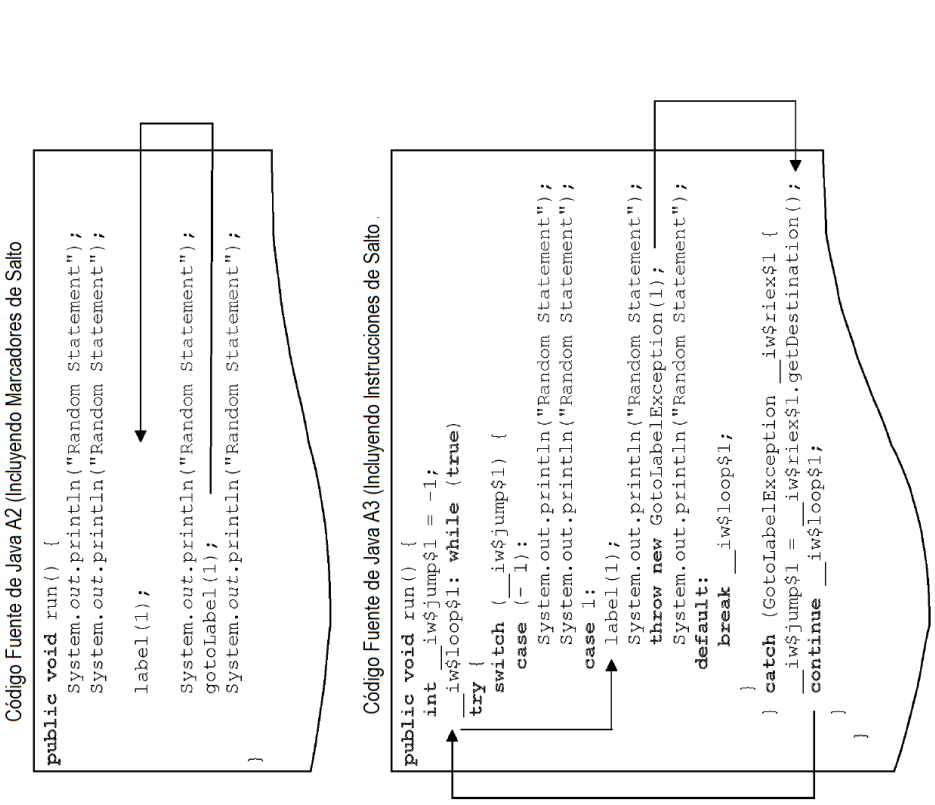
Código Fuente de Java A2 (Incluyendo Marcadores de Salto
public void run() {
System.out.println("Random Statement");
System.out.println("Random Statement");
label(1);
System.out.println("Random Statement");
gotoLabel(1);
System.out.println("Random Statement");
}
Código Fuente de Java A3 (Incluyendo Instrucciones de Salto
public void run() {
int __iw$jump$1 = -1;
__iw$loop$1: while (true)
try {
switch (__iw$jump$1) {
case (-1):
System.out.println("Random Statement");
System.out.println("Random Statement");
case 1:
label(1);
System.out.println("Random Statement");
throw new GotoLabelException(1);
System.out.println("Random Statement");
default:
break __iw$loop$1;
}
} catch (GotoLabelException __iw$riex$1 {
__iw$jump$1 = __iw$riex$1.getDestination();
continue __iw$loop$1;
}
}

Fig. 5a

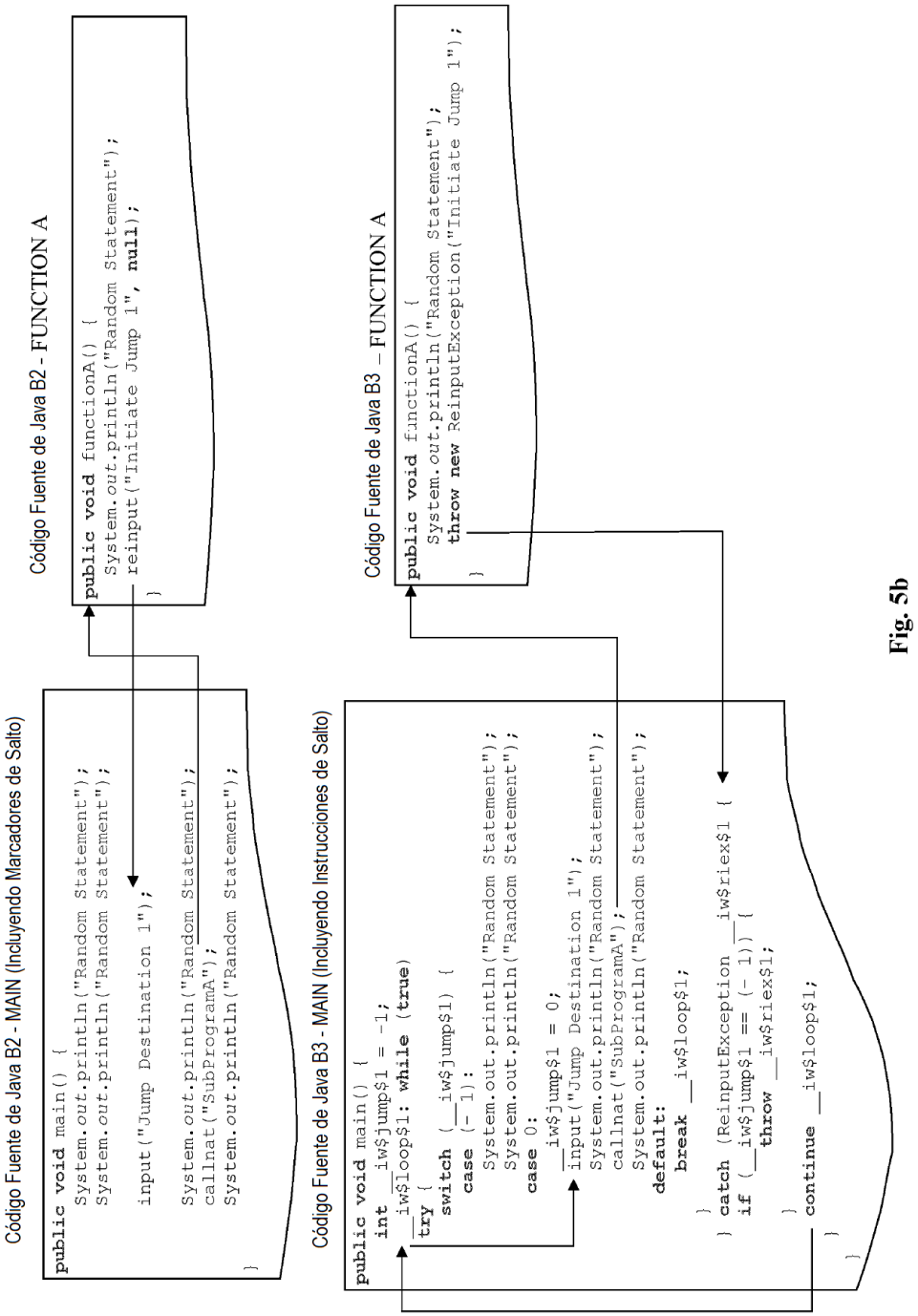
Código Fuente de Java B2 - MAIN (Incluyendo Marcadores de Salto)
public void main() {
System.out.println("Random Statement");
System.out.println("Random Statement");
input("Jump Destination 1");
System.out.println("Random Statement");
callnat("SubProgramA");
System.out.println("Random Statement");
}
Código Fuente de Java B2 - FUNCTION A
public void functionA() {
System.out.println("Random Statement");
reinput("Initiate Jump 1", null);
}
Código Fuente de Java B3 - MAIN (Incluyendo Instrucciones de Salto)
public void main() {
int __iw$jump$1 = -1;
__iw$loop$1: while (true)
try {
switch (__iw$jump$1) {
case (- 1):
System.out.println("Random Statement");
System.out.println("Random Statement");
case 0:
__iw$jump$1 = 0;
input("Jump Destination 1");
System.out.println("Random Statement");
callnat("SubProgramA");
System.out.println("Random Statement");
default:
break __iw$loop$1;
}
} catch (ReinputException __iw$riex$1 {
if (__iw$jump$1 == (- 1)) {
throw __iw$riex$1;
}
continue __iw$loop$1;
}
}
Código Fuente de Java B3 – FUNCTION A
public void functionA() {
System.out.println("Random Statement");
throw new ReinputException("Initiate Jump 1");
}

Fig. 5b

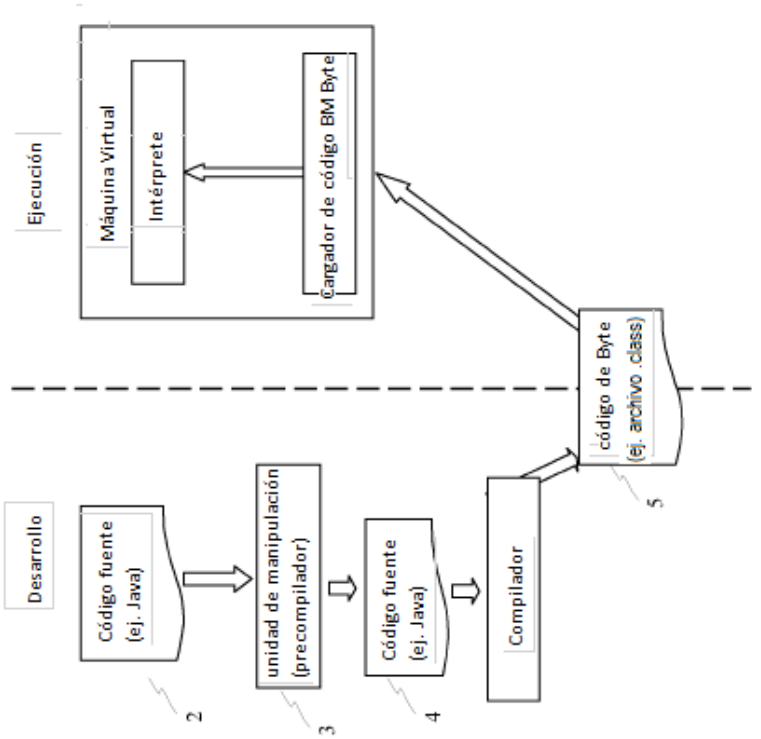
Ejecución
Máquina Virtual
Intérprete
Cargador de código BM Byte
código de Byte
(ej. archivo .class)
5
Desarrollo
Código fuente
(ej. Java)
2
unidad de manipulación
(precompilador)
3
Código fuente
(ej. Java)
4
Compilador

Fig. 6