

OPTIMIZADOR DE CONSULTAS PARA UTILIZACIÓN DE CPU Y REFACTORIZACIÓN DE CÓDIGO

Antecedentes de la invención

Existen diversos tipos de aplicaciones de desarrollo de *software* que los desarrolladores de *software* pueden utilizar para desarrollar *software*. Un entorno de desarrollo integrado (IDE, por sus siglas en inglés) es un tipo de aplicación de desarrollo de *software* que contiene varias herramientas de desarrollo en un paquete. Un IDE puede incluir herramientas como un editor de código fuente ("editor de código"), una herramienta de automatización de compilación y un depurador. Los ejemplos de IDE incluyen Eclipse™ desarrollado por Eclipse Foundation de Ottawa, Canadá, ActiveState Komodo™ desarrollado por ActiveState de Vancouver, Canadá, IntelliJ IDEA desarrollado por JetBrains de la República Checa, Oracle JDeveloper™ desarrollado por Oracle Corporation de Redwood City, California, NetBeans desarrollado por Oracle Corporation, Codenvy™ desarrollado por Codenvy de San Francisco, California, Xcode® desarrollado por Apple Corporation de Cupertino, California, y Microsoft® Visual Studio®, desarrollado por Microsoft Corporation de Redmond, Washington.

Muchos lenguajes de programación modernos son compatibles de manera nativa con las consultas de datos. Por ejemplo, Microsoft .NET Framework, desarrollado por Microsoft Corporation, es compatible con las consultas en forma de LINQ (consulta integrada en el lenguaje), mientras que Java®, desarrollado por Oracle Corporation o Redwood City, California, es compatible con las consultas en forma de Streams. La compatibilidad nativa con las consultas en un lenguaje de programación permite a los desarrolladores concentrarse en la parte lógica de su código de programa, debido a que la funcionalidad de consulta integrada se encarga de la implementación real de las consultas para ellos. Esto puede permitir a un desarrollador acelerar su codificación.

Sin embargo, no saber o tener una mala comprensión de las consecuencias de las consultas ineficientes puede conducir a que se desarrolle un código de programa ineficiente. Además, debido a las mejores prácticas actuales para el desarrollo de *software*, los desarrolladores tienden a dividir métodos/procedimientos más grandes en métodos/procedimientos más pequeños en su código para una mayor legibilidad, lo cual también puede conducir a ineficiencias en las consultas en el código de programa.

Breve descripción de la invención

Esta breve descripción se proporciona para introducir una selección de conceptos en una forma simplificada que se describe con más detalle posteriormente en la descripción detallada. Esta breve descripción no tiene la intención de identificar las características importantes o las características esenciales de la materia objeto reivindicada, ni pretende utilizarse para limitar el alcance de la materia objeto reivindicada.

Se proporcionan métodos, sistemas, aparatos y productos de programa informático para aumentar la eficacia de las consultas en el código de programa. Se detecta una pluralidad de consultas en el código de programa, como cuando el código de programa se ingresa en un editor de código o

durante el tiempo de compilación. Se extiende una evaluación perezosa mediante la cual las consultas se evalúan en el código de programa. Además, un conjunto de reglas que incluye una pluralidad de reglas se aplica a las consultas detectadas para generar un conjunto de consultas funcionalmente equivalentes que se evalúa de manera más eficiente con relación a las consultas detectadas.

5 Las reglas incluidas en el conjunto de reglas pueden generarse de cualquier manera, incluso generarse manualmente por parte de un usuario o generarse automáticamente por un algoritmo, como aprendizaje automático, análisis de macrodatos de ejemplos de código, etcétera. Un conjunto de tales reglas puede mejorar significativamente la mayoría de los errores comunes que ocurren en el trabajo diario de los desarrolladores.

10 Puede generarse un único conjunto de consultas funcionalmente equivalentes e introducirse en el código de programa en lugar de una o más de las consultas existentes, o pueden generarse múltiples conjuntos de consultas posibles funcionalmente equivalentes, a partir de los cuales se selecciona un conjunto de consultas para introducirse en el código de programa. Un usuario puede seleccionar el conjunto de consultas funcionalmente equivalentes posible para introducirse en el código de programa, o
15 el conjunto de consultas funcionalmente equivalentes posible puede seleccionarse automáticamente. La selección del conjunto de consultas funcionalmente equivalentes posible puede verse influida por la posibilidad de que la selección se realice durante el desarrollo (por ejemplo, durante la introducción de código o en el momento de la compilación), o se realice durante el tiempo de ejecución. Durante el tiempo de ejecución, puede seleccionarse un conjunto de consultas funcionalmente equivalentes posible entre
20 las múltiples consultas posibles para introducirse en el código de programa con base (al menos en parte) en los factores de tiempo de ejecución (por ejemplo, condiciones de ejecución como la disponibilidad de red, potencia de procesamiento disponible, etcétera).

Otras características y ventajas de la invención, así como la estructura y operación de diversas modalidades de la invención, se describen en detalle a continuación con referencia a los dibujos adjuntos.
25 Se observa que la invención no se limita a las modalidades específicas descritas en el presente documento. Tales modalidades se presentan en el presente documento únicamente con fines ilustrativos. Las modalidades adicionales serán evidentes para los expertos en la técnica relevante con base en las enseñanzas contenidas en el presente documento.

30 **Breve descripción de los dibujos**

Los dibujos adjuntos, que se incorporan en el presente documento y forman parte de la especificación, ilustran las modalidades de la presente solicitud y, junto con la descripción, sirven, además, para explicar los principios de las modalidades y para permitir que un experto en la técnica relevante realice y utilice las modalidades.

35 La figura 1 muestra un diagrama de bloques de un dispositivo informático que contiene un optimizador de consultas configurado para mejorar la eficiencia de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar.

La figura 2 muestra un diagrama de bloques de un dispositivo informático que incluye una

aplicación de desarrollo que tiene un optimizador de consultas configurado para mejorar la eficiencia de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar.

La figura 3 muestra un diagrama de flujo que proporciona un proceso para mejorar la eficiencia de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar.

5

La figura 4 muestra un diagrama de bloques de un optimizador de consultas, de acuerdo con una modalidad ejemplar.

La figura 5 muestra un diagrama de flujo que proporciona un proceso para generar y presentar un conjunto de consultas de reemplazo sugerido en un editor de código, de acuerdo con una modalidad ejemplar.

10

La figura 6 muestra un diagrama de bloques de un dispositivo informático que incluye una ventana de editor de código que presenta el código de programa con un conjunto de consultas de reemplazo sugerido, de acuerdo con una modalidad ejemplar.

La figura 7 muestra un diagrama de flujo que proporciona un proceso para generar e implementar un conjunto de consultas de reemplazo en el momento de la compilación, de acuerdo con una modalidad ejemplar.

15

La figura 8 muestra un diagrama de flujo que proporciona un proceso para extender la evaluación perezosa de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar.

20

La figura 9 muestra un diagrama de bloques de un extensor de evaluación perezosa, de acuerdo con una modalidad ejemplar.

La figura 10 muestra un diagrama de flujo que proporciona un proceso para aplicar un conjunto de reglas para determinar un conjunto de consulta de reemplazo para el código de programa, de acuerdo con una modalidad ejemplar.

25

La figura 11 muestra un diagrama de bloques de un generador de conjunto de consultas equivalente, de acuerdo con una modalidad ejemplar.

La figura 12 muestra un diagrama de flujo que proporciona un proceso para generar y seleccionar entre una pluralidad de conjuntos de consultas posibles funcionalmente equivalentes, de acuerdo con una modalidad ejemplar.

30

La figura 13 muestra un diagrama de bloques de un dispositivo informático ejemplar que puede utilizarse para implementar las modalidades.

Las características y ventajas de la presente invención se harán más evidentes a partir de la descripción detallada que se expone a continuación cuando se considera junto con los dibujos, en los cuales los caracteres de referencia similares identifican los elementos correspondientes en el presente documento. En los dibujos, los números de referencia similares generalmente indican elementos idénticos, funcionalmente similares y/o estructuralmente similares. El dibujo en el cual aparece un elemento por primera vez se indica con los dígitos extremos izquierdos en el número de referencia correspondiente.

35

Descripción detallada de la invención

I. Introducción

La presente especificación y dibujos adjuntos divulgan una o más modalidades que incorporan las características de la presente invención. El alcance de la presente invención no se limita a las modalidades divulgadas. Las modalidades divulgadas simplemente ejemplifican la presente invención, y las versiones modificadas de las modalidades divulgadas también se incluyen en la presente invención. Las modalidades de la presente invención se definen por las reivindicaciones adjuntas en el presente documento.

Las referencias en la especificación a “una modalidad”, “una modalidad”, “una modalidad ejemplar”, etcétera, indican que la modalidad descrita puede incluir una propiedad, estructura o característica particular, pero cada modalidad puede no incluir necesariamente la propiedad, estructura o característica particular. Además, tales frases no se refieren necesariamente a la misma modalidad. Además, cuando una propiedad, estructura o característica particular se describe con relación a una modalidad, se afirma que es del conocimiento de un experto en la técnica efectuar tal propiedad, estructura o característica con relación a otras modalidades, ya sea que se describa explícitamente o no.

En el análisis, a menos que se indique lo contrario, los adjetivos como “sustancialmente” y “aproximadamente” modifican una característica de condición o relación de una propiedad o propiedades de una modalidad de la divulgación, se entiende que significan que la condición o característica se define dentro de las tolerancias que son aceptables para la operación de la modalidad para una aplicación para la cual está destinada.

Numerosas modalidades ejemplares se describen posteriormente. Se señala que cualquier encabezado de sección/subsección que se proporcione en el presente documento no pretende ser limitativo. Las modalidades se describen a lo largo del presente documento, y cualquier tipo de modalidad puede incluirse en cualquier sección/subsección. Además, las modalidades divulgadas en cualquier sección/subsección pueden combinarse con cualquier otra modalidad descrita en la misma sección/subsección y/o una sección/subsección diferente de cualquier manera.

II. Ejemplo de modalidades para optimizar consultas en código de programa

Muchos lenguajes de programación modernos son compatibles de manera nativa con las consultas. Por ejemplo, Microsoft .NET Framework, desarrollado por Microsoft Corporation, es compatible con las consultas en forma de LINQ (consulta integrada en el lenguaje), mientras Java®, desarrollado por Oracle Corporation o Redwood City, California, es compatible con las consultas en forma de Streams. La compatibilidad nativa con las consultas en un lenguaje de programación permite a los desarrolladores concentrarse en la parte lógica de su código de programa al encargarse de la implementación real de las consultas para ellos. Esto puede permitir a un desarrollador acelerar su codificación. Sin embargo, no saber o tener una mala comprensión de las consecuencias de las consultas ineficientes puede conducir a que se desarrolle un código de programa que no funciona correctamente. Por ejemplo, tal código de programa puede incluir consultas que recuperan datos que no se utilizan, que recuperan cantidades redundantes de datos y/o que recuperan datos con más frecuencia o antes de lo necesario. Las consultas

pueden escribirse de muchas maneras diferentes que producen los mismos resultados lógicos, pero tienen un impacto muy diferente en el rendimiento.

Además, debido a las mejores prácticas actuales para el desarrollo de *software*, los desarrolladores tienden a dividir métodos/procedimientos más grandes en métodos/procedimientos más pequeños en su código para una mayor legibilidad. Los optimizadores de código actualmente disponibles pueden tener dificultades para optimizar el código que se ha desarrollado de esta manera.

De acuerdo con las modalidades, el código de programa se analiza automáticamente para consultas que se implementan de manera ineficiente, y las consultas se reemplazan con un conjunto de consultas más eficientes, pero lógicamente equivalentes. Tales modalidades pueden implementarse para que se apliquen durante el tiempo de compilación (por ejemplo, para la optimización de CPU) y/o se implementen como sugerencias de refactorización en herramientas IDE y/o complementos IDE. Por ejemplo, en una modalidad, un editor de código se configura para determinar un conjunto de consultas más eficiente y para sugerir el conjunto de consultas más eficiente a un desarrollador que se implementará en el código del desarrollador. Las modalidades permiten que los desarrolladores generen un código legible, después transforman automáticamente el código en un código más eficiente.

En una modalidad, un optimizador de consultas puede configurarse para: (a) identificar consultas (por ejemplo, LINQ, Streams, etcétera) en el código de programa, (b) inspeccionar los resultados de las consultas para determinar su uso/contexto, (c) descomponer las consultas en unidades atómicas, (d), agregar (a), (b) y (c) para determinar dónde pueden implementarse las optimizaciones de consultas. Por ejemplo, la coincidencia de patrones puede realizarse en comparación con un conjunto de reglas, que pueden ser reglas definidas explícitamente y/o reglas generadas automáticamente. Por ejemplo, las reglas pueden generarse automáticamente mediante un algoritmo, como aprendizaje automático, análisis de macrodatos de ejemplos de código, etcétera. Un conjunto de tales reglas puede mejorar significativamente los errores más comunes que ocurren en el trabajo diario de los desarrolladores.

En consecuencia, las modalidades proporcionan una o más de: (1) una herramienta y un proceso que analizan y recomiendan cambios de código que mejoran el rendimiento de las consultas, (2) una herramienta y un proceso que cambian automáticamente las consultas en consultas equivalentes más optimizadas, (3) una herramienta y un proceso de análisis de código de preprocesamiento para compiladores para mejoras de rendimiento automáticas para consultas, (4) una herramienta y proceso automáticos que analizan y recomiendan la refactorización de código con respecto a las consultas de bases de datos (como “estructura de entidades” en .NET y equivalentes en otros lenguajes), y una herramienta y proceso de optimización de código que implementa el aprendizaje automático, que muestrea las consultas equivalentes durante el tiempo de ejecución y selecciona y reemplaza automáticamente la mejor consulta para la situación.

Tal como se utiliza en el presente documento, una consulta es una consulta (o solicitud) de datos, y también puede denominarse como una consulta de datos. Puede proporcionarse una consulta a cualquier fuente de datos, que incluye una base de datos, aplicación, API (interfaz de programación de aplicaciones), etcétera. Una consulta a una base de datos puede denominarse “consulta de base de

datos”.

Las modalidades pueden implementarse de varias maneras. Por ejemplo, la figura 1 muestra un diagrama de bloques de un dispositivo informático 102 que contiene un optimizador de consultas 104 configurado para mejorar la eficiencia de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar. Tal como se muestra en la figura 1, el optimizador de consultas 104 recibe el código de programa 106 introducido por parte de un desarrollador (una persona que escribe/introduce/modifica el código de programa). El código de programa 106 incluye una pluralidad de consultas 110. Las consultas 110 incluyen múltiples consultas, que pueden incluir uno o más operadores de consulta separados y/o una o más expresiones de consulta (cadenas/series de operadores de consulta). El optimizador de consultas 104 se configura para analizar y generar un conjunto de consultas de reemplazo para las consultas 110, lo cual genera un código de programa refactorizado 108 que incluye las consultas de reemplazo 112. El código de programa refactorizado 108 es un código de programa que es lógicamente equivalente (realiza las mismas funciones) al código de programa original 106, pero se reescribe con las consultas de reemplazo 112. Las consultas de reemplazo 112 están configuradas para funcionar de manera más eficiente que las consultas originales 110, por ejemplo, al evitar recuperar datos no utilizados por el código de programa, evitando realizar solicitudes redundantes de recuperación de datos, evitando realizar operaciones redundantes y/o recuperar datos con menos frecuencia y/o más tarde (lo cual aumenta la evaluación perezosa).

El optimizador de consultas 104 puede implementarse de manera independiente o incluirse en cualquier sistema o herramienta que un desarrollador pueda utilizar para introducir o procesar el código de programa, como un editor de código, compilador de código, depurador de código, etcétera. Por ejemplo, la figura 2 muestra un diagrama de bloques del dispositivo informático 102 que incluye una aplicación de desarrollo 200 que incluye el optimizador de consultas 104, de acuerdo con una modalidad ejemplar. La aplicación de desarrollo 200 es un ejemplo de un entorno de desarrollo integrado (IDE, por sus siglas en inglés). Tal como se muestra en la figura 2, el dispositivo informático 102 incluye la aplicación de desarrollo 200, el almacenamiento 210 y una interfaz de comunicación 208. El almacenamiento 210 almacena los códigos de programa 106 y 108 y el conjunto de reglas 216. La aplicación de desarrollo 200 incluye un editor de código fuente 202, un compilador 204 y una herramienta de depuración 206. El editor de código fuente 202 incluye una interfaz de usuario 212. Tal como lo indican las flechas punteadas, el optimizador de consultas 104 puede implementarse o activarse por cualquiera de uno o más del editor de código fuente 202, compilador 204 y/o herramienta de depuración 206. Cabe destacar que la aplicación de desarrollo 200 se muestra con fines ilustrativos, y como una modalidad ejemplar, y no todas las características de la aplicación de desarrollo 200 deben estar presentes en todas las modalidades. Además, las características adicionales no mostradas en la figura 2 puede estar presentes en algunas modalidades. Las características de la aplicación de desarrollo 200 mostradas en la figura 2 se describen posteriormente.

Tal como se muestra en la figura 2, la aplicación de desarrollo 200 puede implementarse en uno o más dispositivos informáticos 102. Por ejemplo, el editor de código fuente 202, el compilador 204 y la

herramienta de depuración 206 pueden incluirse en un mismo dispositivo informático, o en uno o más del editor de código fuente 202, compilador 204 y herramienta de depuración 206 pueden implementarse en uno o más dispositivos informático separados de aquellos del editor de código fuente 202, compilador 204 y herramienta de depuración 206.

5 El dispositivo informático 102 puede ser uno o más de cualquier tipo de dispositivos informáticos fijo o móvil, que incluyen una computadora móvil o dispositivo informático móvil (por ejemplo, un dispositivo Microsoft® Surface®, un asistente digital personal (PDA, por sus siglas en inglés), una computadora portátil, una computadora portable, una tableta, como Apple iPad™, una minicomputadora portátil, etc.), un teléfono móvil, un dispositivo informático portátil u otro tipo de dispositivo móvil, o un
10 dispositivo informático fijo como una computadora de escritorio o PC (computadora personal).

El editor de código 202 puede ser cualquier editor de código propietario o convencional configurado para editar el código de programa mencionado en otra parte en el presente documento o de otra manera conocido (por ejemplo, un editor de código de Eclipse™, ActiveState Komodo™, IntelliJ IDEA, Oracle JDeveloper™, NetBeans, Codenvy™, Xcode®, Microsoft® Visual Studio®, etcétera).

15 Un desarrollador puede interactuar con el editor de código fuente 202 para introducir y modificar el código de programa cuando genera el código fuente para una aplicación. Por ejemplo, el desarrollador puede interactuar con una interfaz de usuario 212 del editor de código fuente 202 para agregar, modificar o eliminar el texto de código de programa como al escribir, introducir por voz, seleccionar los bloques de código sugeridos, etcétera. En consecuencia, la interfaz de usuario 212 puede incluir una o más
20 casillas/ventanas de introducción de texto (por ejemplo, la ventana de editor de código 604 de la figura 6), reconocimiento de voz/dialogo, uno o más elementos de interfaz gráfica de usuario (por ejemplo, botones, casillas de verificación, botones de radio, menús desplegables, etcétera), y/u otros elementos de la interfaz de usuario con los cuales un desarrollador puede interactuar. Una vez completado, o en otros intervalos, el usuario puede ser capaz de guardar el código de programa al interactuar con un botón
25 “guardar” u otro elemento de la interfaz de usuario.

Por ejemplo, tal como se muestra en la figura 2, un desarrollador puede interactuar con la interfaz de usuario 212 del editor de código fuente 202 para generar el código de programa 106. El código de programa 106 es un código fuente, que es un conjunto de instrucciones de computadora (posiblemente con comentarios) escritas utilizando un lenguaje de programación informático legible por humanos. Los
30 ejemplos de lenguajes de programación informáticos legibles por humanos incluyen C#, C++, Java, etcétera. El código de programa 106 puede recibirse en uno o más archivos u otra forma. Por ejemplo, el código de programa 106 puede recibirse como uno o más archivos “.c” (cuando se utiliza el lenguaje de programación C), como uno o más archivos “.cpp” (cuando se utiliza el lenguaje de programación C++), etcétera. Cuando el editor de código fuente 202 utiliza el optimizador de consultas 104 para refactorizar el
35 código de programa 106, el código de programa refactorizado 108 puede generarse y guardarse por el editor de código fuente 202. En las modalidades, el editor de código fuente 202 puede aplicar las reglas del conjunto de reglas 216 a los componentes de consulta (también denominados “operadores de consulta”) del código de programa 106 para crear un conjunto de consultas más eficiente en el código de

programa refactorizado 108.

Tal como se muestra en la figura 2, el código de programa 106 y/o 108 pueden almacenarse en el almacenamiento 210. El almacenamiento 210 puede incluir uno o más de cualquier tipo de *hardware*/circuitos de almacenamiento físico para almacenar los datos, que incluyen un disco magnético (por ejemplo, en una unidad de disco duro), un disco óptico (por ejemplo, en una unidad de disco óptico), una cinta magnética (por ejemplo, en una unidad de cinta), un dispositivo de memoria como un dispositivo de RAM, un dispositivo ROM, etcétera, y/o cualquier otro tipo adecuado de *hardware*/circuitaría de almacenamiento físico.

El compilador 204 puede invocarse de cualquier manera, por ejemplo, mediante una línea de comando, una interfaz gráfica de usuario, etcétera. Puede utilizarse un interruptor “completo”, u otro interruptor, cuando se activa el compilador 204 para realizar una compilación completa. El compilador 204 está configurado para recibir y compilar el código de programa 106 (o el código de programa 108) para generar el código máquina 222. En particular, el compilador 204 está configurado para transformar el código de programa 106 y/o 108 en el código máquina 222 en la forma de otro lenguaje informático, normalmente tiene una forma binaria, conocida como código máquina o código objeto. En algunos casos, el compilador 204 puede incluir múltiples etapas y primero puede convertir el código de programa 106 en una forma intermedia (por ejemplo, un lenguaje intermedio), que posteriormente se convierte en el código máquina 222.

El compilador 204 puede configurarse para realizar uno o más tipos de optimizaciones en el código de programa 106 y/o 108 al generar el código máquina 222. Una construcción optimizada da como resultado un código máquina semánticamente equivalente al código máquina generado sin optimizaciones, pero se configura de manera que se utilicen menos recursos durante la ejecución del código máquina optimizado (por ejemplo, menos memoria, menos solicitudes a procedimientos, etcétera). Los ejemplos de optimizaciones que pueden realizarse incluyen optimizaciones de ciclo, optimizaciones de flujo de datos, optimizaciones basadas en SSA, optimizaciones de generador de código, optimizaciones de lenguaje funcional, optimizaciones entre procedimientos y/u otros tipos de optimizaciones que serían conocidas por los expertos en la técnica relevante. Existen muchos tipos específicos de optimizaciones. Por ejemplo, puede realizarse la “inclusión por referencia”, donde se copia una función invocada por una función invocadora en el cuerpo de la función invocadora. En otro ejemplo de una optimización específica, puede realizarse una “eliminación de subexpresión común”, donde se utiliza una única instancia de código para una cantidad que se calcula varias veces en el código fuente. Cuando el compilador 204 utiliza el optimizador de consultas 104 para refactorizar el código de programa 106, el código de programa refactorizado 108 puede generarse y utilizarse para generar el código máquina 222 por el compilador 204.

El código máquina 222 puede incluirse en un archivo (por ejemplo, un objeto o archivo “.obj”), o puede crearse/almacenarse en otra forma, para formar un programa o aplicación ejecutable. El código máquina 222 puede almacenarse opcionalmente en el almacenamiento 210.

Cuando el compilador 204 compila el código de programa 106 y/o 108 para la etapa de

depuración del desarrollo, la herramienta de depuración 206 puede recibir el código máquina 222. La herramienta de depuración 206 está configurada para ejecutar una sesión depuradora (o “depurar”, “depuración”) en la aplicación representada por el código máquina 222. En una sesión depuradora, un desarrollador puede ser capaz de avanzar en la ejecución del código máquina 222, mientras revisa los valores de las variables, matrices, atributos y/o salidas (por ejemplo, contenido de registros, una GUI, etcétera) generados por la ejecución del código máquina 222, que incluye el acceso a los efectos de cualquier código/instrucción de depuración introducido en el código de programa 106 y/o 108 (y transferido al código máquina 222 por el compilador 204 para fines de depuración). De esta manera, un desarrollador puede probar, o resolver problemas (“depurar”), el código de programa 106 y/o 108, haciendo ediciones al código de programa 106 y/o 108 utilizando el editor de código fuente 202 con base en los resultados de la sesión depuradora. La versión modificada del código de programa 106 y/o 108 puede compilarse por el compilador 204 y recibirse por la herramienta de depuración 206 para una mayor depuración. Durante la depuración, la herramienta de depuración 206 puede sugerir y/o volver a escribir las consultas en el código de programa 106 para generar el código de programa 108. La herramienta de depuración 206 puede incluir uno o más procesadores (por ejemplo, una unidad central de procesamiento (CPU, por sus siglas en inglés)), física y/o virtual, que ejecuta el código máquina 222.

Cuando se completa la depuración con la herramienta de depuración 206, y el código de programa 106 y/o 108 está en su versión final, el compilador 204 puede compilar el código de programa 106 y/o 108 para generar el código máquina 222 para la etapa de lanzamiento del desarrollo. La versión de lanzamiento del código máquina 222 puede lanzarse para utilizarse por parte de los usuarios.

La interfaz de comunicación 208 está configurada para transmitir el código de programa 106 y/o 108 a entidades remotas, para recibir las reglas para mejorar el código de programa de acuerdo con las modalidades, y/o para comunicar otros datos de acuerdo con cualquier protocolo de comunicación adecuado, propietario o convencional. Otros ejemplos de interfaces de comunicación y protocolos de comunicación se describen en la siguiente sección.

El optimizador de consultas 104 puede configurarse de varias maneras para realizar sus funciones. Por ejemplo, la figura 3 muestra un diagrama de flujo 300 que proporciona un proceso para mejorar la eficiencia de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar. El optimizador de consultas 104 puede funcionar de acuerdo con el diagrama de flujo 300 en una modalidad. El diagrama de flujo 300 se describe posteriormente con referencia a la figura 1 y 4. La figura 4 muestra un diagrama de bloques del optimizador de consultas 104, de acuerdo con una modalidad ejemplar. Tal como se muestra en la figura 4, el optimizador de consultas 104 incluye un detector de consultas 402, un extensor de evaluaciones perezosa 404 y un generador de conjuntos de consultas equivalentes 406, que se describen posteriormente con referencia al diagrama de flujo 300.

El diagrama de flujo 300 comienza con la etapa 302. En la etapa 302, se detecta una pluralidad de consultas en el código de programa. Tal como se muestra en la figura 1, el optimizador de consultas 104 recibe el código de programa 106, que incluye las consultas 110. El detector de consultas 402 de la figura 4 está configurado para analizar a través del código de programa 106 para detectar las consultas.

El detector de consultas 402 puede detectar las consultas de cualquier manera, como al comparar los términos de código en el código de programa 106 con una lista predeterminada de operadores/componentes de consulta conocidos, etcétera. Por ejemplo, cuando se buscan las consultas LINQ, el detector de consultas 402 puede analizar el código de programa 106 en busca de los operadores LINQ conocidos, como Select, Where, Sum, Min, Max, Average, Aggregate, Join, GroupJoin, OrderBy, GroupBy, Union, Contains, Count, ToList, ToDictionary, etcétera. Cada operador de consulta encontrado (por ejemplo, encontrado por coincidencia de texto) se indica mediante el detector de consultas 402 como una consulta para el código de programa 106.

En un ejemplo ilustrativo, el detector de consultas 402 puede analizar el siguiente código de programa para las consultas:

```
public void Main()
var 1stOfPrimes = ReturnListOfPrimes(1000);
var 1stOrdered = 1stOfPrimes.OrderBy(x => x).ToList();
var 1stOrderedDesc = 1stOrdered.OrderByDescending(x => x).ToList();
Console.WriteLine("Biggest prime in range is: {0}", 1stOrderedDesc.FirstO);
```

En este ejemplo, el detector de consultas 402 puede detectar las consultas LINQ de Orderby(x => x) y ToList() en la tercera línea de código, y las consultas LINQ de OrderByDescending(x => x) y ToList() en la cuarta línea de código.

Cabe destacar que el optimizador de consultas 104 puede implementar la etapa 302 (y el resto del diagrama de flujo 300) en cualquier herramienta o aplicación de desarrollo de código adecuada. Por ejemplo, la figura 5 muestra un diagrama de flujo 500 que proporciona un proceso para generar y presentar un conjunto de consultas de reemplazo sugerido en un editor de código, de acuerdo con una modalidad ejemplar. En una modalidad, el diagrama de flujo 300 de la figura 3 puede implementar el diagrama de flujo 500 de la figura 5. Por ejemplo, la etapa 302 del diagrama de flujo 300 puede implementar la etapa 502 del diagrama de flujo 500, y la etapa 504 puede ser una etapa adicional al diagrama de flujo 300. El diagrama de flujo 500 se describe a continuación.

En la etapa 502, las consultas se detectan en el código de programa en un editor de código. En una modalidad, el detector de consultas 402 puede configurarse para detectar las consultas en el código de programa 106 en un editor de código, como el editor de código 202 (figura 2), donde un desarrollador introduce y edita el código de programa. El detector de consultas 402 puede realizar una búsqueda de términos de consulta en el código de programa 106 a medida que el desarrollador introduce el código, cada vez que el desarrollador guarda el código, en respuesta a la solicitud del desarrollador (por ejemplo, al hacer clic en el botón “detectar consulta”), de forma periódica, y/o en cualquier otro momento o forma deseada.

Por ejemplo, la figura 6 muestra un diagrama de bloques del dispositivo informático 102 que incluye una ventana de edición de código 604 que muestra el código de programa 106, de acuerdo con una modalidad ejemplar. La pantalla 602 puede ser cualquier tipo de dispositivo o pantalla de visualización adecuado, que incluye una pantalla LCD (pantalla de cristal líquido), una pantalla CRT (tubo

de rayos catódicos), una pantalla LED (diodo emisor de luz), una pantalla de plasma, etcétera. La ventana de editor de código 604 es una ventana (enmarcada o sin marco) mostrada por el editor de código 202 en la pantalla 602 como una interfaz gráfica de usuario (GUI, por sus siglas en inglés) para la interacción con el código de programa 106 que se muestra en la ventana de editor de código 604. Cada una de las líneas de código 606, 608 y 610 son una o más líneas de código de cualquier lenguaje de programación adecuado, como conocerían los expertos en la técnica relevante. En una modalidad, el detector de consultas 402 analiza el código de programa 106, incluidas las líneas de código 606, 608 y 610, para consultas de acuerdo con la etapa 502.

En la etapa 504, se presenta una opción para que el editor de código refactorice automáticamente el código de programa para reemplazar la pluralidad de consultas con el conjunto de consultas funcionalmente equivalentes. En una modalidad, después de que el optimizador de consultas 104 detecte las consultas y determine las consultas más eficientes para reemplazar las consultas detectadas en el código de programa 106 (tal como se describe con mayor detalle posteriormente en el presente documento), el editor de código 202 puede presentar una opción para que el desarrollador acepte o rechace las consultas más eficientes determinadas. Por ejemplo, tal como se muestra en la figura 6, puede mostrarse un conjunto de consultas de reemplazo sugerido 612 en la ventana de editor de código 604 para que el desarrollador lo acepte o rechace. El conjunto de consultas de reemplazo sugerido 612 es un conjunto de consultas sugerido determinado por el optimizador de consultas 104 para que sea más eficiente que el conjunto de consultas detectadas en el código de programa 106. Si el desarrollador acepta el conjunto de consultas de reemplazo sugerido 612 (por ejemplo, haciendo clic en un botón), el editor de código 202 puede reemplazar el conjunto de consultas detectado con el conjunto de consultas de reemplazo sugerido 612 en el código de programa 106 para generar y mostrar el código de programa refactorizado 108 en la ventana de editor de código 604. Si el desarrollador rechaza el conjunto de consultas de reemplazo sugerido 612, no se realiza ningún cambio en el programa código 106, y la sugerencia ya no se muestra. Tenga en cuenta que, en otra modalidad, el conjunto de consultas de reemplazo sugerido 612 puede implementarse automáticamente en el código de programa 106 (sin solicitar la aceptación del desarrollador).

En otra modalidad, tal como se describió anteriormente, el detector de consultas 402 puede configurarse para detectar las consultas en el código de programa 106 durante la compilación. Por ejemplo, la figura 7 muestra un diagrama de flujo 700 que proporciona un proceso para generar e implementar un conjunto de consultas de reemplazo en el tiempo de compilación, de acuerdo con una modalidad ejemplar. En una modalidad, el diagrama de flujo 300 de la figura 3 puede implementar el diagrama de flujo 700 de la figura 7. Por ejemplo, la etapa 302 del diagrama de flujo 300 puede implementar la etapa 702 del diagrama de flujo 700, y la etapa 704 puede ser una etapa adicional al diagrama de flujo 300. El diagrama de flujo 700 se describe a continuación.

En la etapa 702, las consultas se detectan en el código de programa durante la compilación. En una modalidad, el detector de consultas 402 puede configurarse para detectar las consultas en el código de programa 106 (por ejemplo, mediante la coincidencia de texto, etcétera) en un compilador, como el

compilador 204 (figura 2), cuando se compila el código de programa 106.

En la etapa 704, el código compilado se genera en el compilador con base en una versión del código de programa donde la pluralidad de consultas se reemplaza con el conjunto de consultas funcionalmente equivalentes. En una modalidad, tal como se describe con más detalle posteriormente, el optimizador de consultas 104 está configurado para generar un conjunto de consultas de reemplazo más eficiente para las consultas detectadas en el código de programa 106 por el detector de consultas 402, para reemplazar las consultas detectadas con el conjunto de consultas de reemplazo en una copia del código de programa 106 para generar el código de programa refactorizado 108, y para generar una versión compilada del código de programa 106 mediante la compilación del código de programa refactorizado 108. De esta manera, se genera un código máquina compilado que incluye el conjunto de consultas de operación más eficiente sin cambiar el código de programa 106, de manera que el desarrollador no vea las consultas modificadas cuando edite posteriormente el código de programa 106 en el editor de código 202. Sin embargo, en otra modalidad, el conjunto de consultas de reemplazo puede implementarse automáticamente en el código de programa 106 en el momento de la compilación (sin solicitar la aceptación del desarrollador).

Con referencia al diagrama de flujo 300 de la figura 3, en la etapa 304, se extiende la evaluación perezosa mediante la cual las consultas se evalúan en el código de programa. En una modalidad, el extensor de evaluación perezosa 404 se configura para analizar las consultas detectadas en el código de programa 106 por el detector de consultas 402, y para extender la evaluación perezosa de la evaluación de las consultas (cuando se ejecuta el código de programa). La “evaluación perezosa” se refiere a una técnica de evaluación que retrasa la evaluación de una expresión hasta que se necesita su valor. El extensor de evaluación perezosa 404 está configurado para recibir una consulta y posponer la ejecución de la mayor cantidad de sus operadores lo más tarde posible. Por consiguiente, el extensor de evaluación perezosa 404 analiza el código de programa 106 para determinar las consultas que pueden retrasar su evaluación, y determina una o más declaraciones de consulta equivalentes que realizan las consultas en forma diferida.

Por ejemplo, la figura 8 muestra un diagrama de flujo 800 que proporciona un proceso para extender la evaluación perezosa de la ejecución de consultas en el código de programa, de acuerdo con una modalidad ejemplar. En una modalidad, el extensor de evaluación perezosa 404 opera de acuerdo con el diagrama de flujo 800. El diagrama de flujo 800 se describe posteriormente con respecto a la figura 9. La figura 9 muestra un diagrama de bloques del extensor de evaluación perezosa 404, de acuerdo con una modalidad ejemplar. Tal como se muestra en la figura 9, el extensor de evaluación perezosa 404 incluye un detector de consultas extensible 902 y un ensamblador de expresiones de consulta 904.

El diagrama de flujo 800 comienza con la etapa 802. En la etapa 802, se detectan una o más de las consultas que son extensibles a la evaluación. En una modalidad, el detector de consultas extensibles 902 del extensor de evaluación perezosa 404 analiza el código de programa 106 para determinar las consultas que pueden tener retrasada su evaluación debido a que sus valores de salida no son necesarios en su ubicación actual en el código de programa 106, sino que se utilizan posteriormente. En

una modalidad de este tipo, el detector de consultas extensibles 902 está configurado, para una consulta particular, para analizar más allá de la consulta en el código de programa 106 para encontrar una ubicación (por ejemplo, una expresión, una línea de código, etcétera) en o cerca de donde el valor de la consulta es necesaria, por ejemplo, cuando el valor se envía directamente a un usuario o cuando una expresión utiliza el valor para generar una salida para el usuario.

En una modalidad, el detector de consultas extensibles 902 comienza por construir una estructura de datos como un árbol de operaciones para cada consulta de las consultas detectadas 906, que incluye las consultas detectadas por el detector de consultas 402 en el código de programa, así como las operaciones de código de programa de no consulta. El detector de consultas extensibles 902 enumera múltiples consultas relacionadas en el mismo árbol de operación. Tal árbol de operación puede representarse como una gráfica, donde cada nodo representa un operador de consulta coincidente de la consulta que no extrae los datos reales de la fuente de datos (por ejemplo, realiza una operación en los datos), y cada hoja del árbol de operación representa un operador de consulta de la consulta que extrae los datos reales de la fuente de datos. El detector de consultas extensibles 902 construye el árbol de operaciones siguiendo cada variable/operador de consulta/método/instrucción de código en el código de programa y agregando recursivamente los datos al árbol de operación. En consecuencia, el detector de consultas extensibles 902 recibe las consultas y las transforma en el árbol relevante.

Por ejemplo, en un ejemplo, el detector de consultas extensibles 902 puede analizar el siguiente código de programa para determinar una o más consultas LINQ que pueden retrasar su evaluación:

```
var 1stOfPrimes = ReturnListOfPrimes (1000);
var 1stOrdered = 1stOfPrimes.OrderBy(x => x).ToList();
Console.WriteLine("Smallest prime in range is: {0}", 1stOrdered.First());
```

Este código de programa ejemplar incluye las consultas LINQ detectadas de OrderBy, ToList y First. El detector de consultas extensibles 902 designa el siguiente árbol de operaciones:

```
1stOfPrimes - ReturnListOfPrimes - OrderBy - ToList - First
```

donde 1stOfPrimes es el elemento de nivel más alto del árbol de operación, y First es el elemento de nivel más bajo del árbol de operación. En este ejemplo, el detector de consultas extensibles 902 indica OrderBy y ToList como extensible (ejecución retrasada) debido a que el valor de 1stOrdered no se necesita de inmediato (es una entrada a la línea de código posterior), pero First es una hoja, y por lo tanto, no se indica como extensible.

En otro ejemplo, el detector de consultas extensibles 902 puede analizar el siguiente código de programa para determinar una o más consultas LINQ que pueden retrasar su evaluación:

```
var 1stOfPrimes = ReturnListOfPrimes (1000);
var 1stOrdered = 1stOfPrimes.OrderBy(x => x).ToList();
var LastOrdered = 1stOfPrimes.OrderByDescending(x => x).ToList();
Console.WriteLine("Smallest prime in range is: {0}", 1stOrdered.First());
Console.WriteLine ("Biggest prime in range is: {0}", LastOrdered.First());
```

Este código de programa ejemplar incluye las consultas LINQ detectadas de OrderBy, ToList,

OrderByDescending, ToList, First y First. El detector de consultas extensibles 902 designa un árbol de operación que tiene las siguientes primera y segunda ramas:

1stOfPrimes - ReturnListOfPrimes - OrderBy - ToList - First

y

5 1stOfPrimes - ReturnListOfPrimes - OrderByDescending - ToList - First

En este árbol de operación, un elemento 1stOfPrimes común es el elemento de nivel más alto, y los nodos OrderBy y OrderByDescending se ramifican desde un nodo ReturnListOfPrimes común vinculado al elemento 1stOfPrimes. En este ejemplo, el detector de consultas extensibles 902 indica OrderBy y ToList en la primera rama y OrderByDescending y ToList en la segunda rama como extensible (ejecución retrasada) debido a que los valores de 1stOrdered y LastOrdered no son necesarios inmediatamente (son entradas a la línea de código subsiguiente), pero First en la primera rama y First en la segunda rama son ambas hojas, y, por lo tanto, no se indican como extensibles.

En otro ejemplo, el detector de consultas extensibles 902 puede analizar el siguiente código de programa ejemplar para determinar una o más consultas LINQ que pueden retrasar su evaluación. Este código de programa ejemplar está diseñado para encontrar el último número primo en el intervalo de 2 a 1000:

```

    public List<int> ReturnListOfPrimes(int range)
    {
        return Enumerable.Range(2, range).Where(IsPrime).ToList();
    }
    public Dictionary<int, double> ReturnDictionaryOfPower(List<int> 1stOfInts, int pow)
    {
        return 1stOfInts.ToDictionary(x => x, x => Math.Pow(x, pow));
    }
    public void MainMethod()
    {
        var 1stOfPrimes = ReturnListOfPrimes(1000);
        var power = ReturnDictionaryOfPower(1stOfPrimes, 2);
        var lastItem = power.Last();
        Console.WriteLine("Last prime: {0}, { 1 }", lastItem.Key, lastItem.Value);
    }

```

En este ejemplo, el tercer método ("MainMethod") hace referencia a dos métodos anteriores ("ReturnListOfPrimes" y "ReturnDictionaryOfPower"). El primer método incluye los componentes de consulta de Range, Where y ToList. El segundo método incluye el componente de consulta ToDictionary. El tercer método incluye el último componente de consulta. El detector de consultas 402 detecta estas consultas en los tres métodos (etapa 302 de la figura 3). Tal como se muestra en la figura 9, el detector de consultas extensibles 902 recibe las consultas detectadas 906, que incluyen las consultas detectadas por el detector de consultas 402 en el código de programa. En una modalidad, el detector de consultas

extensibles 902 analiza las consultas detectadas 906 y el código de programa 106 (por ejemplo, al generar un árbol de operación, no mostrado aquí por motivos de brevedad) para determinar si las consultas detectadas pueden evaluarse perezosamente en el código de programa 106.

En particular, el tercer método incluye cuatro líneas de código. Cuando se ejecuta el tercer método, la primera línea accede al primer método para un primer valor, la segunda línea accede al segundo método para un segundo valor (con base en el primer valor), la tercera línea realiza una consulta (Last) relacionada con el segundo valor para generar un tercer valor, y la cuarta línea genera una salida con base en el tercer valor. En consecuencia, el detector de consultas extensibles 902 determina que la primera y la segunda línea generan valores que no son necesarios de forma inmediata (por ejemplo, no se envían al usuario ni a la base de datos), sino que se utilizan como entradas para las líneas de código subsiguientes y, por lo tanto, la evaluación de las consultas en la primera y segunda líneas pueden retrasarse. Tal como se muestra en la figura 9, el detector de consultas extensibles 902 genera las consultas extensibles 908, lo que indica que las consultas determinadas por el detector de consultas extensibles 902 son extensibles.

Con referencia nuevamente al diagrama de flujo 800 en la figura 8, en la etapa 804, se forma una expresión de consulta única que incluye una o más consultas extensibles por evaluación. En una modalidad, el ensamblador de expresiones de consulta 904 recibe las consultas extensibles 908 y genera una única expresión de consulta que incluye las consultas que se determinan como extensibles por la evaluación (en la etapa 802). Tal como se muestra en la figura 9, el ensamblador de expresiones de consulta 904 genera las consultas modificadas 910, que incluyen la expresión de consulta única que combina las consultas extensibles a evaluación determinadas por el detector de consultas extensibles 902 para el código de programa. Cabe destacar que, para un código de programa particular, el ensamblador de expresiones de consulta 904 puede generar una o más de tales expresiones de consulta que combinan las consultas extensibles a evaluación. Las técnicas realizadas por el ensamblador de expresiones de consulta 904 para combinar las consultas separadas en una sola expresión de consulta serán conocidas para los expertos en la técnica relevante, y dependerán del lenguaje de consulta particular. En un ejemplo, una variable en una expresión puede reemplazarse con una expresión utilizada en otro lugar para determinar la variable.

Por ejemplo, con respecto al código de programa ejemplar anterior, el ensamblador de expresiones de consulta 904 puede generar la siguiente expresión de consulta única para la expresión “var lastItem”, que incluye las consultas LINQ del primer y segundo método y las tres primeras líneas del tercer método:

```
Public void MainMethod()
{
    var lastItem = Enumerable.Range(2, 1000).Where(IsPrime).ToList().ToDictionary(x => x, x =>
    Math.Pow(x, pow)).Last();
    Console.WriteLine("Last prime: {0}, {1}", lastItem.Key, lastItem.Value);
}
```

Tal como se mostró anteriormente, la expresión “var lastItem” es un conjunto agregado de componentes de consulta que incluye los componentes de consulta Range, Where, ToList, ToDictionary y Last de los tres métodos, y es funcionalmente equivalente al código original del programa que se muestra más arriba. De esta manera, los componentes de consulta Range, Where, ToList, ToDictionary y Last se evalúan perezosamente cuando se ejecuta “MainMethod”, y solo se evalúan inmediatamente antes de que se presente la salida de “lastItem”.

Cabe destacar que, en una modalidad, el ensamblador de expresiones de consulta 904 puede operar al recibir el árbol de operación generado por el detector de consultas extensibles 902 para una consulta, y en la estructura de datos del árbol de operación, los operadores de consulta indicar/marcar que explícitamente hacen que la ejecución de la consulta se “desactive” si no es una hoja. Dicho de otro modo, el ensamblador de expresiones de consulta 904 puede marcar el árbol de operación de tal manera que la ejecución de la consulta solo ocurra en las hojas, extendiendo así la evaluación perezosa de la consulta a un último punto en el tiempo.

Con referencia nuevamente al diagrama de flujo 300 en la figura 3, en la etapa 306, un conjunto de reglas que incluye una pluralidad de reglas se aplica a los componentes de consulta para generar un conjunto de consultas funcionalmente equivalentes a la pluralidad de consultas que se evalúa de manera más eficiente con relación a la pluralidad de consultas. En una modalidad, el generador de conjuntos de consultas equivalentes 406 (figura 4) está configurado para analizar las consultas detectadas (consultas 110 de la figura 1) y generar un conjunto de consultas alternativas (consultas de reemplazo 1212 de la figura 1) que es funcionalmente equivalente a las consultas detectadas, pero se ejecuta de manera más eficiente que la configuración original de las consultas detectadas. Por ejemplo, el generador de conjuntos de consultas equivalentes 406 puede generar las consultas de reemplazo 112 para evitar recuperar datos que no utiliza el código de programa, para evitar realizar recuperaciones redundantes de datos, para evitar realizar operaciones en datos que no tienen efecto en la salida, etcétera. Para implementar estas eficiencias, el generador de conjuntos de consultas equivalentes 406 puede eliminar los componentes de consulta, agregar los componentes de consulta y/o modificar los componentes de consulta.

El generador de conjunto de consultas equivalentes 406 puede realizar sus funciones de varias maneras. Por ejemplo, la figura 10 muestra un diagrama de flujo 1000 que proporciona un proceso para aplicar un conjunto de reglas para determinar un conjunto de consultas de reemplazo para el código de programa, de acuerdo con una modalidad ejemplar. El generador de consultas equivalentes 406 puede funcionar de acuerdo con el diagrama de flujo 1000 en una modalidad. El diagrama de flujo 1000 se describe a continuación con respecto a la figura 11. La figura 11 muestra un diagrama de bloques del generador de conjuntos de consultas equivalentes 406, de acuerdo con una modalidad ejemplar. Tal como se muestra en la figura 11, el generador de conjuntos de consultas equivalentes 406 incluye un determinador de contexto 1104 y un selector de reglas 1106, y está conectado de manera comunicativa con el almacenamiento 210 que contiene el conjunto de reglas 216.

El diagrama de flujo 1000 comienza con la etapa 1002. En la etapa 1002, se determina un

contexto lógico común entre múltiples componentes de consulta. Tal como se muestra en la figura 11, el determinador de contexto 1104 recibe las consultas modificadas 910, que son las consultas detectadas por el detector de consultas 402 y reescritas para una evaluación más perezosa por el extensor de evaluación perezosa 404. El determinador de contexto 1104 analiza las consultas modificadas 910 para un contexto lógico común. Por ejemplo, el determinador de contexto 1104 puede descomponer las consultas modificadas 910, que pueden ser una única expresión de consulta, en un conjunto de componentes de consulta, y compara los componentes de consulta entre sí. Cuando se detectan los tipos de componentes de consulta similares o relacionados (por ejemplo, OrderBy y OrderByDescending, etcétera), se establece un contexto lógico común entre los componentes de la consulta. Tal como se muestra en la figura 11, el determinador de contexto 1104 genera las consultas contextuales asociadas 1114, que incluyen grupos de componentes de consulta de las consultas modificadas 910 que están asociadas contextualmente.

En la etapa 1004, una regla del conjunto de reglas correspondiente al contexto lógico común se aplica a los múltiples componentes de consulta. Tal como se muestra en la figura 11, el selector de reglas 1106 recibe las consultas contextuales asociadas 1114. El selector de reglas 1106 analiza las consultas asociadas contextualmente 1114 para la aplicabilidad de las reglas del conjunto de reglas 216, como una primera regla 1108, una segunda regla 1110, etcétera. Cada regla del conjunto de reglas 216 está configurada para aplicarse a los componentes de consulta de un contexto común correspondiente, y para reescribir los componentes de consulta para una mayor eficiencia, como al agregar, modificar y/o eliminar los componentes de consulta de las consultas asociadas contextualmente. Cualquier número y variedad de reglas pueden incluirse en el conjunto de reglas 216. A continuación, se describen siete ejemplos de reglas que pueden incluirse en el conjunto de reglas 216 a modo de ilustración. Cada regla se describe posteriormente con un nombre de regla ejemplar, una síntesis y una descripción de la función/mecánica de la regla:

(A) Nombre de la regla: Minimizador OrderBy

Síntesis: Utilizado para eliminar las operaciones innecesarias de OrderBy.

Mecánica de reglas: El minimizador OrderBy recorre una lista de operadores OrderBy que se encuentran en el código de programa y encuentra cualquiera de los operadores OrderBy cuyo resultado no se utiliza en el código de programa (donde “utilizado” significa que no se basa en el ordenamiento realizado por el operador OrderBy — no se genera o requiere por una expresión posterior o método). Para tal operador OrderBy encontrado, el minimizador OrderBy elimina al operador OrderBy como redundante del código de programa, y reescribe cualquier otra línea del código de programa afectado por la eliminación de la instrucción de línea de código (por ejemplo, reemplazando los nombres de variables, etcétera).

(B) Nombre de regla: Colección ToCurrent

Resumen: Se utiliza en el caso de una consulta complicada que implementa el siguiente tipo de operadores “ToCurrent” varias veces seguidas: ‘ToList’, ‘ToDictionary’, etcétera. Cada uno de estos operadores ToCurrent es costoso, y en una secuencia de ellos, son redundantes, excepto el último

operador en la secuencia.

Mecánica de reglas: se detecta el siguiente patrón — dos o más de tales operadores en un conjunto agregado de componentes de consulta (por ejemplo, la expresión de consulta única que contiene múltiples componentes de consulta que se generó en la etapa 804 del diagrama de flujo 800).

- 5 Todos los operadores de tipo ToCurrent se descartan del conjunto agregado de componentes de consulta excepto uno (sin cambiar la lógica).

(C) Nombre de regla: Minimizador DataRetrival - Tipo 1

- 10 Resumen: Se utiliza cuando una gran cantidad de datos solicitados por un operador de consulta no se utiliza en última instancia y, por lo tanto, la recuperación de datos puede reducirse de forma segura o no se realiza en absoluto.

Mecánica de reglas: se detecta el siguiente patrón — un operador de consulta recupera los datos, los datos recuperados se ordenan y solo se utiliza el primer o último elemento de los datos ordenados. Reemplazar esta consulta por una consulta que simplemente implemente el elemento de valor mínimo o recuperar el elemento de valor máximo, respectivamente, a partir de los datos.

15 **(D) Nombre de regla: Minimizador DataRetrival - Tipo 2**

Resumen: Se utiliza cuando una gran cantidad de datos solicitados por un operador de consulta no se utiliza en última instancia y, por lo tanto, la recuperación de datos puede reducirse de forma segura o no se realiza en absoluto.

- 20 Mecánica de reglas: Se detecta el siguiente patrón — un operador de consulta recupera los datos, y solo se utiliza el primer o último elemento de los datos. Reemplazar esta consulta por una consulta que solo recupera cualquier elemento de datos aleatorios a partir de los datos.

(E) Nombre de la regla: Minimizador DataRetrival - Tipo 3

Resumen: se utiliza cuando se recupera una entidad de datos muy grande, pero solo se utiliza una parte de los datos.

- 25 Mecánica de reglas: Se detecta el siguiente patrón — se solicitan los datos que tienen el potencial de ser un gran conjunto de datos (por ejemplo, datos XML, datos JSON, un archivo completo, etcétera), mientras solo se utilizará un subconjunto de los datos recuperados. Reemplazar esta consulta por una consulta que solo recupera el subconjunto de datos.

Los ejemplos de la aplicabilidad de la regla “Minimizador DataRetrival - Tipo 3” incluyen:

- 30 (1) Solicitar la recuperación de un archivo, pero en última instancia solo utilizar los metadatos del archivo para determinar qué tan grande es el archivo (en lugar de utilizar el contenido del archivo). Esta regla puede utilizarse para evitar el uso de un operador de consulta para recuperar el archivo completo, pero en su lugar, usar un operador de consulta para recuperar los metadatos del archivo (que pueden ser varios factores más pequeños en tamaño que el archivo completo).

- 35 (2) Solicitar una sola línea de base de datos o entidad, y en última instancia, utilizar solo una columna de esa línea/entidad. La línea/entidad DB completa puede ser muy grande. Esta regla puede utilizarse para reemplazar el operador de consulta con otro operador de consulta que recupera solo la columna deseada.

(F) Nombre de la regla: Minimizador de iteración de enumeración

Resumen: Se utiliza cuando una consulta utiliza un solo valor y, por lo tanto, no es necesario crear toda la estructura de datos en la memoria.

Mecánica de reglas: Se detecta el siguiente patrón — solo se utiliza un único valor después de un operador de tipo “ToCurrent”. Cambia el primero con el posterior.

(G) Nombre de regla: Revertir la enumeración cuando se utiliza el operador Last()

Resumen: se utiliza en los casos en los cuales solo se necesita el último elemento, y por lo tanto, la enumeración de principio a fin es ineficaz. Al enumerar en orden inverso aumentamos el rendimiento.

Mecánica de reglas: Se detecta el siguiente patrón — solo el último valor se utiliza después de un operador de tipo “ToCurrent”. Reemplaza la enumeración con su operador coincidente “inverso” y/o invierte los valores del intervalo de enumeración, y reemplaza el operador “Last” con el operador “First”.

Como ilustración adicional, las reglas ejemplares se aplican a la expresión de consulta única generada anteriormente, que se repite a continuación para facilitar la descripción:

```
var lastItem = Enumerable.Range(2, 1000).Where(IsPrime).ToList().ToDictionary(x => x, x =>
Math.Pow(x, pow)).Last();
```

tal como se describió anteriormente, la evaluación perezosa se extendió (etapa 304 del diagrama de flujo 300; extensor de evaluación perezosa 404) para generar este conjunto agregado funcionalmente equivalente de componentes de consulta. Además, la eficiencia de esta expresión de consulta única puede mejorarse aún más al aplicar las reglas del conjunto de reglas 216 (etapa 306 del diagrama de flujo 300; generador de conjunto de consultas equivalente 406).

Por ejemplo, puede aplicarse la regla del minimizador de recopilación de ToCurrent (regla (B) anterior). En tal caso, el determinador de contexto 1104 detecta los operadores ToList y ToDictionary en la instrucción de consulta anterior, que establecen un contexto lógico común. El selector de reglas 1106 aplica la regla del minimizador de recopilación de ToCurrent, que se asocia con este contexto lógico común en el conjunto de reglas 216. La regla del minimizador de recopilación de ToCurrent elimina todos los operadores ToCurrent de la instrucción de consulta, excepto el último, para generar la siguiente expresión de consulta más eficiente, pero funcionalmente equivalente:

```
var lastItem = Enumerable. Range(2, 1000).Where(IsPrime).ToDictionary(x => x, x =>
Math.Pow(x, pow)).Last();
```

Además, la regla del minimizador de iteración de enumeración (regla (F) anterior) puede utilizarse para reubicar a un operador Last()/First() para que se ubique antes de una instrucción de ToCurrent (en este caso ToDictionary). De esta manera, la instrucción de consulta se itera hasta que se encuentra el primer elemento y, por lo tanto, no se rellena toda la estructura de datos. En su lugar, el operador ToDictionary operará en una sola entidad. En consecuencia, el determinador de contexto 1104 determina el operador ToDictionary seguido de un operador Last en la instrucción de consulta anterior para establecer un contexto lógico común. El selector de reglas 1106 aplica la regla de iteración Enumerable, que se asocia con este contexto lógico común. La regla de iteración Enumerable coloca el operador Last antes que el operador ToDictionary, para generar la siguiente expresión de consulta más eficiente, pero

funcionalmente equivalente:

```
var lastItem = Enumerable.Range(2, 1000).Where(IsPrime).Last().ToDictionary(x => x, x =>
Math.Pow(x, pow))
```

Sin embargo, puede utilizarse la enumeración inversa cuando se utiliza el operador Last() (regla (G) anterior) que invierte el orden de búsqueda de un último elemento. Esto se debe a que es mucho más eficiente proceder en orden inverso al buscar el último elemento. Por lo tanto, esta regla enumera los números en orden inverso y toma el primer elemento en lugar del último. En consecuencia, el determinador de contexto 1104 determina el último operador que precede al operador ToDictionary en la instrucción de consulta anterior para establecer un contexto lógico común. El selector de reglas 1106 aplica la regla de optimización de Enumerable, que se asocia con este contexto lógico común. La regla de optimización de Enumerable reemplaza el último operador con un primer operador, para generar la siguiente expresión de consulta más eficiente, pero funcionalmente equivalente:

```
var lastItem = Enumerable.Range(1000, 2).Where(IsPrime).First().ToDictionary(x => x, x =>
Math.Pow(x, pow))
```

Esta consulta final itera los números enteros a partir de 1100 hacia abajo, encuentra el primer número primo y realiza esto de manera mucho más eficiente (por ejemplo, utilizando menos memoria para almacenar los datos y mejor utilizando la CPU) que la versión original de tres métodos mostrada anteriormente con más detalle.

Cabe destacar que, en una modalidad, el generador de conjuntos de consultas equivalentes 406 puede aplicar secuencialmente las reglas del conjunto de reglas 216 hasta que no se puedan aplicar más reglas para determinar un único conjunto de consultas funcionalmente equivalentes. En otra modalidad, los múltiples conjuntos diferentes de conjuntos de consultas funcionalmente equivalentes pueden determinarse por el generador de conjuntos de consultas equivalentes 406, donde cada conjunto se determina como aplicable para reemplazar las consultas detectadas en el código de programa 106. En tal modalidad, el selector de reglas 1106 puede permitir al desarrollador seleccionar manualmente uno de los conjuntos de consultas funcionalmente equivalentes para aplicarse al código de programa 106 para generar el código de programa refactorizado 108. Alternativamente, el selector de reglas 1106 puede realizar la selección automáticamente.

Por ejemplo, la figura 12 muestra un diagrama de flujo 1200 que proporciona un proceso para generar y seleccionar entre una pluralidad de conjuntos de consultas posibles funcionalmente equivalentes, de acuerdo con una modalidad ejemplar. En una modalidad, el selector de reglas 1106 puede realizar el diagrama de flujo 1200. El diagrama de flujo 1200 se describe posteriormente.

En la etapa 1202, la aplicación de una pluralidad de combinaciones de reglas del conjunto de reglas a los componentes de consulta se evalúa para generar una pluralidad de conjuntos de consultas posibles funcionalmente equivalentes. En una modalidad, el selector de reglas 1106 puede aplicar diferentes combinaciones y/o órdenes de reglas al código de programa para generar diferentes conjuntos de consultas funcionalmente equivalentes. Por ejemplo, un código de programa particular puede incluir los componentes de consulta OrderBy, GroupBy y ThenBy en una expresión de consulta. El selector de

reglas 1106 puede aplicar una primera regla a la expresión de consulta que reemplaza OrderBy y GroupBy con un primer operador de consulta para generar un primer conjunto de consultas funcionalmente equivalentes, y entonces, alternativamente, puede aplicar una segunda regla a la expresión de consulta que reemplaza GroupBy y ThenBy, con un segundo operador de consulta para generar un segundo conjunto de consultas funcionalmente equivalente. De esta manera, se generan dos conjuntos de consultas funcionalmente equivalentes para una misma expresión de consulta, que pueden tener cada una sus propias características de eficiencia. Puede generarse cualquier número de conjuntos de consultas funcionalmente equivalentes para una misma expresión de consulta, dependiendo de los operadores de consulta particulares en la expresión de consulta y las reglas disponibles en el conjunto de reglas 216.

En la etapa 1204, se selecciona un conjunto de consultas funcionalmente equivalentes posible de la pluralidad de conjuntos de consultas posibles funcionalmente equivalentes que tienen la mayor ganancia de eficiencia para que sea el conjunto de consultas funcionalmente equivalentes generado. En el ejemplo anterior, el selector de reglas 1106 puede permitir que el desarrollador seleccione cuál del primer y segundo conjunto de consultas funcionalmente equivalentes se refactoriza en el código de programa, o puede seleccionar automáticamente cuál del primer y segundo componentes de consulta se refactoriza en el código de programa. Por ejemplo, el selector de reglas 1106 puede seleccionar uno de los conjuntos de consultas funcionalmente equivalentes disponibles que tengan la mayor eficiencia. Esta mayor eficiencia puede determinar de varias maneras.

Por ejemplo, cada regla en el conjunto de reglas 216 puede tener un valor de eficiencia correspondiente (por ejemplo, un número de 0-1) que indica una eficiencia relativa para la regla. El selector de reglas 1106 puede combinar los valores de eficiencia para cada una de las reglas utilizadas para un conjunto de consultas funcionalmente equivalentes en particular para determinar un valor de eficiencia general de ese conjunto de consultas funcionalmente equivalente, y entonces puede comparar los valores de eficiencia generales para todos los conjuntos de consultas funcionalmente equivalentes para determinar que tiene la mayor eficiencia. En otras modalidades, el selector de reglas 1106 puede determinar una mayor eficiencia de otras maneras.

Cabe destacar que, en una modalidad, el diagrama de flujo 1200 puede realizarse en tiempo de compilación o en tiempo de ejecución. Por ejemplo, en tiempo de ejecución, una versión ejecutable del código de programa 106 o código de programa refactorizado 108 (por ejemplo, código máquina 222) puede ejecutarse por un motor de ejecución (por ejemplo, en un sistema operativo, etcétera). De acuerdo con una modalidad del tiempo de ejecución, el mejor conjunto de consultas funcionalmente equivalentes posible puede determinarse de una manera que tenga en cuenta el estado real del sistema. Esto se debe a que la ganancia de eficiencia proporcionada por uno o más conjuntos de consultas funcionalmente equivalentes posibles puede depender, al menos parcialmente, de las condiciones de ejecución en tiempo de ejecución. Por ejemplo, en tiempo de ejecución, la versión ejecutable del código de programa puede ejecutarse, y cuando debe ejecutarse una consulta particular, el código de programa puede realizar una llamada al selector de reglas 1106 (que, por ejemplo, puede implementarse por el motor de

ejecución), para seleccionar el conjunto de consultas funcionalmente equivalentes posible para implementar las condiciones de tiempo de ejecución (por ejemplo, condiciones de red, ancho de banda de procesamiento disponible, potencia de procesamiento disponible, etcétera).

Por ejemplo, en condiciones normales de tiempo de ejecución (por ejemplo, disponibilidad total de la red, etcétera), el conjunto posible A de consultas funcionalmente equivalentes puede ser más eficiente que el conjunto B de consultas funcionalmente equivalentes. Sin embargo, durante un tiempo de ejecución particular, algo no deseado puede interferir con el rendimiento de los componentes de consulta del conjunto A de consultas funcionalmente equivalentes, como una conexión de internet lenta, etcétera. Por consiguiente, en tal condición de tiempo de ejecución (por ejemplo, respuesta de red deficiente), el conjunto B de consultas funcionalmente equivalentes puede permitir una mayor eficiencia, y, por lo tanto, puede seleccionarse por el selector de reglas 1106 para ejecutarse durante ese tiempo de ejecución particular (en lugar del conjunto A de consultas). Por consiguiente, en las modalidades, el selector de reglas 1106 puede configurarse para permitir la selección de un conjunto de consultas funcionalmente equivalentes posible en tiempo de ejecución.

Se observa que las reglas pueden agregarse al conjunto de reglas 216 de cualquier manera, incluso de forma manual o automática. Por ejemplo, un desarrollador puede agregar una regla al conjunto de reglas 216 con base en la experiencia del desarrollador, que incluye el deseo de solucionar un problema de eficiencia particular con las consultas que el desarrollador haya visto o se las que está enterado. En otra modalidad, un mecanismo automático puede generar nuevas reglas, que incluyen un mecanismo automático que incorpora el aprendizaje automático. El aprendizaje automático puede utilizarse para muestrear las consultas equivalentes durante el tiempo de ejecución para una eficiencia relativa, y con base en ello, seleccionar automáticamente la consulta más eficiente para esa situación. Para cada optimización de consulta proporcionada por una regla, el optimizador de consultas 104 puede mantener un análisis detallado del efecto que tiene la optimización de consultas.

III. Ejemplo de modalidades ejemplares de los dispositivos móviles y estacionarios

El dispositivo informático 102, optimizador de consultas 104, compilador 204, aplicación de desarrollo 200, editor de código fuente 202, compilador 204, herramienta de depuración 206, detector de consultas 402, extensor de evaluación perezosa 404, generador de conjuntos de consultas equivalentes 406, detector de consultas extensibles 902, ensamblador de expresiones de consultas 904, determinador de contenido 1104, selector de reglas 1106, diagrama de flujo 300, diagrama de flujo 500, diagrama de flujo 700, diagrama de flujo 800, diagrama de flujo 1000 y diagrama de flujo 1200 pueden implementarse en *hardware*, o *hardware* combinado con *software* y/o *firmware*. Por ejemplo, el optimizador de consultas 104, compilador 204, aplicación de desarrollo 200, editor de código fuente 202, compilador 204, herramienta de depuración 206, detector de consultas 402, extensor de evaluación perezosa 404, generador de conjuntos de consultas equivalentes 406, detector de consultas extensibles 902, ensamblador de expresiones de consulta 904, determinador de contenido 1104, selector de reglas 1106, diagrama de flujo 300, diagrama de flujo 500, diagrama de flujo 700, diagrama de flujo 800, diagrama de flujo 1000 y/o diagrama de flujo 1200 pueden implementarse como código de programa

informático/instrucciones configuradas para ejecutarse en uno o más procesadores y almacenarse en un medio de almacenamiento legible por computadora. Alternativamente, el dispositivo informático 102, optimizador de consultas 104, compilador 204, aplicación de desarrollo 200, editor de código fuente 202, compilador 204, herramienta de depuración 206, detector de consultas 402, extensor de evaluación perezosa 404, generador de conjuntos de consultas equivalentes 406, detector de consultas extensibles 902, ensamblador de expresiones de consulta 904, determinador de contenido 1104, selector de reglas 1106, diagrama de flujo 300, diagrama de flujo 500, diagrama de flujo 700, diagrama de flujo 800, diagrama de flujo 1000 y/o diagrama de flujo 1200 pueden implementarse como circuitería lógica/eléctrica de *hardware*.

Por ejemplo, en una modalidad, uno o más, en cualquier combinación, del optimizador de consultas 104, compilador 204, aplicación de desarrollo 200, editor de código fuente 202, compilador 204, herramienta de depuración 206, detector de consultas 402, extensor de evaluación perezosa 404, generador de conjuntos de consultas equivalentes 406, detector de consultas extensibles 902, ensamblador de expresiones de consulta 904, determinador de contenido 1104, selector de reglas 1106, diagrama de flujo 300, diagrama de flujo 500, diagrama de flujo 700, diagrama de flujo 800, diagrama de flujo 1000 y/o diagrama de flujo 1200 pueden implementarse juntos en un SoC. El SoC puede incluir un chip de circuito integrado que incluye uno o más de un procesador (por ejemplo, una unidad central de procesamiento (CPU, por sus siglas en inglés), microcontrolador, microprocesador, procesador de señal digital (DSP, por sus siglas en inglés), etcétera), memoria, una o más interfaces de comunicación, y/o otros circuitos, y puede ejecutar opcionalmente el código de programa recibido y/o incluir *firmware* incorporado para realizar las funciones.

La figura 13 representa una implementación ejemplar de un dispositivo informático 1300 en el cual pueden implementarse las modalidades. Por ejemplo, el dispositivo informático 102 y/o el dispositivo informático cliente 104 pueden implementarse en uno o más dispositivos informáticos similares al dispositivo informático 1300 en las modalidades de las computadora estacionaria o móvil, que incluyen una o más características del dispositivo informático 1300 y/o características alternativas. La descripción del dispositivo informático 1300 provista en el presente documento se proporciona con fines ilustrativos y no pretende ser limitativa. Las modalidades pueden implementarse en otros tipos de sistemas informáticos, como conocerían los expertos en la técnica relevante.

Tal como se muestra en la figura 13, el dispositivo informático 1300 incluye uno o más procesadores, denominados circuito procesador 1302, memoria de sistema 1304 y bus 1306 que conectan varios componentes de sistema, que incluyen la memoria de sistema 1304 al circuito procesador 1302. El circuito procesador 1302 es un circuito eléctrico y/u óptico implementado en uno o más elementos del dispositivo de circuito eléctrico de *hardware* físico y/o dispositivos de circuito integrado (chips o matrices de material semiconductor) como una unidad central de procesamiento (CPU, por sus siglas en inglés), un microcontrolador, microprocesador y/u otro circuito procesador de *hardware* físico. El circuito del procesador 1302 puede ejecutar el código de programa almacenado en un medio legible por computadora, como el código de programa del sistema operativo 1330, programas de aplicación 1332,

otros programas 1334, etcétera. El bus 1306 representa uno o más de cualquiera de varios tipos de estructuras de bus, que incluyen un bus de memoria o controlador de memoria, bus periférico, puerto de gráficos acelerado y un procesador o bus local que utiliza cualquiera de una variedad de arquitecturas de bus. La memoria de sistema 1304 incluye la memoria exclusivamente de lectura (ROM, por sus siglas en inglés) 1308 y la memoria de acceso aleatorio (RAM, por sus siglas en inglés) 1310. Un sistema básico de entrada/salida (BIOS, por sus siglas en inglés) 1312 se almacena en la ROM 1308.

El dispositivo informático 1300 también tiene una o más de las siguientes unidades: una unidad de disco duro 1314 para leer y escribir en un disco duro, una unidad de disco magnético 1316 para leer o escribir en un disco magnético extraíble 1318 y una unidad de disco óptico 1320 para leer o escribir en un disco óptico extraíble 1322, como un CD ROM, DVD ROM u otro medio óptico. La unidad de disco duro 1314, unidad de disco magnético 1316 y unidad de disco óptico 1320 están conectadas al bus 1306 mediante una interfaz de unidad de disco duro 1324, una interfaz de unidad de disco magnético 1326 y una interfaz de unidad óptica 1328, respectivamente. Las unidades y sus medios asociados legibles por computadora proporcionan el almacenamiento permanente de las instrucciones, estructuras de datos, módulos de programas y otros datos legibles por computadora para la computadora. Aunque se describe un disco duro, un disco magnético extraíble y un disco óptico extraíble, pueden utilizarse otros tipos de medios de almacenamiento legibles por computadora con base en *hardware* para almacenar los datos, como tarjetas de memoria *flash*, discos de video digital, RAM, ROM y otros medios de almacenamiento de *hardware*.

Pueden almacenarse varios módulos de programa en el disco duro, disco magnético, disco óptico, ROM o RAM. Estos programas incluyen el sistema operativo 1330, uno o más programas de aplicación 1332, otros programas 1334 y datos de programa 1336. Los programas de aplicación 1332 u otros programas 1334 pueden incluir, por ejemplo, lógica de programa de computadora (por ejemplo, código o instrucciones de programa de computadora) para implementar el optimizador de consultas 104, compilador 204, aplicación de desarrollo 200, editor de código fuente 202, compilador 204, herramienta de depuración 206, detector de consultas 402, extensor de evaluación perezosa 404, generador de conjuntos de consultas equivalentes 402, detector de consultas extensibles 902, ensamblador de expresiones de consulta 904, determinador de contenido 1104, selector de reglas 1106, diagrama de flujo 300, diagrama de flujo 500, diagrama de flujo 700, diagrama de flujo 800, diagrama de flujo 1000 y/o diagrama de flujo 1200 (que incluyen cualquier etapa adecuada de los diagramas de flujo 300, 500, 700, 800, 1000, 1200) y/u otras modalidades descritas en el presente documento.

Un usuario puede introducir comandos e información en el dispositivo informático 1300 a través de dispositivos de entrada como el teclado 1338 y el dispositivo señalador 1340. Otros dispositivos de entrada (no mostrados) pueden incluir un micrófono, palanca de mando, consola de juegos, antena parabólica, escáner, una pantalla táctil y/o panel táctil, un sistema de reconocimiento de voz para recibir la entrada de voz, un sistema de reconocimiento de gestos para recibir la entrada de gestos o similares. Estos y otros dispositivos de entrada a menudo están conectados al circuito procesador 1302 a través de una interfaz de puerto serial 1342 que está conectada al bus 1306, pero pueden estar conectados por

otras interfaces, como un puerto paralelo, un puerto de juegos o un bus serial universal (USB, por sus siglas en inglés).

Una pantalla de visualización 1344 también está conectada al bus 1306 a través de una interfaz, como un adaptador de video 1346. La pantalla de visualización 1344 puede ser externa o incorporada en el dispositivo informático 1300. La pantalla de visualización 1344 puede mostrar la información, además de una interfaz de usuario para recibir comandos de usuario y/u otra información (por ejemplo, mediante toques, gestos con los dedos, teclado virtual, etcétera). Además de la pantalla de visualización 1344, el dispositivo informático 1300 puede incluir otros dispositivos de salida periféricos (no mostrados) tales como altavoces e impresoras.

El dispositivo informático 1300 está conectado a una red 1348 (por ejemplo, internet) a través de un adaptador o interfaz de red 1350, un módem 1352 u otro medio para establecer comunicaciones a través de la red. El módem 1352, que puede ser interno o externo, puede conectarse al bus 1306 a través de la interfaz de puerto serial 1342, tal como se muestra en la figura 13, o puede conectarse al bus 1306 utilizando otro tipo de interfaz, que incluye una interfaz paralela.

Tal como se utiliza en el presente documento, los términos “medio de programa informático”, “medio legible por computadora” y “medio de almacenamiento legible por computadora” se utilizan para referirse a medios físicos de *hardware* como el disco duro asociado con la unidad de disco duro 1314, disco magnético extraíble 1318, disco óptico extraíble 1322, otros medios de *hardware* físico como RAM, ROM, tarjetas de memoria *flash*, discos de video digital, discos zip, MEM, dispositivos de almacenamiento con base en nanotecnología y otros tipos de medios de almacenamiento de *hardware* físicos/tangibles (que incluyen la memoria 1320 de la figura 13). Tales medios de almacenamiento legibles por computadora se distinguen y no se superponen con los medios de comunicación (no incluyen los medios de comunicación). Los medios de comunicación incorporan instrucciones legibles por computadora, estructuras de datos, módulos de programas u otros datos en una señal de datos modulada, como una onda portadora. El término “señal de datos modulados” significa una señal que tiene una o más de sus características configuradas o modificadas de tal manera que codifiquen la información en la señal. A modo ejemplar, y no de limitación, los medios de comunicación incluyen medios inalámbricos como medios acústicos, RF, infrarrojos y otros medios inalámbricos, así como medios alámbricos. Las modalidades también se dirigen a tales medios de comunicación que están separados y no se superponen con las modalidades dirigidas a los medios de almacenamiento legibles por computadora.

Tal como se indicó anteriormente, los programas y módulos informáticos (que incluyen los programas de aplicación 1332 y otros programas 1334) pueden almacenarse en el disco duro, disco magnético, disco óptico, ROM, RAM u otro medio de almacenamiento de *hardware*. Tales programas informáticos también pueden recibirse a través de la interfaz de red 1350, la interfaz de puerto serial 1342 o cualquier otro tipo de interfaz. Tales programas informáticos, cuando se ejecutan o cargan por una aplicación, permiten que el dispositivo informático 1300 implemente las características de las modalidades descritas en el presente documento. En consecuencia, tales programas informáticos representan los controladores del dispositivo informático 1300.

Las modalidades también se dirigen a productos de programas informáticos que comprenden un código informático o instrucciones almacenadas en cualquier medio legible por computadora. Tales productos de productos informáticos incluyen unidades de disco duro, unidades de disco óptico, paquetes de dispositivos de memoria, lápices de memoria portátiles, tarjetas de memoria y otros tipos de *hardware* de almacenamiento físico.

IV. Modalidades ejemplares

En una modalidad, un método comprende: detectar una pluralidad de consultas en el código de programa; extendiendo la evaluación perezosa mediante la cual las consultas son evaluadas en el código de programa; y aplicando un conjunto de reglas que incluye una pluralidad de reglas a las consultas detectadas para generar un conjunto de consultas funcionalmente equivalentes que evalúa de manera más eficiente con relación a las consultas detectadas.

En una modalidad, la detección comprende: detectar las consultas en el código de programa en un editor de código; y el método comprende, además: presentar una opción para que el editor de código refactorice automáticamente el código de programa para reemplazar la pluralidad de consultas con el conjunto de consultas funcionalmente equivalentes.

En una modalidad, la detección comprende: detectar las consultas en el código de programa durante la compilación; y el método comprende, además: generar el código compilado en el compilador con base en una versión del código de programa donde la pluralidad de consultas se reemplaza con el conjunto de consultas funcionalmente equivalentes.

En una modalidad, la evaluación perezosa extendida mediante la cual las consultas se evalúan en el código de programa, comprende: detectar una o más de las consultas que son extensibles a la evaluación; y formar una única expresión de consulta que incluye una o más consultas extensibles a la evaluación.

En una modalidad, la aplicación de un conjunto de reglas comprende: determinar un contexto lógico común entre múltiples componentes de consulta; y aplicar una regla del conjunto de reglas correspondiente al contexto lógico común a los múltiples componentes de consulta.

En una modalidad, el conjunto de consultas funcionalmente equivalentes generado evalúa de manera más eficiente con relación a la pluralidad de consultas mediante al menos uno de generar los resultados de consultas que consumen menos espacio de memoria que los resultados de consultas de la pluralidad de consultas, tardar menos tiempo para la ejecución que pluralidad de consultas, consumir menos ancho de banda que la pluralidad de consultas, o consumir menos potencia de procesamiento que la pluralidad de consultas.

En una modalidad, la aplicación de un conjunto de reglas comprende: evaluar la aplicación de una pluralidad de combinaciones de reglas del conjunto de reglas a los componentes de consulta para generar una pluralidad de conjunto de consultas funcionalmente equivalentes posible; y seleccionar un conjunto de consultas funcionalmente equivalentes posible de la pluralidad de conjuntos de consultas funcionalmente equivalentes posibles que tienen una mayor ganancia de eficiencia para que sea el conjunto de consultas funcionalmente equivalentes generado.

En una modalidad, la aplicación de un conjunto de reglas comprende: evaluar durante el tiempo de ejecución la aplicación de una pluralidad de combinaciones de reglas del conjunto de reglas a los componentes de consulta para generar una pluralidad de conjuntos de consultas funcionalmente equivalentes posibles; y seleccionar durante el tiempo de ejecución un conjunto de consultas funcionalmente equivalentes posible de la pluralidad de conjuntos de consultas funcionalmente equivalentes posibles que tienen una mayor ganancia de eficiencia para que sea el conjunto de consultas funcionalmente equivalentes generado.

En una modalidad, la ganancia de eficiencia proporcionada por uno o más de los conjuntos de consultas funcionalmente equivalentes posibles dependen, al menos parcialmente, de las condiciones de ejecución en el tiempo de ejecución.

En una modalidad, el método comprende, además: utilizar el aprendizaje automático para generar al menos una regla para agregar el conjunto de reglas.

En otra modalidad, un dispositivo informático comprende: al menos un circuito procesador; y al menos una memoria que almacena el código de programa configurado que se ejecutará por al menos un circuito procesador, el código de programa está configurado para realizar las operaciones que comprenden: detectar una pluralidad de consultas en el código de programa; extender la evaluación perezosa mediante la cual las consultas son evaluadas en el código de programa; y aplicar un conjunto de reglas, que incluye una pluralidad de reglas, a las consultas detectadas para generar un conjunto de consultas funcionalmente equivalente que evalúe de manera más eficiente con relación a las consultas detectadas.

En una modalidad, la detección comprende: detectar las consultas en el código de programa en un editor de código; y el método comprende, además: presentar una opción para que el editor de código refactorice automáticamente el código de programa para reemplazar la pluralidad de consultas con el conjunto de consultas funcionalmente equivalentes.

En una modalidad, la detección comprende: detectar las consultas en el código de programa durante la compilación; y el método comprende, además: generar código compilado en el compilador con base en una versión del código de programa donde la pluralidad de consultas se reemplaza con el conjunto de consultas funcionalmente equivalentes.

En una modalidad, la evaluación perezosa extendida mediante la cual las consultas se evalúan en el código de programa, comprende: detectar una o más de las consultas que son extensibles a la evaluación; y formar una única expresión de consulta que incluya una o más consultas extensibles a la evaluación.

En una modalidad, la aplicación de un conjunto de reglas comprende: determinar un contexto lógico común entre los múltiples componentes de consulta; y aplicar una regla del conjunto de reglas correspondiente al contexto lógico común a los múltiples componentes de consulta.

En una modalidad, la aplicación de un conjunto de reglas comprende: evaluar la aplicación de una pluralidad de combinaciones de reglas del conjunto de reglas a los componentes de consulta para generar una pluralidad de conjuntos de consultas funcionalmente equivalentes posibles; y seleccionar un

conjunto de consultas funcionalmente equivalentes posible de la pluralidad de conjuntos de consultas funcionalmente equivalentes posibles que tienen una mayor ganancia de eficiencia para que sea el conjunto de consultas funcionalmente equivalentes generado.

En una modalidad, la aplicación de un conjunto de reglas comprende: evaluar durante el tiempo de ejecución la aplicación de una pluralidad de combinaciones de reglas del conjunto de reglas a los componentes de consulta para generar una pluralidad de conjuntos de consultas funcionalmente equivalentes posibles; y seleccionar durante el tiempo de ejecución un conjunto de consultas funcionalmente equivalentes posible de la pluralidad de conjuntos de consultas funcionalmente equivalentes posibles que tenga una mayor ganancia de eficiencia para que sea el conjunto de consultas funcionalmente equivalentes generado.

En una modalidad, la ganancia de eficiencia proporcionada por uno o más de los conjuntos de consultas funcionalmente equivalentes posibles depende, al menos parcialmente, de las condiciones de ejecución en el tiempo de ejecución.

En una modalidad, las operaciones comprenden, además: utilizar el aprendizaje automático para generar al menos una regla para agregar al conjunto de reglas.

En otra modalidad, un dispositivo informático comprende: al menos un circuito procesador; y al menos una memoria que almacena el código de programa configurado que se ejecutará por al menos un circuito procesador, el código de programa comprende: un detector de consultas configurado para detectar una pluralidad de consultas en el código de programa; un extensor de evaluación perezosa configurado para extender la evaluación perezosa mediante el cual las consultas se evalúan en el código de programa; y un generador de conjuntos de consultas equivalentes configurado para aplicar un conjunto de reglas, que incluye una pluralidad de reglas, a las consultas detectadas para generar un conjunto de consultas funcionalmente equivalentes que evalúe de manera más eficiente con relación a las consultas detectadas.

V. Conclusión

Aunque varias modalidades de la presente invención se han descrito anteriormente, debe entenderse que se han presentado únicamente a modo ejemplar, y no de limitación. El experto en la técnica relevante entenderá que pueden realizarse diversos cambios en la forma y detalles sin apartarse de la esencia y alcance de la invención tal como se define en las reivindicaciones adjuntas. Por consiguiente, la extensión y alcance de la presente invención no deberían estar limitados por ninguna de las modalidades ejemplares descritas anteriormente, sino que deberían definirse únicamente de acuerdo con las siguientes reivindicaciones y sus equivalentes.