(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 674 140**

(51) Int. CI.:

G06F 11/36 (2006.01)
G06F 9/45 (2006.01)
G06F 9/44 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(86) Fecha de presentación y número de la solicitud internacional: **31.12.2010 PCT/US2010/062651**

(87) Fecha y número de publicación internacional: **14.07.2011 WO11084875**

(96) Fecha de presentación y número de la solicitud europea: **31.12.2010 E 10842770 (9)**

(97) Fecha y número de publicación de la concesión europea: **25.04.2018 EP 2521967**

(54) Título: **Creación de símbolos inferidos a partir del uso de código**

(30) Prioridad:

06.01.2010 US 652758

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
27.06.2018

(73) Titular/es:

MICROSOFT TECHNOLOGY LICENSING, LLC (100.0%)
One Microsoft Way
Redmond, Washington 98052, US

(72) Inventor/es:

LIU, KAREN y
PILCH-BISSON, KEVIN

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Creación de símbolos inferidos a partir del uso de código

Antecedentes

Un lenguaje de programación dinámica realiza comportamientos en tiempo de ejecución que otros lenguajes (programación estática) suelen realizar, en todo caso, durante la compilación. Los comportamientos realizados en tiempo de ejecución por los lenguajes dinámicos incluyen extender el programa añadiendo un código nuevo, creando objetos y definiciones, modificando el sistema tipo durante la ejecución del programa, etc.

Un lenguaje de programación dinámica se tipa normalmente de manera dinámica, lo que significa que la comprobación de tipos se realiza en tiempo de ejecución. En el tipado dinámico, los tipos están asociados con los valores resultantes de la ejecución del programa. Los lenguajes de tipado dinámico incluyen Clojure, Groovy, JavaScript, Lisp, Objective-C, PHP, Prolog, Python, Ruby, Smalltalk, enlace tardío en Visual Basic, IronPython e IronRuby. El tipado dinámico es menos rígido que el tipado estático, pero puede dar lugar a un mayor potencial de errores de ejecución (por ejemplo, se produce un error debido a que un valor para una variable tiene un tipo que no está permitido). Los sistemas de lenguaje de tipado dinámico en general hacen menos comprobaciones en tiempo de compilación en el código fuente. Las comprobaciones en tiempo de ejecución potencialmente pueden ser más sofisticadas debido a que pueden usar información dinámica (tiempo de ejecución) además de la información que estaba presente durante la compilación y que aún está disponible en el tiempo de ejecución. Las comprobaciones de tiempo de ejecución garantizan que las condiciones se mantienen en una ejecución específica del programa y se repiten para cada ejecución del programa.

El enlace selecciona qué operación real usar cuando se aplica una operación sintáctica en el código fuente. El enlace puede ocurrir o en tiempo de compilación, en cuyo caso se denomina “enlace estático” o puede producirse dinámicamente en tiempo de ejecución, lo que se denomina “enlace dinámico”. El enlace dinámico pospone la resolución de variables indefinidas hasta que se ejecuta un programa. El enlace dinámico es diferente del enlace estático debido a su resultado, el significado asignado a una operación, por ejemplo, depende de los tipos de tiempo de ejecución de los valores reales en los que opera en lugar de los tipos de tiempo de compilación de las variables en el código fuente. Normalmente, los lenguajes de programación tipados de manera estática realizan un enlace estático y los lenguajes de programación tipados de manera dinámica realizan un enlace dinámico. Sin embargo, es posible tener un híbrido de los dos, donde un lenguaje estático contiene un enlace dinámico (tal como C# 4.0 con la característica de lenguaje dinámico) y viceversa.

El desarrollo guiado por pruebas se refiere a una técnica de desarrollo de software iterativo en el que un desarrollador escribe primero un caso de prueba que valida una mejora deseada o una nueva función, a continuación, el desarrollador produce el código que hace que se pase la prueba, seguido por la refactorización del código base para tener en cuenta los nuevos requisitos de sistema. Por ejemplo, el código que incluye la lógica que manipula los objetos puede escribirse antes de escribir el código que define los objetos. Otra forma de decir esto es: el código de consumir (consumo) se escribe antes de escribir los objetos que se consumen.

El documento US 2008/313604 A1 se refiere a un sistema y a un procedimiento para declarar variables durante la codificación de un programa de software. El procedimiento incluye, para cada tipo de variable, definir una cadena única que representa una instrucción de declaración de variable y adaptar un módulo de codificación en el que, cuando una cadena que representa una instrucción de declaración de variable se tipea adyacente a un nuevo nombre de variable, el módulo de codificación genera automáticamente código para la instrucción de declaración de variable correspondiente para una nueva variable que tiene el nuevo nombre de variable y el tipo de variable especificado.

El documento US 2006/026559 A1 se refiere a un procedimiento que se obtiene a partir de definiciones de metadatos de los valores permisibles para un argumento de procedimiento, en el que dichos valores permitidos son un subconjunto o un conjunto restringido de valores de un intervalo definido de valores para ese tipo de argumento de procedimiento. El subconjunto de valores permitidos se convierte a continuación en un fragmento de código fuente que puede usarse para inicializar la variable de argumento con uno de los valores permitidos cuando se invoca el procedimiento. El fragmento de código fuente se inserta en el código fuente para ese argumento de procedimiento usando una herramienta de edición. Un sistema para implementar el procedimiento puede comprender un programa de entorno de desarrollo integrado (IDE).

El documento US 2006/277525 A1 se refiere a un mecanismo para inferir una intención de un usuario informático en desarrollar el código de sistema informático. Específicamente, se proporciona un mecanismo de inferencia que puede inferir una intención de un usuario al menos en tres niveles: un nivel léxico, un nivel sintáctico y un nivel semántico. El mecanismo de inferencia emplea diversas técnicas basadas en reglas y basadas en heurística que incluyen la coerción de tipo, la proximidad de ámbito, el recuento de parámetros y argumentos, etc.

Steve Freeman y col: “Roles Mock, Not Objects”, Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, OOPSLA '04, 1 de enero de 2004, páginas 236-246 se refiere a los objetos ficticios, una extensión del desarrollo guiado por pruebas que soporta un buen diseño

orientado a objetos guiando el descubrimiento de un sistema coherente de tipos dentro de una base de código.

Sumario

El objeto de la presente invención es proporcionar un procedimiento y un sistema para crear símbolos inferidos a partir de un código fuente en el campo del desarrollo de programas interactivo.

Este objeto se resuelve por el objeto de las reivindicaciones independientes.

Las realizaciones preferidas están definidas por las reivindicaciones dependientes.

Las técnicas de programación dinámica y el desarrollo guiado por pruebas tienen al menos un desafío mutuo. En ambos casos, debido a que el código consumido subyacente aún no se ha creado, las herramientas que dependen de que el código subyacente esté disponible no pueden ayudar en el procedimiento de desarrollo, por ejemplo, mostrando posibles opciones de autocompletado a medida que el usuario escribe código en un editor de código fuente. De acuerdo con aspectos del objeto desvelado en el presente documento, se crean estructuras de datos tales como árboles de sintaxis o árboles semánticos que incluyen nodos inferidos y/o tablas de símbolos que incluyen símbolos inferidos, basándose en el uso de un símbolo (o uso potencial) en todo el código. Los nodos inferidos y/o los símbolos inferidos pueden generarse a través de un algoritmo de aprendizaje. Las herramientas pueden usar árboles de sintaxis, etc., creados de esta manera para proporcionar información sobre símbolos que aún no se han creado o que aún no están vinculados, para su uso en la programación dinámica y el desarrollo guiado por pruebas.

Una estructura de datos que representa el código fuente puede generarse por un compilador en segundo plano durante el desarrollo de programas interactivo. Se pueden añadir uno o más símbolos a la estructura de datos en función del código fuente introducido en un editor de código fuente, donde el símbolo se infiere a partir del código fuente en función del uso del símbolo sin definición previa del símbolo. El símbolo inferido puede crearse en función de la aplicación de un conjunto de reglas para un símbolo indefinido. En respuesta a recibir una indicación desde un usuario para hacerlo, un símbolo inferido puede convertirse en un símbolo real. La conversión de un símbolo inferido en un símbolo real puede desencadenarse mediante una acción del desarrollador para generar automáticamente el código fuente en función del símbolo inferido. El código fuente añadido automáticamente comprenderá una definición del símbolo indefinido. La información asociada con el símbolo inferido puede visualizarse en las herramientas de desarrollo de software en respuesta a la activación de una opción en el entorno de programación para introducir un “modo de sugerencia”.

Se proporciona este Sumario para introducir una selección de conceptos de una forma simplificada que se describen más adelante en la descripción detallada. Este Sumario no tiene la intención de identificar características clave o características esenciales del objeto reivindicado, ni está destinado a usarse para limitar el ámbito del objeto reivindicado.

Breve descripción de los dibujos

En los dibujos:

la figura 1a ilustra un ejemplo de un sistema 100 para crear símbolos inferidos a partir del uso de código de acuerdo con aspectos del objeto desvelado en el presente documento;
la figura 1b ilustra un ejemplo de un árbol 10 como se conoce en la técnica anterior;
la figura 1c ilustra un ejemplo de un árbol 12 de acuerdo con aspectos del objeto desvelado en el presente documento;
la figura 2 es un diagrama de flujo de un ejemplo de un procedimiento 200 para crear símbolos inferidos a partir del uso de código de acuerdo con aspectos del objeto desvelado en el presente documento;
la figura 3 es un diagrama de bloques que ilustra un ejemplo de un entorno informático en el que pueden implementarse aspectos del objeto desvelado en el presente documento; y
la figura 4 es un diagrama de bloques de un ejemplo de un entorno de desarrollo integrado de acuerdo con aspectos del objeto desvelado en el presente documento.

Descripción detallada

Visión de conjunto

Las prácticas de desarrollo guiado por pruebas están aumentando en popularidad. El lenguaje dinámico influye en que los lenguajes de programación sean cada vez más prominentes. Por lo tanto, sería útil si las herramientas de desarrollo de software pudieran proporcionar información relativa al código potencial cuando todavía no existe o aún no se conoce un modelo de código, pero aún se consume. El objeto descrito en el presente documento hace posible crear herramientas para estilos de programación de primer consumo que proporcionen información relevante antes de que los símbolos existan o antes de que se vinculen los símbolos.

El objeto desvelado en el presente documento se aplica en particular a la escritura de código con un estilo de desarrollo de primer consumo o de primera prueba y a la escritura de código dinámico (vinculado en tiempo de ejecución), aunque se apreciará que el objeto desvelado en el presente documento puede aplicarse a cualquier estilo de desarrollo de programas informáticos. Al escribir código, puede crearse un árbol de sintaxis, un árbol semántico o una tabla de símbolos que represente que está basado o está asociado con el código fuente. Para los símbolos que aún no están vinculados o definidos en la tabla de símbolos global, puede crearse un nodo o símbolo potencial o "inferido" y puede añadirse al árbol de sintaxis, al árbol semántico o a la tabla de símbolos creada por un compilador.

Pueden ser posibles múltiples candidatos para el tipo de símbolo para el símbolo inferido (por ejemplo, el símbolo puede representar una clase o puede representar una estructura, el símbolo puede representar un procedimiento o puede representar una propiedad). Todos los candidatos pueden representarse en el árbol/tabla como símbolos potenciales, o como alternativa, puede proporcionarse un servicio o módulo de software para recuperar el conjunto de candidatos. A medida que el usuario continúa escribiendo el código, el modelo de código puede actualizarse para representar la información adicional basada en un modelo de aprendizaje (por ejemplo, usando técnicas de aprendizaje automático, como se conoce en la técnica). Por ejemplo, si existe ambigüedad con respecto a lo que representa un determinado símbolo inferido, a medida que el desarrollador continúa escribiendo el código, puede resolverse la ambigüedad o puede ser más probable una elección específica entre una recopilación de opciones. En este caso, el símbolo inferido en el árbol de sintaxis, el árbol semántico o la tabla de símbolos puede actualizarse para representar el estado actual del conocimiento.

Los símbolos inferidos pueden usarse por las herramientas de desarrollo de software que incluyen pero no se limitan a: lista de finalización, jerarquía de llamadas, lista de referencias, navegadores de objetos, vistas de clase, búsqueda de referencias, herramientas de navegación, etc., para proporcionar ayuda de desarrollador para los símbolos que aún no se han creado o aún no se han vinculado. De acuerdo con algunos aspectos del objeto desvelado en el presente documento, los símbolos inferidos pueden distinguirse visualmente por una diferencia tal como el color, el resaltado, por ir acompañados por un icono específico, etc. Los símbolos inferidos pueden incluirse o hacerse visibles dentro de los entornos de desarrollo integrados tradicionales activando un modo específico (por ejemplo, activando una opción de "modo de sugerencia"). Basándose en el símbolo inferido, las condiciones de error descubiertas por el compilador pueden desencadenar la generación automática de código fuente adicional para corregir la condición de error. Los símbolos inferidos pueden convertirse en símbolos reales mediante, por ejemplo, la activación de una opción para "hacerlo real" o "generar a partir del uso". Por ejemplo, un nodo inferido puede convertirse en un nodo concreto en un árbol de sintaxis o árbol semántico mediante la activación de una opción de este tipo. De manera similar, un símbolo inferido en una tabla de símbolos puede convertirse en un símbolo concreto mediante la activación de una opción de este tipo para la tabla de símbolos.

Crear símbolos inferidos a partir del uso de código

La figura 1a ilustra un ejemplo de un sistema 100 para crear símbolos inferidos a partir del uso de código de acuerdo con aspectos del objeto desvelado en el presente documento. Todo o partes del sistema 100 pueden residir en uno o más ordenadores tales como los ordenadores descritos a continuación con respecto a la figura 3. Todo o partes del sistema 100 pueden residir en uno o más ordenadores de desarrollo de software (por ejemplo, el ordenador 102) tales como los ordenadores descritos a continuación con respecto a la figura 4. El sistema 100 o partes del mismo pueden comprender una parte de un entorno de desarrollo integrado (por ejemplo, IDE 104) tal como los descritos e ilustrados a continuación con respecto a la figura 4. Como alternativa, el sistema 100 o partes del mismo pueden proporcionarse como un sistema independiente o como un programa adicional o un complemento.

El sistema 100 puede incluir uno o más de: un procesador (tal como el procesador 142), una memoria tal como la memoria 144, y un módulo para crear símbolos inferidos a partir del uso 106 de código. También pueden incluirse otros componentes bien conocidos en la técnica pero no se muestra en este caso. Se apreciará que el módulo para crear símbolos inferidos a partir del uso 106 de código puede cargarse en la memoria 144 para hacer que uno o más procesadores, tal como el procesador 142, realicen las acciones atribuidas al módulo para crear los símbolos inferidos a partir del uso 106 de código.

Una estructura de datos creada o generada por un compilador en segundo plano basándose en el código fuente introducido puede modificarse por el módulo para crear símbolos inferidos a partir del uso 106 de código para incluir un símbolo inferido creado por el módulo para crear símbolos inferidos a partir del uso 106 de código. El símbolo inferido puede representar un código consumido en el que el código de consumo está presente en el código fuente cuando el código consumido no está presente en el código fuente. El módulo para crear símbolos inferidos a partir del uso 106 de código, de acuerdo con aspectos del objeto desvelado en el presente documento, puede crear uno o más símbolos 108 inferidos basándose en el código de consumo introducido en un editor 110.

El editor 110 puede representar un editor de código fuente asociado con un compilador 116 en segundo plano que genera un árbol de sintaxis, un árbol semántico o una tabla de símbolos de símbolos reales a medida que un usuario introduce el código fuente. El módulo para crear símbolos inferidos a partir del uso 106 de código, de acuerdo con aspectos del objeto desvelado en el presente documento, puede añadir o modificar uno o más símbolos 108 inferidos al árbol de sintaxis, al árbol semántico o a la tabla de símbolos de símbolos reales creados por el

compilador 116 basándose en el código introducido en un editor 110. Además, el módulo para crear símbolos inferidos a partir del uso 106 de código puede generar un código fuente basándose en el símbolo(s) inferido como se describe más completamente a continuación. Las herramientas 112 que operan sobre los símbolos 114 reales pueden ajustarse para operar además o en lugar de los símbolos 108 inferidos.

El módulo para crear símbolos inferidos a partir del uso 106 de código puede incluirse o incorporarse dentro de uno o más de los siguientes: un compilador tal como un compilador en segundo plano, un compilador paralelo o un compilador incremental, un analizador sintáctico tal como un analizador en segundo plano, un analizador sintáctico paralelo o un analizador sintáctico incremental o un programa adicional, un preprocesador o un complemento o una extensión a un IDE, un analizador sintáctico, un compilador o un preprocesador. El módulo para crear símbolos inferidos a partir del uso 106 de código puede estar separado de o estar asociado con un compilador tal como un compilador en segundo plano, un compilador paralelo o un compilador incremental, un analizador sintáctico tal como un analizador sintáctico en segundo plano, un analizador sintáctico paralelo o un analizador sintáctico incremental o un programa adicional, un pre-procesador, o un complemento o una extensión para un IDE, un analizador sintáctico, un compilador o un preprocesador. El módulo para crear símbolos inferidos a partir del uso 106 de código puede estar asociado con o incorporado dentro o separado de unas herramientas de desarrollo de programas incluidas, pero no limitadas a, una lista de finalización, una jerarquía de llamadas, una lista de referencias, unos navegadores de objeto, unas vistas de clase, unas referencias de búsqueda y/o unas herramientas de navegación. El módulo para crear símbolos inferidos a partir del uso 106 de código puede convertir uno o más símbolos inferidos en símbolos reales en una tabla de símbolos, un árbol de sintaxis o un árbol semántico.

Un editor 110 puede ser un editor de código fuente respaldado por un compilador en segundo plano que genera árboles semánticos y/o de sintaxis mientras que el desarrollador o usuario está escribiendo el código fuente. Supóngase, por ejemplo, que un usuario escribe lo siguiente:

```
public class Engine {
}
public class Bar {
  public void Foo() {
    Engine e = new Engine();
    e.Start();
} }
```

Un compilador en segundo plano como se conoce en la técnica puede generar un árbol 10 de sintaxis como se ilustra en la figura 1b a medida que el desarrollador introduce el código en el editor de código fuente (por ejemplo, el editor 110 de la figura 1a). El árbol 10 de sintaxis es un ejemplo de una estructura de datos de símbolos reales como se muestra en la figura 1a, los símbolos 114 reales. El árbol de sintaxis creado por el compilador en segundo plano incluye el nodo para la clase Bar y el nodo para la clase Engine en respuesta a las declaraciones de estas clases por las líneas:

```
public class Bar {...}
and
public class Engine {
}
respectivamente.
```

De acuerdo con aspectos del objeto desvelado en el presente documento, el módulo para crear símbolos inferidos a partir del uso 106 de código crea el nodo inferido para el procedimiento de “Start”. Cuando el módulo para crear símbolos inferidos del uso 106 de código detecta la declaración:

```
e.Start ();
```

el módulo para crear símbolos inferidos a partir del uso 106 de código deduce que se necesita una declaración para el inicio del procedimiento. Haciendo referencia ahora a la figura 1c, el módulo para crear símbolos inferidos a partir del uso 106 de código crea un nodo inferido, el nodo 14 para el inicio del procedimiento en el árbol 12. La naturaleza inferida del nodo se representa en la figura 1c por las líneas discontinuas del nodo 14. Una vez que el nodo inferido se ha añadido al árbol de sintaxis, el nodo inferido puede usarse por las herramientas 112 que incluyen, pero no se limitan a una lista de finalización, una jerarquía de llamadas, una lista de referencias, unos navegadores de objetos, unas vistas de clase, una búsqueda de referencias y/o unas herramientas de navegación.

Por ejemplo, una vez que el nodo inferido se ha añadido al árbol de sintaxis, si el desarrollador fuera a escribir una “e.” en su código, una de las opciones que se le presenta mediante una herramienta de finalización automática podría ser “Start.”. Además, una vez que el nodo inferido se ha añadido al árbol de sintaxis, el nodo inferido puede convertirse en una parte permanente del árbol de sintaxis activando una opción de “hacerlo real” o “generar a partir del uso” disponible en las herramientas tales como, pero no limitadas a las herramientas que asocian información y acciones con la información tipeada (por ejemplo, etiquetas de acción). En respuesta a realizar un nodo inferido real, puede generarse el código fuente de acuerdo con un conjunto de heurísticas. Por ejemplo, en respuesta a realizar el procedimiento de nodo Start real, puede generarse el siguiente código fuente:

```
public void Start() {
}
```

Puede proporcionarse una opción de activación en el entorno de programación que permita a los nodos inferidos hacerse visibles o referenciarse en las herramientas disponibles.

El símbolo inferido creado por el módulo para crear símbolos inferidos a partir del uso 106 de código puede crearse basándose en la aplicación de un conjunto de reglas, como se describe más completamente a continuación con respecto a la figura 2.

La figura 2 ilustra un ejemplo de un procedimiento 200 para construir árboles de sintaxis inferidos a partir del uso de código de acuerdo con aspectos del objeto desvelado en el presente documento. En 202, el módulo para crear símbolos inferidos a partir del uso de código monitoriza el código introducido en un editor de código fuente que está respaldado por un compilador en segundo plano. En respuesta a la detección del código de consumo para el que se ha perdido el código a consumir, el módulo para crear símbolos inferidos a partir del uso de código aplica heurísticas proporcionadas en 204 para crear uno o más símbolos inferidos en 206. Los símbolos inferidos creados incluyen símbolos inferidos añadidos a una tabla de símbolos, nodos inferidos añadidos a un árbol de sintaxis o nodos inferidos añadidos a un árbol semántico.

La heurística puede incluir un conjunto de valores predeterminados. Por ejemplo, en un caso en el que un símbolo podría representar una clase, una interfaz o una estructura es parte de la misma, una regla o reglas pueden proporcionar un mecanismo para decidir qué tipo de símbolo representa el símbolo inferido. Para un símbolo que representa un procedimiento que tiene diferentes conjuntos de parámetros, una regla puede determinar si los parámetros para los procedimientos deben unificarse en un pequeño conjunto de sobrecargas de un solo tipo o en un conjunto más amplio de sobrecargas de múltiples tipos o en cualquier punto intermedio. Por ejemplo, si, usando el ejemplo anterior, se pasa un parámetro de 3 al procedimiento e.Start, puede determinarse que el procedimiento Start toma un parámetro entero. Si en otro lugar del código fuente se pasa una cadena a e.Start, puede determinarse que hay dos procedimientos llamados Start, uno que toma cadenas y otro que toma enteros. Puede configurarse una heurística para determinar qué símbolos inferidos se generan de manera predeterminada con un acceso mínimo (por ejemplo, “privado”) o con acceso público. Con respecto a determinar si un símbolo inferido es una clase o una interfaz, puede determinarse por el lenguaje circundante que un símbolo inferido determinado es una interfaz o una clase. Por ejemplo, la instrucción:

```
Engine e = new Engine ();
```

identifica Engine como una clase en lugar de una interfaz debido a que esta instrucción crea un nuevo constructor para el motor de clase y se sabe que las interfaces no pueden construirse directamente.

Una vez que se ha añadido el símbolo inferido a la estructura de datos (por ejemplo, árbol de sintaxis, árbol semántico o tabla de símbolos,) hay varias opciones disponibles para el usuario. Por ejemplo, la selección de una opción (por ejemplo, una opción “hacerlo real” o “generar a partir del uso”) puede desencadenar la inserción automática generada por compilador del código fuente en el programa fuente en 208 y la conversión del símbolo inferido en un símbolo real en 212. El código generado por compilador puede representar un código consumido, que incluye, por ejemplo, una definición de un símbolo anteriormente indefinido (por ejemplo, la generación de una declaración de procedimiento, declaración de clase, etc., tal como interfaces, eventos y estructuras). La selección de otra opción puede hacer que las herramientas de desarrollo de software muestren símbolos inferidos en 210 en las herramientas de desarrollo de software tales como, pero no limitadas a, una lista de finalización, una jerarquía de llamadas, una lista de referencias, unos navegadores de objetos, unas vistas de clases, una búsqueda de referencias y/o unas herramientas de navegación. El código generado por compilador puede visualizarse como código de “vista previa”. En tal caso, los símbolos concretos pueden crearse temporalmente. En respuesta a una acción del usuario que confirma la acción, los símbolos concretos temporales pueden volverse permanentes. Como alternativa, en respuesta a una acción de usuario que abandona el código, los símbolos concretos temporales pueden descartarse. Se apreciará que cuando se han usado anteriormente el término “compilador” y la expresión “generado por compilador”, el módulo para crear símbolos inferidos a partir del uso de código puede contribuir al código generado por compilador e incorporarse o invocarse por el compilador, como se ha descrito anteriormente con respecto a la figura 1a, etc.

Ejemplo de un entorno informático adecuado

Con el fin de proporcionar un contexto para diversos aspectos del objeto desvelado en el presente documento, la figura 3 y la siguiente exposición pretenden proporcionar una breve descripción general de un entorno 510 informático adecuado en el que pueden implementarse varias realizaciones. Si bien el objeto desvelado en el presente documento se describe en el contexto general de instrucciones ejecutables por ordenador, tales como módulos de programa, ejecutados por uno o más ordenadores u otros dispositivos informáticos, los expertos en la materia reconocerán que las partes del objeto desvelado en el presente documento pueden también implementarse en combinación con otros módulos de programa y/o una combinación de hardware y software. En general, los módulos de programa incluyen rutinas, programas, objetos, artefactos físicos, estructuras de datos, etc., que realizan

tareas específicas o implementan tipos de datos específicos. Normalmente, la funcionalidad de los módulos de programa puede combinarse o distribuirse como se desee en diversas realizaciones. El entorno 510 informático es solo un ejemplo de un entorno operativo adecuado y no pretende limitar el ámbito de uso o la funcionalidad del objeto desvelado en el presente documento.

Haciendo referencia a la figura 3, se describe un dispositivo informático para crear símbolos inferidos a partir del uso de código en la forma de un ordenador 512. El ordenador 512 puede incluir una unidad 514 de procesamiento, una memoria 516 de sistema y un bus 518 de sistema. La unidad 514 de procesamiento puede ser cualquiera de los diversos procesadores disponibles. También pueden emplearse microprocesadores duales y otras arquitecturas de multiprocesador como la unidad 514 de procesamiento. La memoria 516 de sistema puede incluir memoria 520 volátil y memoria 522 no volátil. La memoria 522 no volátil puede incluir memoria de solo lectura (ROM), ROM programable (PROM), ROM eléctricamente programable (EPROM) o memoria flash. La memoria 520 volátil puede incluir memoria de acceso aleatorio (RAM) que puede actuar como una memoria caché externa. El bus 518 de sistema acopla los artefactos físicos de sistema incluyendo la memoria 516 de sistema a la unidad 514 de procesamiento. El bus 518 de sistema puede ser de diversos tipos incluyendo un bus de memoria, un controlador de memoria, un bus periférico, un bus externo o un bus local y puede usar cualquier variedad de arquitecturas de bus disponibles.

El ordenador 512 incluye normalmente una variedad de medios legibles por ordenador tales como unos medios volátiles y no volátiles, medios extraíbles y no extraíbles. Los medios de almacenamiento informático pueden implementarse en cualquier procedimiento o tecnología para el almacenamiento de información tal como instrucciones legibles por ordenador, estructuras de datos, módulos de programa u otros datos. Los medios de almacenamiento informático incluyen, entre otros, RAM, ROM, EEPROM, memoria flash u otra tecnología de memoria, CDROM, discos versátiles digitales (DVD) u otro tipo de almacenamiento en disco óptico, casetes magnéticos, cinta magnética, almacenamiento en disco magnético u otros dispositivos de almacenamiento magnéticos o cualquier otro medio no transitorio que pueda usarse para almacenar la información deseada y a la que se pueda acceder por el ordenador 512.

Se apreciará que la figura 3 describe un software que puede actuar como intermediario entre los usuarios y los recursos informáticos. Este software puede incluir un sistema 528 operativo que puede almacenarse en el almacenamiento 524 en disco y que puede controlar y asignar recursos del sistema 512 informático. El almacenamiento 524 en disco puede ser una unidad de disco duro conectada al bus 518 de sistema a través de una interfaz de memoria no extraíble tal como la interfaz 526. Las aplicaciones 530 de sistema aprovechan la gestión de recursos mediante el sistema 528 operativo a través de los módulos 532 de programa y los datos 534 de programa almacenados o en la memoria 516 de sistema o en el almacenamiento 524 en disco. Se apreciará que los ordenadores pueden implementarse con diversos sistemas operativos o combinaciones de sistemas operativos.

Un usuario puede introducir comandos o información en el ordenador 512 a través de un dispositivo(s) 536 de entrada. Los dispositivos 536 de entrada incluyen, pero no están limitados a un dispositivo de puntero tal como un ratón, bola de seguimiento, lápiz óptico, pantalla táctil, teclado, micrófono, y similares. Estos y otros dispositivos de entrada se conectan a la unidad 514 de procesamiento a través del bus 518 de sistema a través de un puerto(s) 538 de interfaz. Un puerto(s) 538 de interfaz puede representar un puerto serie, un puerto paralelo, un bus serie universal (USB) y similares. Los dispositivos 540 de salida pueden usar el mismo tipo de puertos que los dispositivos de entrada. El adaptador 542 de salida se proporciona para ilustrar que hay algunos dispositivos 540 de salida como los monitores, los altavoces y las impresoras que requieren adaptadores específicos. Los adaptadores 542 de salida incluyen, pero no están limitados a, tarjetas de video y sonido que proporcionan una conexión entre el dispositivo 540 de salida y el bus 518 de sistema. Otros dispositivos y/o sistemas o dispositivos tales como los ordenadores 544 remotos pueden proporcionar tanto capacidades de entrada como de salida.

El ordenador 512 puede operar en un entorno de red usando conexiones lógicas a uno o más ordenadores remotos, tales como un ordenador(s) 544 remoto. El ordenador 544 remoto puede ser un ordenador personal, un servidor, un encaminador, un PC de red, un dispositivo del mismo nivel u otro nodo de red común, e incluye normalmente muchos o todos los elementos descritos anteriormente con respecto al ordenador 512, aunque en la figura 4 se ha ilustrado solamente un dispositivo 546 de almacenamiento de memoria. Los ordenadores 544 remotos pueden conectarse lógicamente a través de una conexión 550 de comunicación. La interfaz 548 de red abarca redes de comunicación tales como las redes de área local (LAN) y las redes de área extensa (WAN) pero también puede incluir otras redes. La conexión(es) 550 de comunicación se refiere al hardware/software empleado para conectar la interfaz 548 de red al bus 518. La conexión 550 puede ser interna o externa al ordenador 512 e incluir tecnologías internas y externas tales como módems (teléfono, cable, DSL e inalámbrico) y adaptadores RDSI, tarjetas Ethernet, etc.

Se apreciará que las conexiones de red mostradas son solo ejemplos y que pueden usarse otros medios de establecer un enlace de comunicaciones entre los ordenadores. Un experto en la materia puede apreciar que un ordenador 512 u otro dispositivo cliente, puede implementarse como parte de una red informática. A este respecto, el objeto desvelado en la presente memoria pertenece a cualquier sistema informático que tenga cualquier cantidad de unidades de memoria o almacenamiento, y cualquier cantidad de aplicaciones y procedimientos que se produzcan en cualquier cantidad de unidades o volúmenes de almacenamiento. Los aspectos del objeto desvelado

en el presente documento pueden aplicarse a un entorno con ordenadores servidor y ordenadores cliente desplegados en un entorno de red, que tengan almacenamiento remoto o local. Los aspectos del objeto desvelado en el presente documento también pueden aplicarse a un dispositivo informático independiente, que tenga funcionalidades de lenguaje de programación, interpretación y ejecución.

La figura 4 ilustra un entorno 600 de desarrollo integrado (IDE) y un entorno 602 de tiempo de ejecución de lenguaje común. Un IDE 600 puede permitir a un usuario (por ejemplo, desarrollador, programador, diseñador, codificador, etc.) diseñar, codificar, compilar, probar, ejecutar, editar, depurar o crear un programa, conjunto de programas, sitios web, aplicaciones web y servicios web en un sistema informático. Los programas de software pueden incluir código fuente (componente 610), creado en uno o más lenguajes de código fuente (por ejemplo, Visual Basic, Visual J#, C++. C#, J#, Java Script, APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl, Python, Scheme, Smalltalk y similares). El IDE 600 puede proporcionar un entorno de desarrollo de código nativo o puede proporcionar un desarrollo de código gestionado que se ejecuta en una máquina virtual o puede proporcionar una combinación de los mismos. El IDE 600 puede proporcionar un entorno de desarrollo de código gestionado usando .NET framework.

Un componente 650 de lenguaje intermedio puede crearse a partir del componente 610 de código fuente y el componente 611 de código nativo usando un compilador 620 fuente de lenguaje específico y el componente 611 de código nativo (por ejemplo, instrucciones ejecutables por máquina) se crea a partir del componente 650 de lenguaje intermedio que usa el compilador 660 de lenguaje intermedio (por ejemplo, un compilador de justo a tiempo (JIT)), cuando se ejecuta la aplicación. Es decir, cuando se ejecuta una aplicación IL, se compila mientras se ejecuta en el lenguaje de máquina apropiado para la plataforma en la que se está ejecutando, haciendo de este modo que el código sea portable en varias plataformas. Como alternativa, en otras realizaciones, los programas pueden compilarse al lenguaje de máquina de código nativo (no mostrado) apropiado para su plataforma deseada.

Un usuario puede crear y/o editar el componente de código fuente de acuerdo con las técnicas de programación de software conocidas y las reglas lógicas y sintácticas específicas asociadas con un lenguaje fuente específico a través de una interfaz 640 de usuario y un editor 651 de código fuente en el IDE 600. A continuación, el componente 610 de código fuente puede compilarse a través de un compilador 620 de fuente, por lo que puede crearse una representación de lenguaje intermedio del programa, tal como el conjunto 630. El conjunto 630 puede comprender el componente 650 de lenguaje intermedio y los metadatos 642.
Los diseños de aplicaciones pueden validarse antes del desarrollo.

Las diversas técnicas descritas en el presente documento pueden implementarse junto con hardware o software o, en su caso, con una combinación de ambos. Por lo tanto, los procedimientos y aparatos descritos en el presente documento, o ciertos aspectos o partes de los mismos, pueden tomar la forma de código de programa (es decir, instrucciones) incorporado en medios tangibles, tales como disquetes, CD-ROM, discos duros o cualquier otro medio de almacenamiento legible por máquina, en el que, cuando el código de programa se carga y se ejecuta en una máquina, tal como un ordenador, la máquina se convierte en un aparato para practicar aspectos del objeto divulgado en el presente documento. En el caso de la ejecución de código de programa en ordenadores programables, el dispositivo informático incluirá en general un procesador, un medio de almacenamiento legible por el procesador (que incluye memoria volátil y no volátil y/o elementos de almacenamiento), al menos un dispositivo de entrada, y en al menos un dispositivo de salida. Uno o más programas que pueden usar la creación y/o la implementación de aspectos de modelos de programación específicos del dominio, por ejemplo, a través del uso de una API de procesamiento de datos o similar, pueden implementarse en un lenguaje de programación orientado a objetos o procedimientos de alto nivel para comunicarse con un sistema informático. Sin embargo, el programa(s) puede implementarse en ensamblador o lenguaje máquina, si se desea. En cualquier caso, el lenguaje puede ser un lenguaje compilado o interpretado, y combinado con implementaciones de hardware.

Mientras que el objeto desvelado en el presente documento se ha descrito en relación con las figuras, se ha de entender que pueden hacerse modificaciones para realizar las mismas funciones de diferentes maneras.

REIVINDICACIONES

1. Un sistema para el desarrollo de programas interactivo, que comprende:

un editor (110) de código fuente adaptado para recibir el código fuente introducido por un usuario;
un compilador (116) en segundo plano adaptado para generar una estructura de datos que representa el código fuente mientras el usuario introduce el código fuente en el editor de código fuente, siendo la estructura de datos una de entre un árbol de sintaxis, un árbol semántico y una tabla de símbolos; y
un módulo (106) adaptado para monitorizar el código fuente introducido, para detectar al menos un símbolo no declarado, para deducir que se necesita una declaración para el símbolo no declarado, para crear un símbolo inferido para cada símbolo no declarado detectado, y para añadir el al menos un símbolo inferido a la estructura de datos, en el que el al menos un símbolo inferido representa uno de entre una clase, una estructura, un procedimiento y una propiedad.

2. El sistema de la reivindicación 1, en el que el al menos un símbolo inferido se crea en un entorno de desarrollo guiado por pruebas o en un entorno de programación dinámica.

3. El sistema de acuerdo con la reivindicación 1, en el que el al menos un símbolo inferido se convierte en un símbolo (114) real en la estructura de datos, en el que un símbolo real es un símbolo inferido que se convierte en una parte permanente de la estructura de datos tras la activación de un opción por el usuario.

4. El sistema de la reivindicación 1, en el que la conversión del al menos un símbolo inferido en un símbolo real desencadena la adición automática generada por compilador del código fuente que define el al menos un símbolo inferido, en el que el código fuente que define el al menos un símbolo inferido se añade al código fuente en el editor de código fuente.

5. El sistema de la reivindicación 1, en el que la información de ayuda de desarrollador asociada con el al menos un símbolo inferido se visualiza durante el desarrollo del programa en respuesta a la selección de usuario de una opción para visualizar la información asociada con el al menos un símbolo inferido.

6. Un procedimiento implementado por ordenador para el desarrollo de programas interactivo, que comprende:

recibir un código fuente introducido por un usuario en un editor (110) de código fuente;
generar una estructura de datos por un compilador en segundo plano que representa el código fuente mientras el usuario introduce el código fuente en el editor de código fuente, siendo la estructura de datos una de entre un árbol de sintaxis, un árbol semántico y una tabla de símbolos;
monitorizar (202) el código fuente introducido;
detectar al menos un símbolo no declarado;
deducir que se necesita una declaración para el símbolo no declarado;
crear (206) un símbolo inferido para cada símbolo no declarado detectado; y
añadir el al menos un símbolo inferido a la estructura de datos,
en el que el al menos un símbolo inferido representa uno de entre una clase, una estructura, un procedimiento y una propiedad.

7. El procedimiento de la reivindicación 6, en el que el al menos un símbolo inferido se crea en un entorno de desarrollo guiado por pruebas o en un entorno de programación dinámica.

8. El procedimiento de la reivindicación 6, que comprende además:

en respuesta a la activación de una opción, visualizar la información asociada con el al menos un símbolo inferido.

9. El procedimiento de la reivindicación 6,
en el que el código generado por compilador se añade al código fuente que define el al menos un símbolo no declarado.

10. El procedimiento de la reivindicación 6, que comprende además:

convertir el al menos un símbolo inferido en un símbolo real en respuesta a recibir una indicación desde un usuario para convertir el al menos un símbolo inferido en un símbolo real, en el que el al menos un símbolo inferido se convierte en un símbolo (114) real en la estructura de datos, en el que un símbolo real es un símbolo inferido que se convierte en una parte permanente de la estructura de datos tras la activación de una opción por el usuario.

11. Un medio de almacenamiento legible por ordenador que comprende unas instrucciones ejecutables por ordenador que, cuando se ejecutan, hacen que al menos un procesador realice el procedimiento de una de las reivindicaciones 6 a 10.

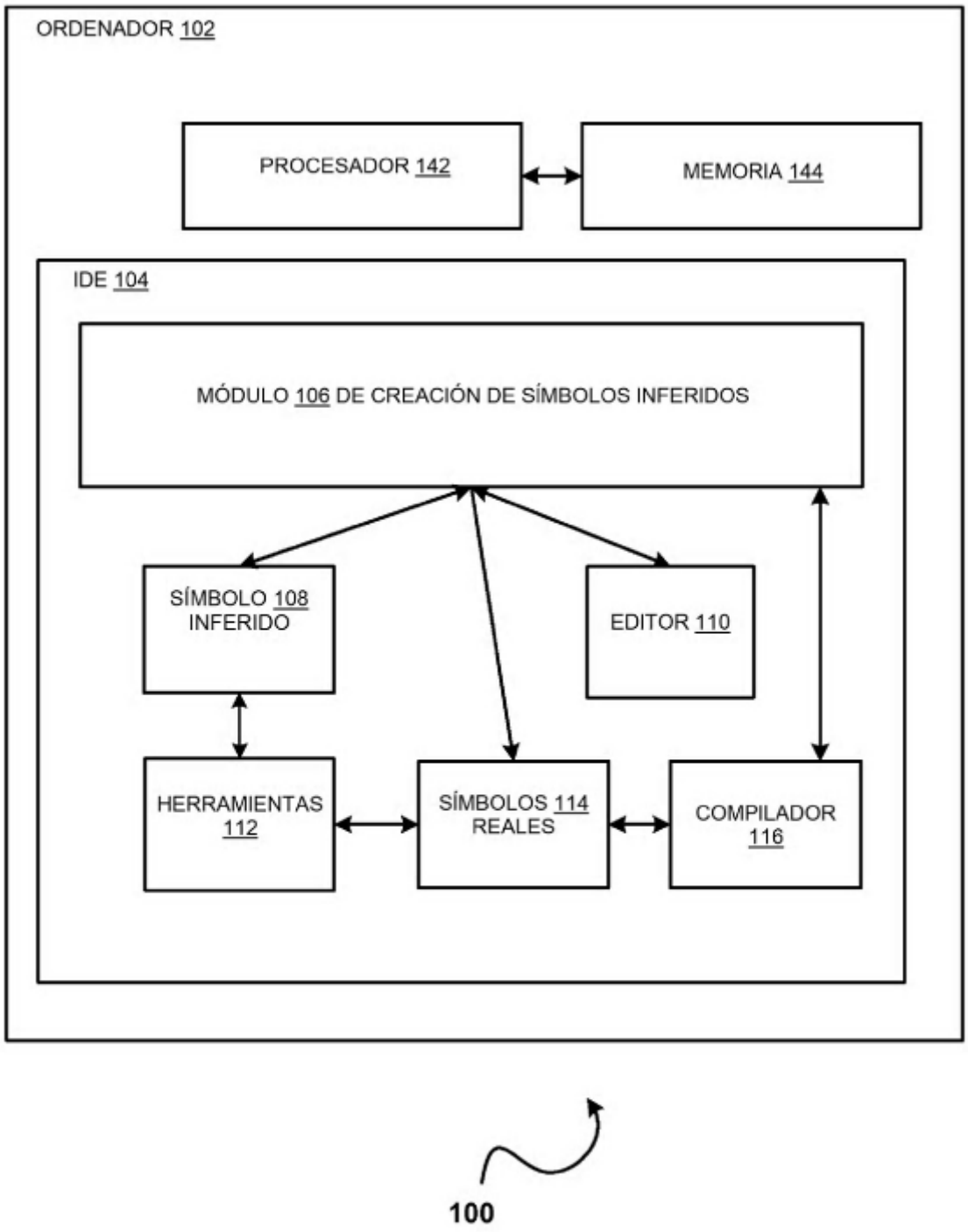
ORDENADOR 102
PROCESADOR 142
MEMORIA 144
IDE 104
MÓDULO 106 DE CREACIÓN DE SÍMBOLOS INFERIDOS
SÍMBOLO 108 INFERIDO
EDITOR 110
HERRAMIENTAS 112
SÍMBOLOS 114 REALES
COMPILADOR 116
100

FIG. 1a

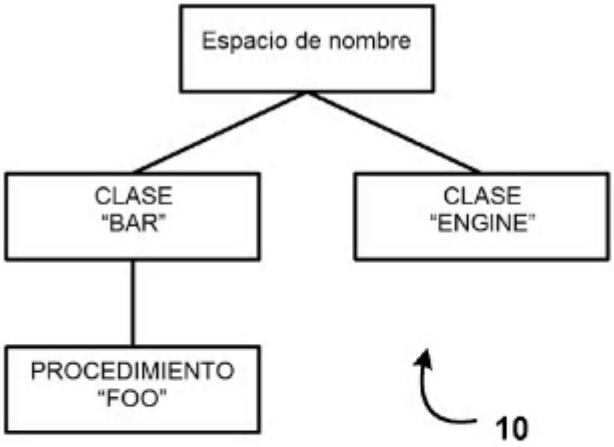
Espacio de nombre
CLASE
"BAR"
CLASE
"ENGINE"
PROCEDIMIENTO
"FOO"
10

FIG. 1b

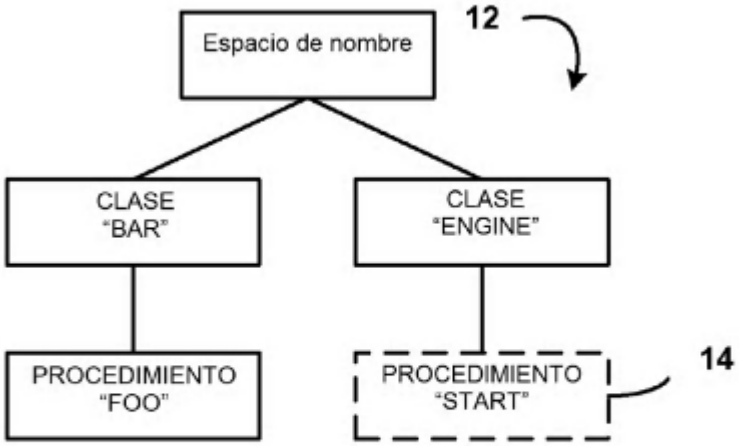
Espacio de nombre
12
CLASE
"BAR"
CLASE
"ENGINE"
PROCEDIMIENTO
"FOO"
PROCEDIMIENTO
"START"
14

FIG. 1c

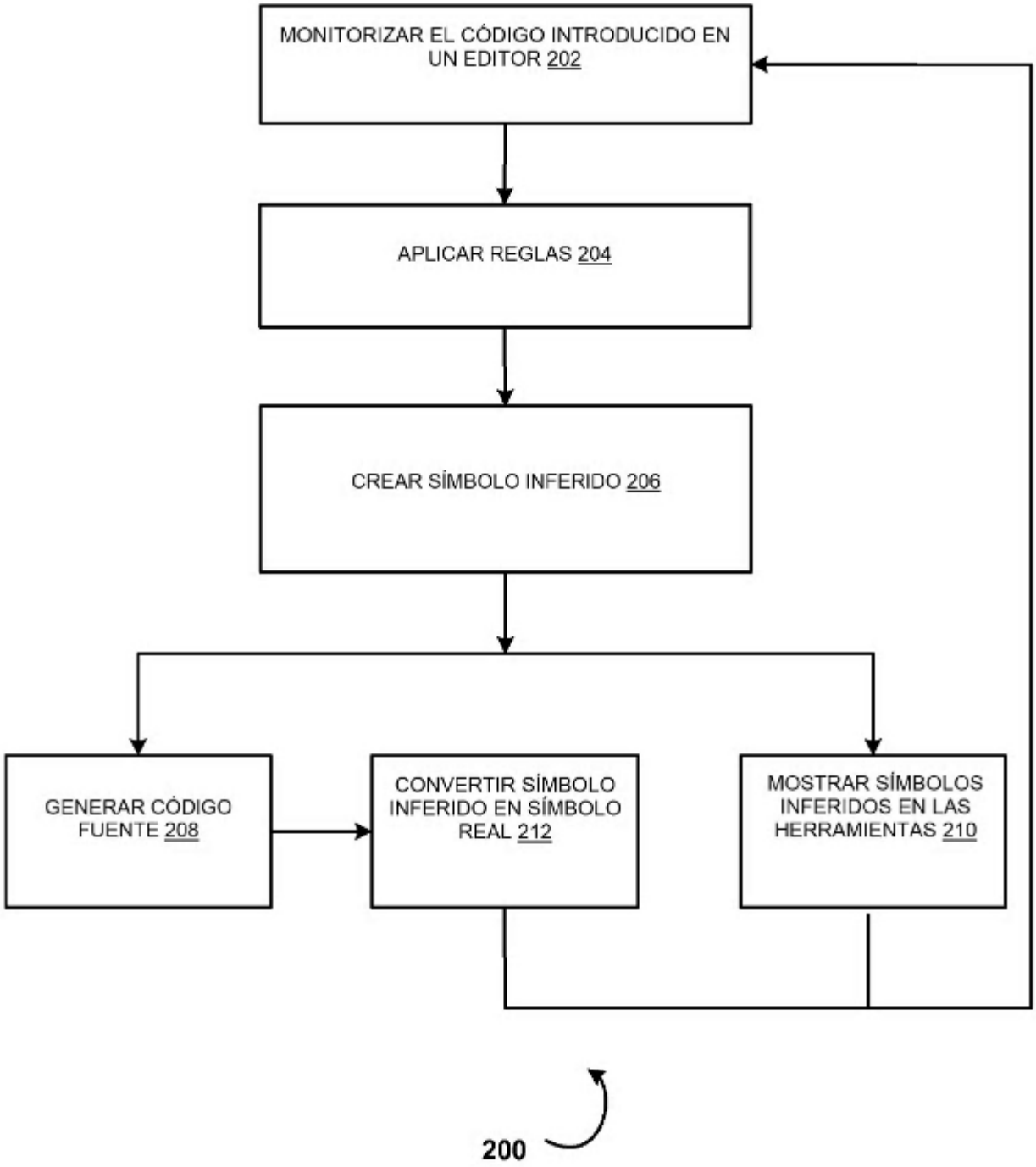
MONITORIZAR EL CÓDIGO INTRODUCIDO EN UN EDITOR 202
APLICAR REGLAS 204
CREAR SÍMBOLO INFERIDO 206
GENERAR CÓDIGO FUENTE 208
CONVERTIR SÍMBOLO INFERIDO EN SÍMBOLO REAL 212
MOSTRAR SÍMBOLOS INFERIDOS EN LAS HERRAMIENTAS 210
200

FIG. 2

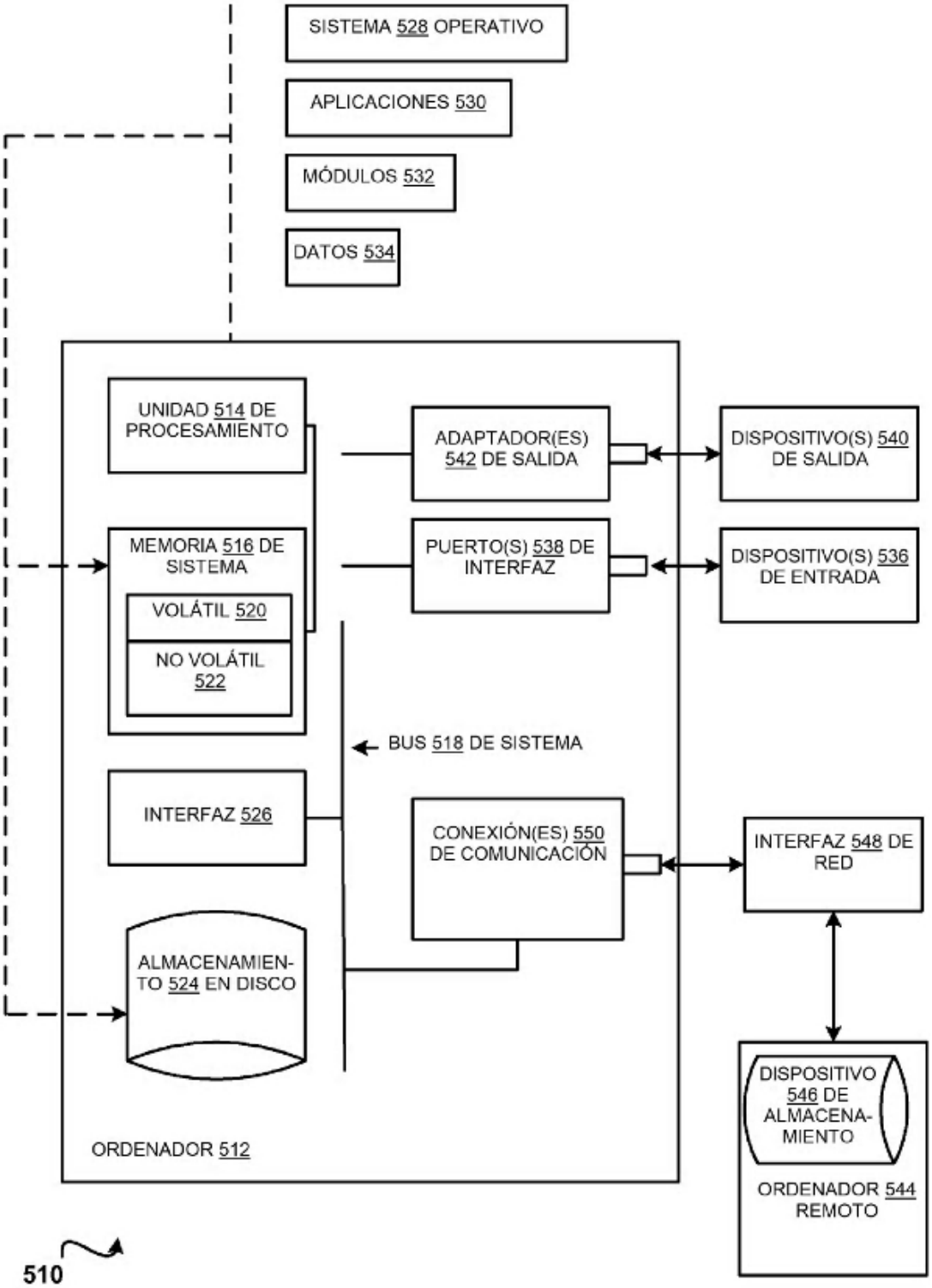
SISTEMA 528 OPERATIVO
APLICACIONES 530
MÓDULOS 532
DATOS 534
UNIDAD 514 DE PROCESAMIENTO
ADAPTADOR(ES) 542 DE SALIDA
DISPOSITIVO(S) 540 DE SALIDA
MEMORIA 516 DE SISTEMA
VOLÁTIL 520
NO VOLÁTIL 522
PUERTO(S) 538 DE INTERFAZ
DISPOSITIVO(S) 536 DE ENTRADA
BUS 518 DE SISTEMA
INTERFAZ 526
CONEXIÓN(ES) 550 DE COMUNICACIÓN
INTERFAZ 548 DE RED
ALMACENAMIENTO 524 EN DISCO
DISPOSITIVO 546 DE ALMACENAMIENTO
ORDENADOR 544 REMOTO
ORDENADOR 512
510

FIG. 3

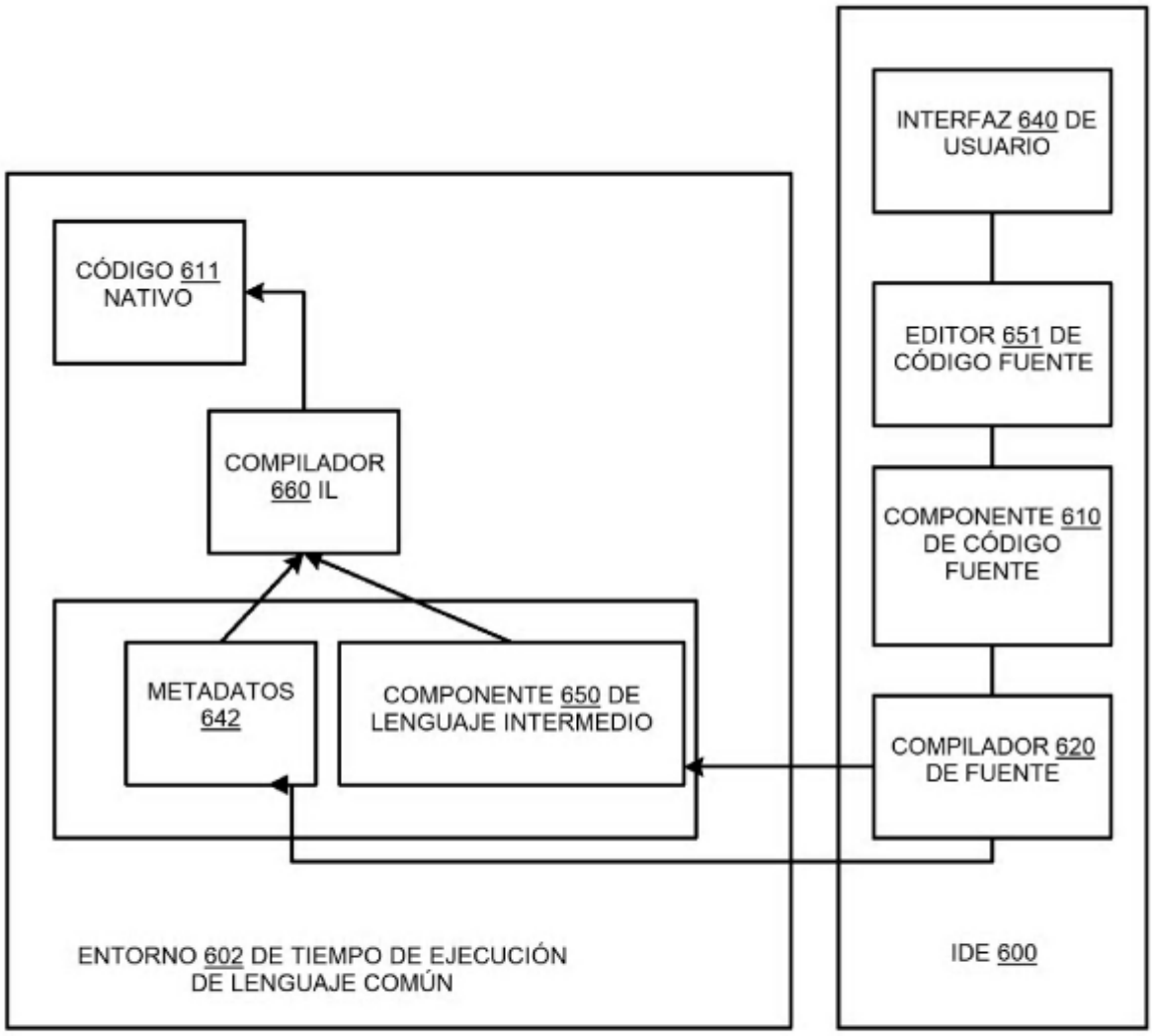
INTERFAZ 640 DE USUARIO
EDITOR 651 DE CÓDIGO FUENTE
COMPONENTE 610 DE CÓDIGO FUENTE
COMPILADOR 620 DE FUENTE
IDE 600
CÓDIGO 611 NATIVO
COMPILADOR 660 IL
METADATOS 642
COMPONENTE 650 DE LENGUAJE INTERMEDIO
ENTORNO 602 DE TIEMPO DE EJECUCIÓN DE LENGUAJE COMÚN

FIG. 4