(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 804 506**

(51) Int. CI.:

G06F 9/445 (2008.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **24.10.2012 E 12189820 (9)**

(97) Fecha y número de publicación de la concesión europea: **29.04.2020 EP 2587372**

(54) Título: **Compartición de objetos de primera clase a través de múltiples lenguajes de programación interpretados**

(30) Prioridad:

24.10.2011 US 201113279748

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
08.02.2021

(73) Titular/es:

THE BOEING COMPANY (100.0%)
100 North Riverside Plaza
Chicago, IL 60606-1596, US

(72) Inventor/es:

THUNEMANN, PAUL Z. y
RAY, STEPHEN L

(74) Agente/Representante:

CARVAJAL Y URQUIJO, Isabel

DESCRIPCIÓN

Compartición de objetos de primera clase a través de múltiples lenguajes de programación interpretados

Antecedentes

La presente divulgación se refiere en general a interfaces que permiten el uso en un lenguaje de programación interpretado de construcciones de lenguaje escritos en un lenguaje de programación interpretado diferente. En particular, la presente divulgación se refiere a interfaces que permitirían al usuario de una secuencia de comandos de alto nivel integrar secuencias de comandos de menor nivel escritos en diferentes lenguajes de secuencias de comandos.

Los lenguajes compilados no se pueden usar de forma interactiva. El caso de uso estándar es escribir el código, compilar el código y, después, ejecutar el código. Este es un buen paradigma para un desarrollador de software que está escribiendo una aplicación, pero no es tan útil para un ingeniero o matemático que necesita explorar y resolver un problema.

Los lenguajes interpretados se pueden usar de forma interactiva. Un ingeniero puede tener un "aviso" en el que escribe un código y el código se interpreta y luego se ejecuta mientras se ejecuta la aplicación subyacente. Los ejemplos de lenguajes interpretados incluyen Python, Ruby, etc.

Se añaden lenguajes de secuencias de comandos a las aplicaciones para permitir que un usuario maneje la aplicación desde una señal de comando o un archivo de texto. Los lenguajes interpretados se pueden usar como lenguajes de secuencias de comandos y, en adelante, la presente divulgación hará referencia a los lenguajes interpretados que se usan para las secuencias de comandos como "lenguajes de secuencias de comandos".

Python es un lenguaje interpretado. Tiene varias implementaciones, incluidos CPython y Jython. CPython está escrito en lenguaje C y comúnmente se llama Python. Jython está escrito en Java.

Java es un lenguaje compilado que se ejecuta en una máquina virtual. Hay varios lenguajes de secuencias de comandos que se crean sobre la máquina virtual Java. Estos se conocen comúnmente como lenguajes de secuencias de comandos basados en J. (Los ejemplos incluyen Jython, JRuby, Groovy y Rhino).

Se conoce un lenguaje de secuencias de comandos basado en J patentado que es un lenguaje específico de dominio diseñado para la manipulación de geometría y mallado (creación de representaciones discretas). También se conoce una aplicación escrita en Java para construcción geométrica, análisis y manipulación, que está diseñada para secuencias de comandos. El código Java subyacente define muchas operaciones de geometría. Los ingenieros pueden usar estas secuencias de comandos para hacer uso de operaciones de geometría específicas para resolver su problema de ingeniería. Estas secuencias de comandos representan la captura de conocimiento del proceso de ingeniería. Hay una gran necesidad de reutilizar estas secuencias de comandos. Existe la necesidad de poder controlar la aplicación de construcción geométrica mencionada anteriormente utilizando estas secuencias de comando del lenguaje patentado preexistentes.

Cabe considerar un proceso de ingeniería de alto nivel que consiste en varios procesos de ingeniería de nivel inferior. Es posible que cada proceso de nivel inferior ya haya sido escrito. Un ingeniero debe poder escribir una secuencia de comandos de alto nivel que integre las secuencias de comandos de nivel inferior. Dado que estas secuencias de comandos pueden estar escritas en más de un lenguaje de secuencias de comandos, se necesita una forma de compartir fácilmente las funciones, clases, instancias y objetos que se crearon en los diferentes lenguajes de secuencias de comandos.

Actualmente es posible en entornos de secuencias de comandos existentes compartir datos entre lenguajes de secuencias de comandos, pero para hacerlo se requiere habilidades de programación y conocimiento del lenguaje Java subyacente. Una solución a este problema debe estar diseñada para ingenieros, no para desarrolladores de aplicaciones, por lo que la solución no debería requerir programación/desarrollo de software adicional por parte del ingeniero.

Orfali R et al: "Instant Corba, Passage", 1 de enero de 1997 (1997-01-01), Wiley Computer Publisher, Canadá, ISBN: 0-471-18333-4, páginas 2-28, describe la Arquitectura Común de Agente de Solicitud de Objetos (CORBA), relacionados con el bus de objetos CORBA y los servicios del sistema de objetos que extienden el bus.

Por lo tanto, existe la necesidad de una metodología que permita a diferentes ingenieros programar en diferentes lenguajes de secuencias de comandos y luego compartir el código que escriben en estos lenguajes de secuencias de comandos sin considerar los problemas de las implementaciones de lenguaje subyacentes. Preferentemente, la solución también incluirá lenguajes específicos de dominio.

Sumario

Según un aspecto, se proporciona un método para compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos como se define en la reivindicación 1. De acuerdo con otro aspecto, se proporciona un sistema que permite compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos como se define en la reivindicación 10.

Se desvelan sistemas y métodos para permitir a los usuarios escribir códigos de secuencias de comandos en un primer lenguaje de secuencias de comandos basado en Java, tal como Jython, JRuby y Matlab, y luego usar un segundo lenguaje de secuencias de comandos basado en Java para invocar construcciones de lenguaje escritas en ese primer lenguaje de secuencias de comandos. Las construcciones del lenguaje incluyen, por ejemplo, listas, conjuntos, mapas, funciones, definiciones de clase, instancias de clase y módulos de código. Por ejemplo, la metodología desvelada permite a un ingeniero escribir una función en un lenguaje de secuencias de comandos y llamarla desde otro lenguaje de secuencias de comandos. Las técnicas desveladas en el presente documento también son aplicables a lenguajes específicos de dominio.

De acuerdo con las realizaciones desveladas a continuación, funciones, definiciones de clase, instancias de clase, Los módulos y otras construcciones de lenguaje se tratan como objetos de primera clase que se pueden compartir entre los diferentes lenguajes de secuencias de comandos. En informática, un objeto de primera clase es una entidad que se puede construir en el tiempo de ejecución, pasado como un parámetro o argumento, devuelto desde una subrutina o asignado a una variable. Como parte de la metodología desvelada en el presente documento, una representación subyacente respectiva de cada uno de estos tipos de objetos está diseñada como una interfaz Java y, a continuación, dicha interfaz se implementa en cada lenguaje de secuencias de comandos. Además, el código está escrito en cada implementación de lenguaje de secuencias de comandos para permitir que este último use la interfaz Java para representar una Función, Clase u otra construcción de lenguaje.

La metodología desvelada en el presente documento permite que diferentes lenguajes de secuencias de comandos interactúen entre sí de una manera que sea natural para el lenguaje dado, de modo que el ingeniero que usa ese lenguaje de secuencias de comandos no necesita tener en cuenta ni tener experiencia o conocimiento de otros lenguajes de secuencias de comandos cuando escribe sus secuencias de comandos.

De manera más específica, un ejemplo de la invención es un método para compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos, que comprende: (a) definir una interfaz del núcleo de programación de aplicaciones del núcleo que sea independiente del lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos, la interfaz del núcleo de programación de aplicaciones del núcleo que comprende una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje; (b) llamar a una construcción de lenguaje de un tipo escrito en un primer lenguaje de secuencias de comandos, estando la llamada realizada por una construcción de lenguaje del tipo escrito en un segundo lenguaje de secuencias de comandos; (c) crear una instancia de una interfaz del núcleo que maneje construcciones de lenguaje del tipo; (d) redirigir la llamada a la instancia de la interfaz del núcleo; y (e) redirigir la llamada recibida por la instancia de la interfaz del núcleo a la construcción de lenguaje del tipo escrito en el primer lenguaje de secuencias de comandos.

Otro ejemplo de la invención es un sistema que permite compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos, comprendiendo el sistema una interfaz de programación de aplicaciones del núcleo que es independiente del lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos, comprendiendo la interfaz de programación aplicaciones del núcleo una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje y un procesador programado para ejecutar las operaciones (b) a (e) establecidas en el párrafo anterior en respuesta a un comando del usuario.

Otro ejemplo de la invención es un método para hacer una llamada a una construcción de lenguaje, en el que la llamada se escribe en un primer lenguaje de secuencias de comandos mientras que la construcción del lenguaje se escribe en un segundo lenguaje de secuencias de comandos, que comprende: (a) definir una interfaz de programación de aplicaciones del núcleo que sea neutral en cuanto al lenguaje en relación con el primer y segundo lenguajes de secuencias de comandos, la interfaz del núcleo de programación de aplicaciones del núcleo que comprende una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje; (b) crear el primer y el segundo objeto de interfaz del núcleo; (c) crear un primer objeto adaptador del lenguaje que redirige la llamada al primer objeto de interfaz del núcleo; (d) crear un objeto de construcción de lenguaje escrito en el segundo lenguaje de secuencia de comandos; (e) crear un primer objeto adaptador del núcleo que redirige la llamada recibida por el primer objeto de interfaz del núcleo al objeto de construcción del lenguaje; (f) crear un segundo objeto adaptador de lenguaje que devuelve al segundo objeto de interfaz del núcleo un resultado producido por el objeto de construcción de lenguaje; y (g) crear un segundo objeto adaptador del núcleo que redirige los resultados devueltos recibidos por el segundo objeto de interfaz del núcleo a un objeto de construcción de lenguaje escrito en el primer lenguaje de secuencias de comandos.

Otro ejemplo más de la invención es un sistema para hacer una llamada a una construcción de lenguaje, en el que la llamada se escribe en un primer lenguaje de secuencias de comandos mientras que la construcción del lenguaje se escribe en un segundo lenguaje de secuencias de comandos, comprendiendo el sistema una interfaz de programación de aplicaciones del núcleo que es neutral en cuanto al lenguaje en relación con el primer y segundo lenguaje de secuencias de comandos, comprendiendo la interfaz de programación de aplicaciones del núcleo una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje y un procesador programado para ejecutar las operaciones (b) a (g) establecidas en el párrafo anterior en respuesta a un comando del usuario.

La invención implica un método para compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos que comprende: definir una interfaz de programación de aplicaciones del núcleo que sea neutral en cuanto al lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos, comprendiendo la interfaz de programación de aplicaciones del núcleo que comprende una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje; llamar a una construcción de lenguaje de un primer tipo escrito en un primer lenguaje de secuencias de comando, estando la llamada realizada por una construcción de lenguaje de dicho primer tipo escrito en un segundo lenguaje de secuencias de comandos; crear una primera instancia de una primera interfaz del núcleo que maneja construcciones de lenguaje de dicho primer tipo; redirigir la llamada a dicha primera instancia de dicha primera interfaz del núcleo; y redirigir la llamada recibida por dicha primera instancia de dicha primera interfaz del núcleo a dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos. El método puede incluir la creación de una instancia de un adaptador de primer lenguaje; y crear una instancia de un primer adaptador del núcleo, en el que la operación (d) se implementa mediante dicha instancia de dicho primer adaptador de lenguaje y la operación (e) se implementa mediante dicha instancia de dicho primer adaptador del núcleo.

El método también puede incluir la creación de una instancia de dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos en respuesta a la llamada; devolver un resultado de dicha instancia de dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos; crear una segunda instancia de dicha primera interfaz del núcleo; redirigir el resultado devuelto a dicha segunda instancia de dicha primera interfaz del núcleo; y redirigir el resultado devuelto recibido por dicha segunda instancia de dicha primera interfaz del núcleo a dicha construcción de lenguaje de dicho primer tipo escrito en dicho segundo lenguaje de secuencia de comandos. El método puede comprender crear una instancia de un segundo adaptador de lenguaje; y crear una instancia de un segundo adaptador del núcleo, en el que la operación (i) se implementa mediante dicha instancia de dicho segundo adaptador de lenguaje y la operación (j) se implementa mediante dicha instancia de dicho segundo adaptador del núcleo.

El primer tipo de construcción de lenguaje puede seleccionarse del siguiente conjunto de tipos: primitivos, cadenas, listas, diccionarios, conjuntos, tuplas, matrices, funciones, definiciones de clase, instancias de clase, métodos y módulos. El primer y segundo lenguaje de secuencias de comandos pueden estar basados en J. El método puede comprender llamar a una construcción de lenguaje de un segundo tipo escrito en un tercer lenguaje de secuencias de comandos, estando la llamada realizada por una construcción de lenguaje de dicho primer tipo escrito en un dicho segundo lenguaje de secuencias de comandos; crear una primera instancia de una segunda interfaz del núcleo que maneja construcciones de lenguaje de dicho segundo tipo; redirigir la llamada a dicha primera instancia de dicha segunda interfaz del núcleo; y redirigir la llamada recibida por dicha primera instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencia de comandos.

El método puede incluir crear una instancia de un tercer adaptador de lenguaje; y crear una instancia de un tercer adaptador del núcleo, en el que la operación (m) se implementa mediante dicha instancia de dicho tercer adaptador del lenguaje y la operación (n) se implementa mediante dicha instancia de dicho tercer adaptador del núcleo. Además, el método puede comprender además crear una instancia de dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencias de comandos en respuesta a la llamada; devolver un resultado de dicha instancia de dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencia de comandos; crear una segunda instancia de dicha segunda interfaz del núcleo; redirigir el resultado devuelto a dicha segunda instancia de dicha segunda interfaz del núcleo; y redirigir el resultado devuelto recibido por dicha segunda instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje de dicho segundo tipo escrito en dicho segundo lenguaje de secuencia de comandos. También puede implicar crear una instancia de un cuarto adaptador de lenguaje; y crear una instancia de un cuarto adaptador del núcleo, en el que la operación (r) se implementa mediante dicha instancia de dicho cuarto adaptador de lenguaje y la o las operaciones se implementan mediante dicha instancia de dicho cuarto adaptador del núcleo.

Un sistema puede permitir que las construcciones de lenguaje se compartan entre diferentes lenguajes de secuencias de comandos, comprendiendo dicho sistema una interfaz de programación de aplicaciones del núcleo que es independiente del lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos, dicha interfaz del núcleo de programación de aplicaciones comprende una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje y un procesador programado para ejecutar las siguientes

operaciones en respuesta a un comando del usuario: llamar a una construcción de lenguaje de un primer tipo escrito en un primer lenguaje de secuencias de comandos, estando la llamada realizada por una construcción de lenguaje de dicho primer tipo escrito en un segundo lenguaje de secuencias de comandos; crear una primera instancia de una primera interfaz del núcleo que maneja construcciones de lenguaje de dicho primer tipo; redirigir la llamada a dicha primera instancia de dicha primera interfaz del núcleo; y redirigir la llamada recibida por dicha primera instancia de dicha primera interfaz del núcleo a dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos. El sistema puede incluir un procesador que está programado para ejecutar las siguientes operaciones: crear una instancia de un primer adaptador de lenguaje; y crear una instancia de un primer adaptador del núcleo, en el que la operación (c) se implementa mediante dicha instancia de dicho primer adaptador de lenguaje y la operación (d) se implementa mediante dicha instancia de dicho primer adaptador del núcleo. Opcionalmente, el sistema en el que dicho procesador se programa adicionalmente para ejecutar las siguientes operaciones: crear una instancia de dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos en respuesta a la llamada; devolver un resultado de dicha instancia de dicha construcción de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos; crear una segunda instancia de dicha primera interfaz del núcleo; redirigir el resultado devuelto a dicha segunda instancia de dicha primera interfaz del núcleo; y redirigir el resultado devuelto recibido por dicha segunda instancia de dicha primera interfaz del núcleo a dicha construcción de lenguaje de dicho primer tipo escrito en dicho segundo lenguaje de secuencia de comandos.

El sistema en el que dicho procesador está además programado para ejecutar las siguientes operaciones: crear una instancia de un segundo adaptador de lenguaje; y crear una instancia de un segundo adaptador del núcleo, en el que la operación (h) se implementa mediante dicha instancia de dicho segundo adaptador de lenguaje y la operación (i) se implementa mediante dicha instancia de dicho segundo adaptador del núcleo. El sistema puede incluir un procesador que está programado para ejecutar las siguientes operaciones: llamar a una construcción de lenguaje de un segundo tipo escrito en un tercer lenguaje de secuencias de datos, estando la llamada realizada por una construcción de lenguaje de dicho primer tipo escrito en un dicho segundo lenguaje de secuencias de comandos; crear una primera instancia de una segunda interfaz del núcleo que maneja construcciones de lenguaje de dicho segundo tipo; redirigir la llamada a dicha primera instancia de dicha segunda interfaz del núcleo; y redirigir la llamada recibida por dicha primera instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencia de comandos. El procesador del sistema puede programarse para ejecutar las siguientes operaciones: crear una instancia de un tercer adaptador de lenguaje; y crear una instancia de un tercer adaptador del núcleo, en el que la operación (l) se implementa mediante dicha instancia de dicho tercer adaptador del lenguaje y la operación (m) se implementa mediante dicha instancia de dicho tercer adaptador del núcleo.

El procesador del sistema se puede programar adicionalmente para ejecutar las siguientes operaciones: crear una instancia de dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencias de comandos en respuesta a la llamada; devolver un resultado de dicha instancia de dicha construcción de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencia de comandos; crear una segunda instancia de dicha segunda interfaz del núcleo; redirigir el resultado devuelto a dicha segunda instancia de dicha segunda interfaz del núcleo; y redirigir el resultado devuelto recibido por dicha segunda instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje de dicho segundo tipo escrito en dicho segundo lenguaje de secuencia de comandos. El procesador del sistema también se puede programar para ejecutar las siguientes operaciones: crear una instancia de un cuatro adaptador de lenguaje; y crear una instancia de un cuarto adaptador del núcleo, en el que la operación (r) se implementa mediante dicha instancia de dicho cuarto adaptador de lenguaje y la o las operaciones se implementan mediante dicha instancia de dicho cuarto adaptador del núcleo.

La invención implica un método para realizar una llamada a una construcción de lenguaje, en el que la llamada se escribe en un primer lenguaje de secuencias de comandos mientras que la construcción del lenguaje se escribe en un segundo lenguaje de secuencias de comandos, que comprende: definir una interfaz de programación de aplicaciones del núcleo que es neutral en cuanto al lenguaje en relación con dichos primer y segundo lenguaje de secuencias de comandos, comprendiendo la interfaz de programación de aplicaciones del núcleo que comprende una interfaz del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje; crear dichos primer y segundo objetos de interfaz del núcleo; crear un primer objeto adaptador de lenguaje que redirige la llamada a dicho primer objeto de interfaz del núcleo; crear un objeto de construcción de lenguaje escrito en el segundo lenguaje de secuencias de comandos; crear un primer objeto adaptador del núcleo que redirige la llamada recibida por dicho primer objeto de interfaz del núcleo a dicho objeto de construcción de lenguaje; crear un segundo objeto adaptador de lenguaje que devuelve a dicho segundo objeto de interfaz del núcleo un resultado producido por dicho objeto de construcción de lenguaje; y crear un segundo objeto adaptador del núcleo que redirige los resultados devueltos recibidos por dicho segundo objeto de interfaz del núcleo a un objeto de construcción de lenguaje escrito en dicho primer lenguaje de secuencias de comandos.

Un sistema para realizar una llamada a una construcción de lenguaje se escribe en un primer lenguaje de secuencia de comandos mientras que la construcción de lenguaje se escribe en un segundo lenguaje de secuencia de comandos, comprendiendo dicho sistema una interfaz de programación de aplicaciones del núcleo que es neutral en

cuanto al lenguaje en relación con dicho primer y segundo lenguajes de secuencias de comandos, comprendiendo dicha programación de aplicaciones del núcleo una interfaz del núcleo respectiva para cada una de una pluralidad de tipos de construcciones de lenguaje y un procesador programado para ejecutar las siguientes operaciones en respuesta a un comando del usuario: crear el primero y segundo objeto de interfaz del núcleo; crear un primer objeto adaptador de lenguaje que redirige la llamada a dicho primer objeto de interfaz del núcleo; crear un objeto de construcción de lenguaje escrito en el segundo lenguaje de secuencias de comandos; crear un primer objeto adaptador del núcleo que redirige la llamada recibida por dicho primer objeto de interfaz del núcleo a dicho objeto de construcción de lenguaje; crear un segundo objeto adaptador de lenguaje que devuelve a dicho segundo objeto de interfaz del núcleo un resultado producido por dicho objeto de construcción de lenguaje; y crear un segundo objeto adaptador del núcleo que redirige los resultados devueltos recibidos por dicho segundo objeto de interfaz del núcleo a un objeto de construcción de lenguaje escrito en dicho primer lenguaje de secuencias de comandos.

Otros aspectos de la invención se desvelan y reivindican a continuación.

Breve descripción de los dibujos

La figura 1 es un diagrama que muestra un patrón de adaptador para usar cuando un usuario de Python llama a una función (denominada DSLFunction) escrita en un lenguaje específico de dominio por medio de una API Core. La figura 2 es un diagrama que representa el código orientado a objetos para facilitar la compartición de construcciones de lenguaje entre una pluralidad de lenguajes de secuencias de comandos de acuerdo con una realización.

A continuación se hará referencia a los dibujos en los que elementos similares en dibujos diferentes tienen los mismos números de referencia.

Descripción detallada

En la divulgación detallada que sigue, los términos la interfaz, definición de clase, instancia de clase, interfaz de programación de aplicaciones y adaptador tendrán los siguientes significados:

Interfaz: Un conjunto de métodos relacionados (solo las firmas de los métodos, no las implementaciones).

Definición de clase: Un conjunto de métodos implementados juntos. Una definición de clase puede implementar una o más interfaces. La implementación a menudo define algunos datos que se encapsulan y actúan con los métodos implementados.

Instancia de clase: Una definición de clase se puede instanciar varias veces. Cada instanciación de la definición de clase se llama una instancia de clase.

Interfaz de programación de aplicaciones (API): Un conjunto de interfaces y definiciones de clase con un conjunto de comportamientos asociados con dichas interfaces y dichas clases.

Adaptador. Un patrón de diseño que permite que una API existente use una definición de clase existente que no sea parte de la API. Esto se hace habitualmente cuando la definición de la clase no es lo mismo que alguna interfaz o clase que está en la API. El adaptador es una nueva definición de clase que se ajusta a la API existente. La definición de clase existente (que no está en la API) se llama adaptado. El adaptador contiene el adaptado e implementa una interfaz en la API existente (o extiende una clase en la API existente) haciendo uso del adaptado.

Para comprender la metodología desvelada a continuación, es útil observar cómo dos secuencias de comandos interactúan entre sí cuando se escriben en el mismo lenguaje. Primero, se debe considerar una secuencia de comandos simple escrita en el lenguaje Python que define una función que añade dos números y devuelve el resultado. A continuación, hay que considerar una segunda secuencia de comandos que use la primera secuencia de comandos para realizar la operación de adición.

A continuación se expone el código para la primera secuencia de comandos; está escrito en un archivo llamado pyExample.py:

```
---------------start example.py---------------
def add(a, b):
   return a + b
   ---------------end example.py---------------
```

Y a continuación se expone el código de una segunda secuencia de comandos que usa la primera secuencia de comandos:

```
---------------start useExample.py---------------
from example import add
print'3 + 4 = add(3, 4)
---------------end useExample.py---------------
```

El resultado de ejecutar esta secuencia de comandos es que imprimirá: 3 + 4 = 7

La presente divulgación adopta el lenguaje Python y en adelante se referirá a cada secuencias de comandos escrita en su propio archivo como un "módulo". La declaración de importación de Python permite que un módulo acceda a una función escrita en otro módulo. Por lo que, en este caso, el módulo useExample.py importa el módulo example.py y obtiene la función llamada "add" de ese módulo. A continuación, hace uso de dicha función sumando los números 3 y 4. Tenga en cuenta que la declaración de importación está buscando "add" que está en "example" y, para hacerlo, busca en los directorios de la ruta de Python archivos que tengan el nombre "example.py" (y .pyo, .pyc, etc.), se ejecuta "ejemplo.py", lo que da como resultado la creación de un objeto de módulo y, después, busca en el módulo algo llamado "add".

Sería deseable introducir una funcionalidad adicional al lenguaje Jython que le permita al usuario reemplazar el archivo example.py con algún otro archivo escrito en un lenguaje de secuencias de comandos diferentes. A continuación se expone un ejemplo de un archivo escrito en un lenguaje específico de dominio:

```
---------------start example.DSL---------------
function add(a as double, b as double)
   return a + b
   end function
   ---------------end example.DSL---------------
```

El objetivo es poder poner example.DSL en la ruta de Python en lugar de example.py y luego hacer que el módulo de secuencias de comandos useExample.py funcione sin modificaciones. La metodología desvelada a continuación permite que eso suceda.

A continuación se expone una lista de construcciones de lenguaje que son típicas en la mayoría de los lenguajes de procedimiento:

Tipos de datos:

- Primitivos (incluyendo entero, doble, carácter, número entero corto, de punto flotante, byte y construcciones específicas del lenguaje)
- Cadenas
- Listas
- Diccionarios (a veces llamados mapas)
- Conjuntos (comúnmente implementados como las claves de un diccionario, solo disponibles en algunos lenguajes)
- Tuplas (una lista fija, solo algunos lenguajes)
- Matrices

Tipos de lenguaje:

- Funciones
- Definiciones de clase
- Instancias de clase
- Métodos (es decir, métodos de una instancia de clase)
- Módulos

La metodología desvelada en el presente documento permite a un usuario compartir cada uno de estos tipos de construcciones en diferentes lenguajes. Para ello, primero se debe definir una API (escrita en Java) que sea neutral respecto del lenguaje en relación con cada uno de una pluralidad de lenguajes de secuencias de comandos. Esta API consta de una interfaz para cada tipo de datos y cada tipo de lenguaje (es decir, para cada tipo de construcción del lenguaje). Estas interfaces de construcción de lenguaje se denominarán en el presente documento las interfaces Core y se nombrarán de tal manera que la palabra "Core" se anteponga al tipo (por ejemplo, CorePrimitive, CoreString, CoreFunction, Core-ClassDefinition, etc.). Además, todas las interfaces que componen el Core heredarán de una interfaz común llamada CoreObject.

Una interfaz de ejemplo para CoreFunction es la siguiente:

```
/**
*Una interfaz para representar genéricamente una función.
*/
la interfaz pública CoreFunction extiende CoreObject {
        /**
        * Llama a la función.
        */
        CoreObject exec (CoreObject ... args);
        // ....
```

La metodología desvelada en el presente documento hace un uso extensivo del patrón del adaptador para implementar y usar las interfaces Core. Para cada lenguaje,, se tienen que implementar dos adaptadores para cada tipo de construcción del lenguaje.

Se debe considerar una situación en la que el usuario de un ordenador realiza una llamada al Core que se refiere a un tipo específico de construcción de lenguaje y un lenguaje específico de secuencias de comandos. La programación del ordenador tendrá disponible una implementación del tipo de construcción del lenguaje en la API del lenguaje específico de secuencias de comandos y tendrá una interfaz para dicho tipo en la API del Core. El primero de los dos adaptadores envuelve la implementación del tipo de construcción del lenguaje en el lenguaje de la secuencia de comandos específica de modo que el primer adaptador se adapta a la API del Core y usa la implementación del lenguaje. El segundo adaptador realiza lo contrario y envuelve la interfaz Core de dicho tipo de construcción de lenguaje de modo que se adapta a la API del lenguaje de secuencia de comandos específica pero usa la instancia de la interfaz Core.

De manera más específica, para cada tipo de construcción del lenguaje, se toma una definición de clase de lenguaje existente y la interfaz Core existente correspondiente y luego escribe una nueva definición de clase que implementa esa interfaz Core (en adelante denominada Adaptador Core) y el otro adaptador permite que la interfaz Core para ese tipo de construcción de lenguaje se use dentro de la API de la implementación del lenguaje (en adelante denominado Adaptador de lenguaje).

Por ejemplo, se debe considerar el lenguaje Jython y considerar el tipo Función. La API para la implementación de la implementación del lenguaje Jython tiene una definición de clase llamada PyFunction. Un usuario tendría que escribir un Adaptador de Core que implemente la interfaz CoreFunction y redirija las llamadas del método realizadas hacia las llamadas del método en PyFunction. El usuario también tiene que escribir un Adaptador de lenguaje que emule una función de Python y redirija todas las llamadas a la CoreFunction que se está envolviendo.

La figura 1 muestra un patrón de adaptador para su uso cuando un usuario de Python llama a una función 14 (denominada DSLFunction) escrita en un lenguaje específico de dominio. El intérprete de Python determina que se debe hacer una llamada. Ejecuta la llamada en PyWrappedFunction 10, que es un adaptador de lenguaje y redirige la llamada a la interfaz CoreFunction 12, que en este caso es una implementación de CoreFunctionFromDSL. Esta implementación de CoreFunctionFromDSL 12 es un adaptador Core que redirige la llamada a la función DSL. DSLFunction es parte de la implementación del lenguaje específico del dominio y sabe cómo resolver internamente el método.

Por ejemplo, a continuación se expone cómo se define una función DSL en un archivo de texto:

```
-------------------------------------------
function add(a as double, b as double)
   return a + b
   end function
   ------------------------------------------------------------------------------
```

El intérprete DSL interpretará el texto y creará una instancia de función DSL. La función DSL normalmente se puede llamar desde el lenguaje DSL de esta manera:

```
------------------------------
c = add(1.0, 2.0)
------------------------------
```

El intérprete DSL interpretará el texto anterior al buscar la instancia de la DSL Function existente llamada "add". Pasará a dicha instancia de DSLFunction, dos entradas (1,0 y 2,0). La instancia DSL Function sabe internamente cómo "resolver la función". Ejecutará "a + b", que en este caso es "1,0 + 2,0" y devolverá la salida, 3,0.

Por tanto, cuando Python llama a esta misma función, el sistema convierte de objetos Py a objetos Core y, luego, de objetos Core a objetos DSL y luego llama a la función "real", que sabe cómo coger esas entradas y ejecutar el código que se utilizó para definir la función.

Un adaptador de lenguaje de ejemplo para convertir una función Python en una función Core puede adoptar la siguiente forma:

```
clase pública PvWrappedFunction extendiende PyObject {
    Función CoreFunction;

    público PyWrappedFunction (CoreFunction fcn) {
      function = fcn;
    }
    @Override
    público PyObject call ( PyObject args[], String keywords[]) {
    List <CoreObject> veArgs = new ArrayList <CoreObject>();

        para (PyObject arg: args) {
        CoreObject veArg = JythonAdapterFactory. fromJython( arg );
          veArgs.add( veArg );
        }
        resultado en CoreObject = function.exec( veArgs.toArray( new CoreObject[0] )
    );
        retorno JythonAdapterFactory.toJython( resultado);
      }
```

El Adaptador de lenguaje permite que una función de Python se disfrace como una función de Core.

Un Adaptador e Core ejemplar para convertir una función Core a una función Python tiene la siguiente forma:

```
        clase pública CoreFunctionImplementedInPython implementa
        CoreFunction {

            privado PyObject pycallable;

          @Override
          público CoreObject exec( CoreObject... args ) {

            PyObject[] pyargs pyargs = nuevo PyObject[args.length];

            int start = 0;
            para (int i = start; i < args.length + start; i++ ) {
              PyObject pyobj = JythonAdaptorFactory.toJython( args[i]);
              pyargs[i] = pyobj;
            }
            /**

        * En la práctica, debería haber un código adicional que atrape cualquier error en
        * la función proporcionada por el usuario y generar un seguimiento de pila significativo en
        * el mensaje de error. (Esto permite rastreos en varios lenguajes).
        */
        PyObject result = pycallable. call ( pyargs );

        retorno JythonAdaptorFactory.fromJyf/7on( result)
    } El Adaptador Core permite que una función Core se disfrace como una función Python.
```

Para implementar cada adaptador, el usuario debe poder crear adaptadores para una API basada en objetos existentes de otra API. Para apoyar esto, cada lenguaje de secuencias de comandos debe implementar una clase AdapterFactory con dos métodos. Un método convierte objetos para la API Core a la API del lenguaje y el otro método convierte objetos de la API del lenguaje a la API Core. La fábrica puede crear nuevas instancias de adaptadores o puede utilizar el almacenamiento en caché para poder reutilizar los adaptadores. Cabe considerar el caso en el que una función de Python se envuelve en una función Core y luego debe convertirse de nuevo en la API del lenguaje Python. En este caso, no se desea crear un adaptador, pero en su lugar debería recuperar la función subyacente de Python y devolverla.

Elaborando sobre la situación representada en la figura 1, la función DSL toma objetos DSL como entrada y devuelve un objeto DSL como salida. La función Core toma los objetos Core como entrada y devuelve un objeto Core como salida. La función Python toma los objetos Py como entrada y devuelve un objeto Py como salida.

Para que Python llame a la función DSL, es necesaria la función Python que adapta una función Core que adapta una función DSL. Por tanto,la función Python toma objetos Py como entrada. Estos objetos Py se convierten en objetos Core y la función Core es llamada por las entradas del objeto Core. Esta función Core toma las entradas del objeto Core y las convierte en entradas de objetos DSL y llama a la función DSL "real". Como resultado se devuelve un objeto DSL. La función Core que está adaptando la función DSL convierte el objeto DSL devuelto en un objeto Core y devuelve el objeto Core. Por último, la función Python toma el objeto Core devuelto y lo convierte en un objeto Python.

Como se usa en el presente documento, "convertir" significa que se llama a AdapterFactory para que se puedan crear o recuperar los adaptadores correctos.

Debe apreciarse que las interfaces Core no se limitan a aquellas que facilitan la traducción de objetos DSL en objetos Python y la traducción de objetos Python en objetos DSL, como se representa en la figura 1, sino que puede expandirse para incluir interfaces Core para facilitar la compartición de objetos escritos en otros lenguajes de secuencia de comandos. La figura 2 es un diagrama que muestra una clase CoreObject que se puede ampliar para facilitar la compartición de construcciones de lenguaje escritas en tres lenguajes de secuencias de comandos diferentes: Jython, JRuby y un lenguaje específico de dominio.

De acuerdo con el esquema representado en la figura 2, la API Core (clase CoreObject y extensiones) permite a un usuario de Jython acceder a una construcción de lenguaje escrita en JRuby o DSL; permite a un usuario de JRuby acceder a una construcción de lenguaje escrita en Jython o DSL; y permite a un usuario de DSL acceder a una construcción de lenguaje escrita en Jython o JRuby.

Por ejemplo, para permitir que un usuario de Jython acceda a una construcción de lenguaje escrita en un DSL, una primera instancia de un Adaptador de lenguaje Jython (creado o recuperado por Jython-Core AdapterFactory 24) convierte una primera instancia de un PyObject 16 en una primera instancia de un CoreObject 18; y luego una primera instancia de un adaptador DSL-Core (creado o recuperado por la DSL-Core AdapterFactory 26) convierte esa primera instancia de un CoreObject 18 en una primera instancia de un DSLObject 20. Posteriormente, una primera instancia de un adaptador de lenguaje DSL (creado o recuperado por la DSL-Core AdapterFactory 26) convierte una segunda instancia de un DSLObject 20 en una segunda instancia de un CoreObject 18; y luego una primera instancia de un Adaptador Jython-Core (creado o recuperado por la Jython-Core AdapterFactory 24) convierte esa segunda instancia de un CoreObject 18 en una segunda instancia de un PyObject 16.

De manera similar, para permitir que un usuario de JRuby acceda a una construcción de lenguaje escrita en un DSL, una primera instancia de un Adaptador de lenguaje JRuby (creado o recuperado por JRuby-Core AdapterFactory 28) convierte una primera instancia de un RubyObject 22 en una tercera instancia de un CoreObject 18; y luego una segunda instancia de un adaptador DSL-Core (creado o recuperado por la DSL-Core AdapterFactory 26) convierte esa tercera instancia de un CoreObject 18 en una tercera instancia de un DSLObject 20. Posteriormente, una segunda instancia de un adaptador de lenguaje DSL (creado o recuperado por DSL-Core AdapterFactory 26) convierte una cuarta instancia de un DSLObject 20 en una cuarta instancia de un CoreObject 18; y luego una primera instancia del adaptador JRuby-Core (creada o recuperada por JRuby-Core AdapterFactory 28) convierte esa cuarta instancia de un CoreObject 18 en una segunda instancia de un RubyObject 22.

De forma similar, un usuario de DSL puede importar construcciones de lenguaje escritas en el lenguaje de secuencias de comandos Jython o JRuby; un usuario de JRuby puede importar construcciones de lenguaje escritas en el lenguaje de secuencias de comandos Jython; y un usuario de Jython puede importar construcciones de lenguaje escritas en el lenguaje de secuencias de comandos JRuby usando la API Core. Como se ha mencionado anteriormente, la API Core incluye una interfaz Core respectiva para cada construcción de lenguaje, de modo que las interfaces Core heredan de una interfaz común llamada CoreObject.

Algunos lenguajes distinguen entre mayúsculas y minúsculas y otros no. Por lo tanto, las interfaces para las instancias de Diccionario y Clase no buscan por Cadena. En su lugar, se implementa una interfaz separada que define una clave llamada CoreKey. La realización incluye implementaciones de CoreKey que distinguen entre mayúsculas y minúsculas y otras que no distinguen entre mayúsculas y minúsculas.

Para habilitar el sistema descrito anteriormente, cada lenguaje debe soportar el concepto de un gancho (hook) de importación. El término gancho de importación proviene de la comunidad Python y es un mecanismo para redefinir la función a la que se llama cuando se realiza una importación. La función de importación tiene que mejorarse para que pueda manejar módulos que fueron escritos en otros lenguajes. La convención común es usar la extensión del nombre de archivo para determinar el lenguaje que se debe usar para interpretar el archivo. Una vez que se

interpreta el archivo, debería producir alguna representación de Módulo o Diccionario del contenido del archivo. Esa representación se puede usar para crear un CoreModuleAdapter que se puede envolver en un módulo.

Obsérvese que los ganchos de importación no están soportados por el lenguaje de destino, entonces el mecanismo general seguirá funcionando correctamente, pero puede requerir alguna modificación de las secuencias de comandos de llamada para que se realice una llamada a la función para importar las secuencias de comandos en lugar de una declaración de importación más natural.

Obsérvese también que algunos lenguajes almacenan en caché los módulos importados para que no tengan que ser interpretados nuevamente (asimismo, las definiciones de clase no tienen que redefinirse). Cuando se produce una importación en un lenguaje, el implementador de la realización debe tener cuidado de almacenar en caché el módulo importado en cada uno de los lenguajes admitidos.

La metodología descrita anteriormente proporciona ahorros de costes al permitir que los ingenieros compartan sus códigos de análisis fácilmente. Se pueden integrar rápidamente diferentes aplicaciones escritas en diferentes lenguajes. Hay otro ahorro de costes cuando los ingenieros pueden escribir códigos en lenguajes con los que se sientan cómodos. No necesitan pasar tiempo entrenando en un nuevo lenguaje. También hay una evitación de costes. Al facilitar la compartición del código, hay muchas más posibilidades de reutilizar el código.

Si bien la invención se ha descrito con referencia a diversas realizaciones, los expertos en la materia entenderán que se pueden realizar diversos cambios sin apartarse del alcance de la invención. Además, se pueden realizar muchas modificaciones para adaptar una situación o material concreto a las enseñanzas de la invención sin desviarse del alcance esencial de la misma. Por lo tanto, se pretende que la invención no se limite a la realización en particular desvelada como el mejor modo contemplado para llevar a cabo la presente invención.

No debe interpretarse que las reivindicaciones del método expuestas más adelante requieren que todas las operaciones del método se realicen en el orden en que se citan.

REIVINDICACIONES

1. Un método para compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos basados en Java, en el que las construcciones del lenguaje incluyen tipos de datos, tales como primitivas (incluyendo entero, doble, carácter, número entero corto, de punto flotante, byte y construcciones específicas del lenguaje), cadenas, listas, diccionarios, conjuntos, tuplas, matrices y tipos de lenguaje, tales como funciones, definiciones de clase, instancias de clase, métodos y módulos de código, en el que los lenguajes de secuencias de comandos se refieren a los lenguajes interpretados que están configurados para permitir que un usuario maneje una aplicación desde una señal de comando o un archivo de texto, en el que las construcciones de lenguaje se tratan como objetos de primera clase configurados para ser compartidos entre los diferentes lenguajes de secuencias de comandos basados en Java, comprendiendo el método:

(a) definir, usando un lenguaje de programación, una interfaz del núcleo de programación de aplicaciones que es neutral en cuanto al lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos basados en Java que permite el uso en un lenguaje de programación interpretado de construcciones de lenguaje escritas en un lenguaje de programación interpretado diferente, comprendiendo dicha interfaz de programación de aplicaciones del núcleo que comprende una interfaz (12) del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje; en el que un procesador está configurado para ejecutar las siguientes operaciones:
(b) llamar a una construcción de lenguaje (14) de un primer tipo escrito en un primer lenguaje de secuencias de comandos basado en Java, estando la llamada realizada por una construcción de lenguaje (10) de dicho primer tipo escrito en un segundo lenguaje de secuencias de comandos basado en Java;
(c) crear una primera instancia de una primera interfaz del núcleo (12) que maneja construcciones de lenguaje de dicho primer tipo;
(d) redirigir la llamada a dicha primera instancia de dicha primera interfaz (12) del núcleo; y
(e) redirigir la llamada recibida por dicha primera instancia de dicha primera interfaz (12) del núcleo a dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos basado en Java.

2. El método según la reivindicación 1, que comprende adicionalmente:

crear una instancia de un primer adaptador del lenguaje; y
crear una instancia de un primer adaptador del núcleo,
en el que la operación (d) se implementa mediante dicha instancia de dicho primer adaptador de lenguaje y la operación (e) se implementa mediante dicha instancia de dicho primer adaptador del núcleo.

3. El método según la reivindicación 2, que comprende adicionalmente:

(f) crear una instancia de dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos basado en Java en respuesta a la llamada;
(g) devolver un resultado de dicha instancia de dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos basado en Java;
(h) crear una segunda instancia de dicha primera interfaz del núcleo;
(i) redirigir el resultado devuelto a dicha segunda instancia de dicha primera interfaz del núcleo; y
(j) redirigir el resultado devuelto recibido por dicha segunda instancia de dicha primera interfaz del núcleo a dicha construcción (10) de lenguaje de dicho primer tipo escrito en dicho segundo lenguaje de secuencias de comandos basado en Java.

4. El método según la reivindicación 3, que comprende adicionalmente:

crear una instancia de un segundo adaptador de lenguaje; y
crear una instancia de un segundo adaptador del núcleo,
en el que la operación (i) se implementa mediante dicha instancia de dicho segundo adaptador de lenguaje y la operación (j) se implementa mediante dicha instancia de dicho segundo adaptador del núcleo.

5. El método según la reivindicación 1, 2, 3 o 4, en el que dicho primer tipo de construcción de lenguaje se selecciona del siguiente conjunto de tipos: primitivas, cadenas, listas, diccionarios, conjuntos, tuplas, matrices, funciones, definiciones de clase, instancias de clase, métodos y módulos.

6. El método según la reivindicación 4 o 5 que comprende además:

(k) llamar a una construcción (22) de lenguaje de un segundo tipo escrito en un tercer lenguaje de secuencias de comandos basado en Java, estando la llamada realizada por una construcción (16) de lenguaje de dicho segundo tipo escrito en dicho segundo lenguaje de secuencias de comandos basado en Java;

(l) crear una primera instancia de una segunda interfaz del núcleo que maneja construcciones de lenguaje de dicho segundo tipo;
(m) redirigir la llamada a dicha primera instancia de dicha segunda interfaz del núcleo; y
(n) redirigir la llamada recibida por dicha primera instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje (22) de dicho segundo tipo escrita en dicho tercer lenguaje de secuencias de comandos basado en Java.

7. El método según la reivindicación 4, 5 o 6, que comprende adicionalmente:

crear una instancia de un tercer adaptador del lenguaje; y
crear una instancia de un tercer adaptador del núcleo,
en el que la operación (m) se implementa mediante dicha instancia de dicho tercer adaptador del lenguaje y la operación (n) se implementa mediante dicha instancia de dicho tercer adaptador del núcleo.

8. El método según la reivindicación 4, 5, 6 o 7, que comprende adicionalmente:

(o) crear una instancia de dicha construcción (22) de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencias de comandos basado en Java en respuesta a la llamada;
(p) devolver un resultado de dicha instancia de dicha construcción (22) de lenguaje de dicho segundo tipo escrito en dicho tercer lenguaje de secuencias de comandos basado en Java;
(q) crear una segunda instancia de dicha segunda interfaz del núcleo;
(r) redirigir el resultado devuelto a dicha segunda instancia de dicha segunda interfaz del núcleo; y
(s) redirigir el resultado devuelto recibido por dicha segunda instancia de dicha segunda interfaz del núcleo a dicha construcción (16) de lenguaje de dicho segundo tipo escrito en dicho segundo lenguaje de secuencias de comandos basado en Java.

9. El método según la reivindicación 8, que comprende adicionalmente:

crear una instancia de un cuarto adaptador de lenguaje; y
crear una instancia de un cuarto adaptador del núcleo,
en el que la operación (r) se implementa mediante dicha instancia de dicho cuarto adaptador de lenguaje y la o las operaciones se implementan mediante dicha instancia de dicho cuarto adaptador del núcleo.

10. Un sistema que permite compartir construcciones de lenguaje entre diferentes lenguajes de secuencias de comandos basados en Java, en el que las construcciones del lenguaje incluyen tipos de datos, tales como primitivas (incluyendo entero, doble, carácter, número entero corto, de punto flotante, byte y construcciones específicas del lenguaje), cadenas, listas, diccionarios, conjuntos, tuplas, matrices y tipos de lenguaje, tales como funciones, definiciones de clase, instancias de clase, métodos y módulos de código, en el que los lenguajes de secuencias de comandos se refieren a los lenguajes interpretados que están configurados para permitir que un usuario maneje una aplicación desde una señal de comando o un archivo de texto, en el que las construcciones de lenguaje se tratan como objetos de primera clase configurados para ser compartidos entre los diferentes lenguajes de secuencias de comandos basados en Java, comprendiendo dicho sistema una interfaz de programación de aplicaciones del núcleo que es neutral en cuanto al lenguaje en relación con una pluralidad de lenguajes de secuencias de comandos basados en Java que permite el uso en un lenguaje de programación interpretado de construcciones de lenguaje escritas en un lenguaje de programación interpretado diferente, comprendiendo dicha interfaz de programación de aplicaciones del núcleo una interfaz (12) del núcleo respectiva para cada uno de una pluralidad de tipos de construcciones de lenguaje y un procesador programado para ejecutar las siguientes operaciones:

(a) llamar a una construcción (14) de lenguaje de un primer tipo escrito en un primer lenguaje de secuencias de comandos basado en Java, estando la llamada realizada por una construcción de lenguaje (10) de dicho primer tipo escrito en un segundo lenguaje de secuencias de comandos basado en Java;
(b) crear una primera instancia de una primera interfaz (12) del núcleo que maneja construcciones de lenguaje de dicho primer tipo;
(c) redirigir la llamada a dicha primera instancia de dicha primera interfaz (12) del núcleo; y
(d) redirigir la llamada recibida por dicha primera instancia de dicha primera interfaz (12) del núcleo a dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencia de comandos basado en Java.

11. El sistema según la reivindicación 10, en el que dicho procesador se programa además para ejecutar las siguientes operaciones:

crear una instancia de un primer adaptador del lenguaje; y
crear una instancia de un primer adaptador del núcleo,
en el que la operación (c) se implementa mediante dicha instancia de dicho primer adaptador de lenguaje y la

operación (d) se implementa mediante dicha instancia de dicho primer adaptador del núcleo.

12. El sistema informático según la reivindicación 11, en el que dicho procesador se programa además para ejecutar las siguientes operaciones:

(e) crear una instancia de dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos basado en Java en respuesta a la llamada;
(f) devolver un resultado de dicha instancia de dicha construcción (14) de lenguaje de dicho primer tipo escrito en dicho primer lenguaje de secuencias de comandos basado en Java;
(g) crear una segunda instancia de dicha primera interfaz del núcleo;
(h) redirigir el resultado devuelto a dicha segunda instancia de dicha primera interfaz del núcleo; y
(i) redirigir el resultado devuelto recibido por dicha segunda instancia de dicha primera interfaz del núcleo a dicha construcción (10) de lenguaje de dicho primer tipo escrito en dicho segundo lenguaje de secuencias de comandos basado en Java.

13. El sistema informático según la reivindicación 12, en el que dicho procesador se programa además para ejecutar las siguientes operaciones:

crear una instancia de un segundo adaptador de lenguaje; y
crear una instancia de un segundo adaptador del núcleo,
en el que la operación (h) se implementa mediante dicha instancia de dicho segundo adaptador de lenguaje y la operación (i) se implementa mediante dicha instancia de dicho segundo adaptador del núcleo.

14. El sistema informático según la reivindicación 13, en el que dicho procesador se programa además para ejecutar las siguientes operaciones:

(j) llamar a una construcción (22) de lenguaje de un segundo tipo escrito en un tercer lenguaje de secuencias de comandos basado en Java, estando la llamada realizada por una construcción (16) de lenguaje de dicho segundo tipo escrito en dicho segundo lenguaje de secuencias de comandos basado en Java;
(k) crear una primera instancia de una segunda interfaz del núcleo que maneja construcciones de lenguaje de dicho segundo tipo;
(l) redirigir la llamada a dicha primera instancia de dicha segunda interfaz del núcleo; y
(m) redirigir la llamada recibida por dicha primera instancia de dicha segunda interfaz del núcleo a dicha construcción de lenguaje (22) de dicho segundo tipo escrita en dicho tercer lenguaje de secuencias de comandos basado en Java.

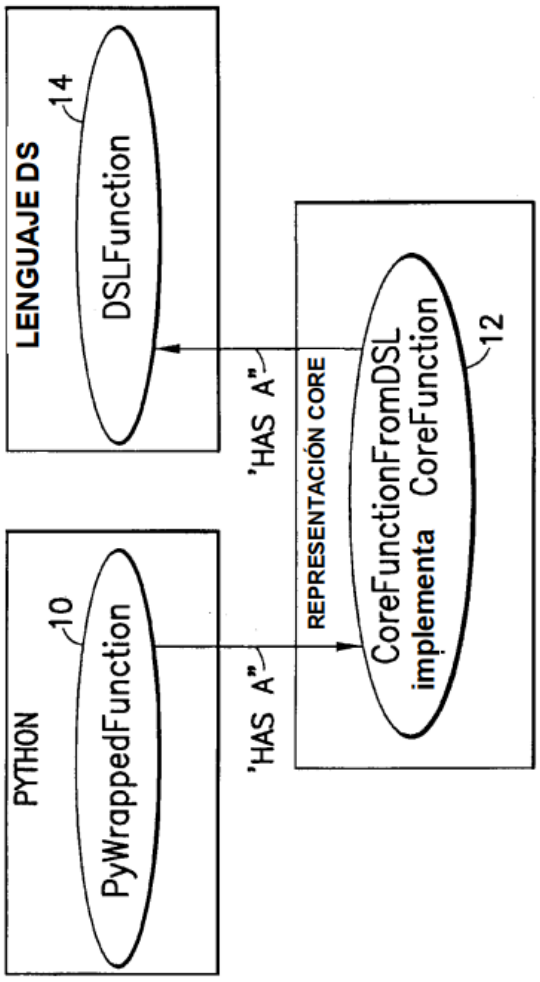
LENGUAJE DS
14
DSLFunction
'HAS A"
REPRESENTACIÓN CORE
CoreFunctionFromDSL
implementa CoreFunction
12
'HAS A"
PYTHON
10
PyWrappedFunction

FIG. 1

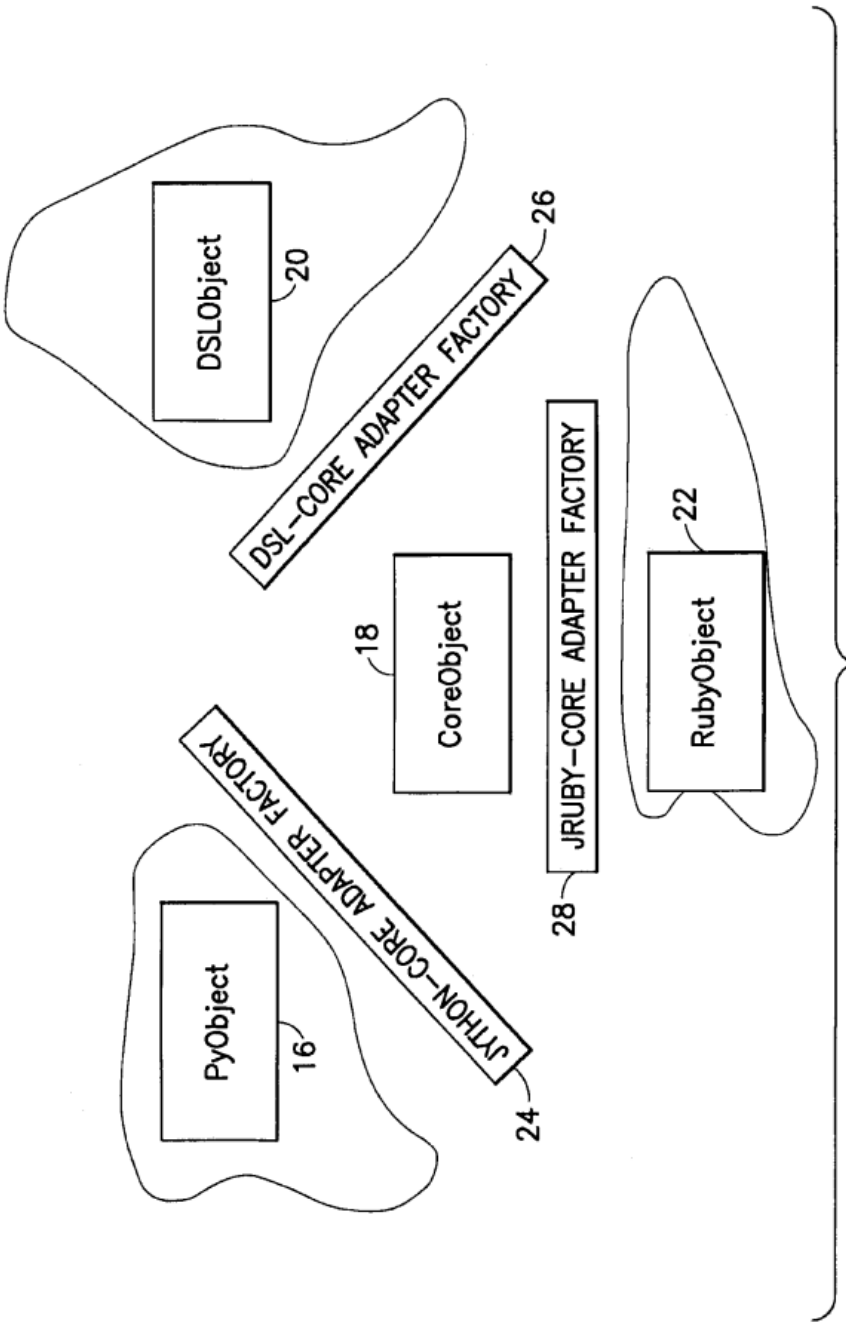
DSLObject
20
DSL-CORE ADAPTER FACTORY
26
CoreObject
18
JRUBY-CORE ADAPTER FACTORY
28
RubyObject
22
JYTHON-CORE ADAPTER FACTORY
24
PyObject
16

FIG.2