

MULTIPLEXACIÓN Y CADENA DE TAREA ASÍNCRONICA

REFERENCIA CRUZADA A LA SOLICITUD RELACIONADA

[0001] La presente solicitud reivindica la prioridad para la solicitud de patente no provisional U.S. número de serie 14/843.870 presentada el 2 de septiembre de 2015 y la solicitud de patente provisional U.S. número de serie 62/046.116, presentada el 4 de septiembre de 2014, la totalidad de cada una se incorporan aquí como referencia.

ANTECEDENTES

[0002] En la ejecución de un programa de computador, una pieza de trabajo de cómputo puede tomar una cantidad indefinida de tiempo para completarse. Por ejemplo, la finalización de una solicitud de red depende de varios factores, incluyendo latencia, ancho de banda disponible, si el enlace de comunicación de red está inactivo, si el servidor está inactivo o está funcionando lentamente, etc.

[0003] En un modelo de programación sincrónico, si se realiza una llamada a algo que tarda un rato, el código de programa bloquea y espera a que se complete la llamada (aunque el código de programa eventualmente puede agotar tiempo si transcurre demasiado tiempo). Esperar la finalización generalmente es indeseable porque otras partes del código de programa pueden ser capaces de realizar un trabajo útil mientras se espera que la tarea lenta se complete.

[0004] Una forma de resolver este problema es usar múltiples subprocesos de ejecución. Sin embargo, los múltiples subprocesos de ejecución no siempre están disponibles; e.g., ciertos entornos de programación son de subproceso único.

Además, el uso de múltiples procesos de ejecución no siempre es eficiente en algunos escenarios, e.g., porque la conmutación de subprocesos consume recursos.

[0005] Otra forma de resolver el problema de espera es hacer la llamada y obtener el resultado de forma asincrónicamente, en un momento posterior cuando el trabajo esté completo. Hay muchas variaciones diferentes de esta idea asincrónica básica (e.g., futuros, promesas, tareas, canales..., denominados aquí "tareas asincrónicas" o "trabajo asincrónico"). Entre los beneficios de la llamada asincrónica se incluyen los que son subprocesos agnósticos. Sin embargo, también hay posibles complicaciones con la ejecución de tareas asincrónicas que se deben considerar para proporcionar código de programa robusto.

RESUMEN

[0006] Este resumen se proporciona para introducir una selección de conceptos representativos en una forma simplificada que se describen más adelante en la descripción detallada. El presente resumen no pretende identificar las características clave o características esenciales de la materia reivindicada ni tampoco pretende ser utilizado de ninguna manera que limite el alcance de la materia reivindicada.

[0007] En resumen, la tecnología descrita en el presente documento está dirigida a multiplexar una tarea asincrónica que se identifica en al menos dos cadenas de tarea, incluyendo compartir resultados de la tarea asincrónica entre un conjunto de oyentes que comprende al menos dos oyentes dependientes de nivel inferior. La tecnología incluye mantener la información de identidad de cada oyente dependiente de nivel inferior en asociación con la tarea asincrónica, e

impide la cancelación de un oyente a partir de la cancelación de la tarea asincrónica, siempre y cuando por lo menos un oyente permanezca dependiente de la tarea asincrónica con base en la información de identidad.

[0008] También se describe aquí el envolver las tareas en el código de verificación de anulación que impide que una tarea realice su trabajo si se pretende cancelar la tarea, pero se pone en cola para ejecutarse antes de que la solicitud de cancelación quede en cola para ejecutarse.

[0009] Otras ventajas pueden resultar evidentes a partir de la siguiente descripción detallada cuando se toma en conjunción con los dibujos.

BREVE DESCRIPCIÓN DE LOS DIBUJOS

[0010] La presente tecnología se ilustra a modo de ejemplo y no se limita en las figuras adjuntas en las que números de referencia similares indican elementos similares y en los que:

[0011] La FIG. 1 es un diagrama de bloques que muestra una configuración de ejemplo de componentes que se pueden utilizar para compartir tareas asincrónicas (asinc), incluyendo multiplexar la tarea asincrónica compartida para gestionar cancelaciones de oyentes, de acuerdo con una o más implementaciones de ejemplo.

[0012] La FIG. 2 es una representación de cómo una tarea asincrónica compartida se puede multiplexar para notificar a sus oyentes no cancelados al finalizar la tarea, de acuerdo con una o más implementaciones de ejemplo.

[0013] La FIG. 3 es un diagrama de flujo que muestra las etapas de ejemplo que se pueden tomar para poner en cola una tarea asincrónica para correr hasta su finalización o cancelación, de acuerdo con una o más implementaciones de ejemplo.

[0014] La FIG. 4 es un diagrama de flujo que muestra las etapas de ejemplo que se pueden tomar para gestionar solicitudes de cancelación dirigidas a una tarea asincrónica compartida, de acuerdo con una o más implementaciones de ejemplo.

[0015] La FIG. 5 es un diagrama de bloques que muestra un proceso de envoltura que envuelve tareas asincrónicas en el código de cancelación de verificación para evitar la ejecución no intencionada de una tarea asincrónica cancelada, de acuerdo con una o más implementaciones de ejemplo.

[0016] Las FIGS. 6A y 6B son diagramas de flujo que muestran las etapas de ejemplo dirigidos a ejecutar una tarea asincrónica (FIG. 6A) envuelta con código de cancelación de verificación, incluyendo la verificación de cancelación (FIG. 6B), de acuerdo con una o más implementaciones de ejemplo.

[0017] La FIG. 7 es un diagrama de flujo que muestra las etapas de ejemplo que se pueden tomar con respecto a la ejecución de una cadena de tareas, de acuerdo con uno o más implementaciones de ejemplo.

[0018] La FIG. 8 es un diagrama de flujo que muestra las etapas de ejemplo que se pueden tomar con respecto a la preparación de tareas y cadenas de tareas para correr y para posible cancelación, de acuerdo con una o más implementaciones de ejemplo.

[0019] Las FIGS. 9A y 9B son diagramas de flujo que muestran las etapas de ejemplo que se pueden tomar con respecto al uso de un número de versión con respecto a la cancelación de cadenas de tareas, de acuerdo con una o más implementaciones de ejemplo.

[0020] La FIG. 10 es un diagrama de flujo que muestra las etapas de ejemplo que se pueden tomar con respecto a la optimización de cadenas de tareas para ejecutar, incluyendo compartir tareas asincrónicas y multiplexar tareas asincrónicas compartidas, de acuerdo con uno o más implementaciones de ejemplo.

[0021] Las FIGS. 11 y 12 son ejemplos de representaciones de cómo se pueden procesar cadenas de tareas para optimización, de acuerdo con uno o más implementaciones de ejemplo.

[0022] La FIG. 13 es un diagrama de bloques que representa un ejemplo del entorno de cómputo en el que se pueden incorporar aspectos de la materia objeto de la presente invención.

DESCRIPCIÓN DETALLADA

[0023] La tecnología descrita en el presente documento está dirigida hacia encadenamiento y optimización de tareas o trabajo asincrónicos “asinc”, a la vez que se protege de los problemas que resultan de la cancelación de una tarea asincrónica. Como se comprenderá, esto incluye la optimización de tareas redundantes en tareas compartidas, utilizando uno o más multiplexores de tareas asincrónicas compartidas que gestionan la cancelación y la protección contra la ejecución de una tarea asincrónica destinada a ser cancelada.

[0024] En un aspecto, como parte de la optimización de cadenas de tareas asincrónicas, una tarea asincrónica que depende de más de una cadena puede ser compartida por oyentes posteriores, que comprende otras entidades (e.g., otras tareas asincrónicas) que dependen de la tarea asincrónica compartida. En general y como se comprenderá, si se cancela un oyente, en lugar de cancelar también la tarea asincrónica compartida, la multiplexación de la tarea asincrónica compartida evita la cancelación de la tarea asincrónica multiplexada compartida (a menos y hasta que no haya otros oyentes). Esto se debe a que otros oyentes también pueden depender de la tarea asincrónica compartida.

[0025] Además, en general y como se comprenderá, aquí se describe una manera de evitar que se ejecute una tarea "a ser cancelada". Una tarea que se va a cancelar puede estar en cola para ejecutarse antes de que se cancele la tarea de cancelación (es decir, la tarea que cancela esa tarea a ser cancelada) está en cola, en cuyo caso la tarea que se va a cancelar se ejecuta antes de la tarea de cancelación, lo cual puede ser problemático. Para hacer frente a esta situación, una tarea se envuelve con el código que verifica su cancelación antes de realizar sus operaciones; incluso si está en cola antes de la tarea de cancelación, la tarea cuando se deshace y se ejecuta no realizará ninguna otra operación más allá de la verificación de cancelación envuelta si la tarea se ha cancelado.

[0026] Se debe entender que cualquiera de los ejemplos de la presente invención no son limitativos. Por ejemplo, mientras que los programas de aplicación de un solo subproceso, como los escritos en JavaScript®, se pueden beneficiar de la tecnología aquí descrita, y de hecho JavaScript® promete utilizar en algunos ejemplos, cualquier lenguaje de programación adecuado, incluidos aquellos en los que se dispone de multiproceso, pero, por ejemplo, se considera menos deseable, al menos en parte, por un desarrollador. Como otro ejemplo, se

ejemplifica aquí un entorno informático general, sin embargo se entiende que cualquier máquina/dispositivo capaz de ejecutar tareas asincrónicas se puede beneficiar de la tecnología descrita aquí, incluyendo consolas de juegos y entretenimiento, computadores personales (e.g., computadores portátiles y de escritorio) estaciones de trabajo, servidores, dispositivos portátiles, teléfonos inteligentes, dispositivos de computación de tableta y similares. Como tal, la tecnología descrita aquí no está limitada a ninguna implementación, aspectos, conceptos, estructuras, funcionalidades o ejemplos particulares descritos en el presente documento. Más bien, cualquiera de las implementaciones, aspectos, conceptos, estructuras, funcionalidades o ejemplos descritos en el presente documento no son limitativos y la presente tecnología se puede usar de diversas maneras que proporcionan beneficios y ventajas en la tecnología informática asincrónica en general.

[0027] La FIG. 1 es un diagrama de bloques que muestra componentes de ejemplo dirigidos hacia la multiplexación y encadenamiento de tareas asincrónicas. En general, el código de programa 102 incluye un número de entidades solicitantes 104, tales como procesos, objetos y similares. Por ejemplo, se muestra una entidad solicitante 104(a) (e.g., una instancia de objeto) que tiene un número de tareas asincrónicas 106 que necesitan ejecutarse.

[0028] En general, las tareas son organizadas por las entidades solicitantes para que sean relativamente pequeñas y rápidas de operar. Por ejemplo, considere una parte del código del programa en la que una cadena debe ser buscada y analizada para buscar una condición, después de la cual se toma alguna acción con base en la condición que se cumpla o no se cumpla como resultado del análisis. Tales actos pueden tomar un tiempo largo o una cantidad de tiempo indeterminado para completarse, y por lo tanto el código de programa

puede dividir la lógica en una cadena de tareas más pequeñas, (también denominadas artículos de trabajo), e.g., una tarea para solicitar la cadena, seguida por una tarea dependiente para analizar la cadena después de completar la solicitud de búsqueda, seguida por una tarea dependiente para determinar la acción a realizar después de completar el análisis, seguida por una tarea dependiente que realiza la acción. Además, la tarea de análisis se puede dividir, e.g., sólo se puede analizar un número máximo de caracteres a la vez. De esta manera, el código de programa posiblemente lento se puede manejar por una serie de tareas asincrónicas encadenadas, permitiendo que otras tareas asincrónicas que se deben manejar (haciendo colas de forma similar a esas otras tareas a medida que vienen) sin tener que esperar a que una parte mayor del código de programa lento se complete en su totalidad. Además, tener tareas pequeñas permite volver al código de programa sobre una base generalmente regular, e.g., para una parte de cada trama de reproducción de modo que el programa no aparezca congelado para el usuario.

[0029] En general, se puede encadenar un conjunto de tareas (hasta cualquier número práctico), e.g., las tareas [A, B, C,...] se pueden encadenar de forma que la tarea A se ejecute, la tarea B se ejecuta si la tarea A se completa correctamente y, a continuación, se ejecuta la tarea C si la tarea B se completa correctamente y así sucesivamente. Además, tal como se describe en el presente documento, la optimización de tareas 108 se puede realizar para evitar duplicar innecesariamente tareas que son necesarias en más de una cadena. Por ejemplo, una tarea asincrónica X se puede ejecutar una vez para proporcionar un resultado que es necesario por alguna otra entidad Y y alguna otra entidad Z. Con este fin, la optimización 108 puede combinar una solicitud que pide la tarea X luego Y, y la Tarea X luego Z, de modo que la tarea asincrónica X se ejecuta una vez, con su

resultado compartido por Y y Z, referidos aquí como "oyentes" en relación con la tarea asincrónica compartida X.

[0030] Como se puede apreciar fácilmente, la optimización se puede realizar dinámicamente a nivel de entidad, e.g., un objeto se puede codificar para reconocer la capacidad de compartir la tarea X una vez que el objeto determina las cadenas de tareas que necesita ejecutar. Alternativamente, (o además de la optimización dinámica), al menos algunas cadenas de tareas se pueden optimizar antes del tiempo de ejecución, por ejemplo, un objeto puede tener sus tareas optimizadas para compartir durante el tiempo de carga del programa (o tiempo de compilación) durante el tiempo de ejecución, las cadenas ya están optimizadas. Más aún, es factible tener una instrucción de lenguaje de programación que permita al desarrollador especificar directamente el uso compartido, e.g., compartir (X), oyentes (Y, Z).

[0031] Además, una tarea que se comparte se puede multiplexar como se describe en el presente documento. En general, la multiplexación se refiere a proteger una tarea compartida de ser cancelada por un oyente cuando hay otro oyente independiente que todavía depende de esa tarea compartida.

[0032] Tal como se representa en la FOG. 1 por el bloque 110, hay por lo tanto tareas, tareas multiplexadas (compartidas) y sus oyentes, y tareas encadenadas que pueden estar presentes para realizar el trabajo asincrónico de un código de programa. Una tarea multiplexada es seguida por los oyentes, y un oyente puede ser parte de una cadena de tareas. Una tarea multiplexada puede ser un oyente de una tarea anterior o una tarea compartida anterior (multiplexada).

[0033] Durante el tiempo de ejecución, el código de programa 102 (e.g., la instancia de objeto 104(a) en el mismo) llama una función o similar de un procesador de tareas asincrónico 112 que pone en cola cada tarea en una o más colas internas o externas adecuadas 114 (e.g., puede haber diferentes colas de prioridad). En general, no se solicita que una tarea encadenada dependiente quede en cola hasta que la tarea de la que depende se completa exitosamente. El procesador de tareas asincrónicas 112 deshace y ejecuta (bloque 116) las tareas en cola, por ejemplo, con base en su ordenación de cola (incluyendo el factorización en cualquier consideración de prioridad de cola).

[0034] Un procesador de tareas asincrónico adecuado comprende un despachador asincrónico, tal como se describe en la solicitud de patente de Estados Unidos serie número 14/803.842, incorporado aquí como referencia. Dicho procesador de tareas asincrónico 112 y sus colas 114 se pueden alojar en el código de programa 102 alojado en un navegador u otro programa adecuado acoplado al código de programa 102 alojado en un navegador u otro programa adecuado que está alojado en el código de programa 102, o de otro modo ejecutado de cualquier otra manera adecuada, e.g., como una entidad independiente.

[0035] En general, una tarea asincrónica se puede completar con éxito o fracaso, como se representa en la FIG. 1 por la flecha de "Resultados de tarea" que devuelve a las entidades solicitantes 104. Además, si el código de programa desea detener una tarea asincrónica antes de que se complete, el código de programa puede cancelar la tarea asincrónica. Tenga en cuenta, sin embargo, que la cancelación de una tarea se basa en poner en cola una tarea de cancelación, (tal como se describe aquí puede dar lugar a problemas si una tarea que se ha

cancelado ya se ha puesto en cola antes de que la tarea de cancelación esté en cola).

[0036] Para resumir, el código de programa 102 puede tener una tarea asincrónica dependiente de otra para crear "cadenas" o "árboles" de ellos, e.g., una tarea B puede necesitar que la tarea A se complete con éxito para que la tarea B pueda usar los resultados de la tarea A. Las promesas (e.g., en JavaScript®) son una forma de establecer este vínculo de dependencia a través del método `luego()`, e.g., `A.luego(B)`. Se puede establecer una cadena más grande, e.g., `A.luego(B)`, `luego (C)`, `luego (D)`, y así sucesivamente. Además, a través de la optimización, una sola tarea asincrónica puede tener más de una tarea asincrónica dependiendo de ella. Por ejemplo, la tarea asincrónica A puede recuperar algunos datos de la red que la tarea B necesita, al igual que una cadena de tareas separada que incluye X; `A.luego(B)` y `A.luego(X)` pueden ser dos promesas JavaScript® separadas, por lo que mediante la optimización ambas pueden hacerse dependientes de la tarea A. A continuación se describen detalles relacionados con la optimización con referencia a las FIGS. 10-12.

[0037] Volviendo a los aspectos de multiplexación, en la programación contemporánea, la cancelación de una tarea asincrónica que forma parte de una cadena de tareas hace burbujas hacia arriba y hacia abajo en la cadena de tareas. Por ejemplo, si se cancela la tarea B de una cadena de tareas $A \rightarrow B \rightarrow C$, la tarea C se cancela, al igual que la tarea A. Esto puede causar una serie de problemas.

[0038] A modo de ejemplo de una cuestión problemática, considere que hay una pieza de trabajo asincrónica A desde la cual múltiples entidades (tales como procesos u otros elementos de trabajo asincrónicos) quieren el resultado, e.g., A

→ B → C y A → X. Cada entidad puede emitir el elemento de trabajo A por separado, pero esto es típicamente ineficiente. Como ejemplo típico, considere que el elemento de trabajo asincrónico A recupera una gran cantidad de datos de la red, que es procesada posteriormente por varias entidades; es ineficiente ejecutar A para recuperar los mismos datos para cada entidad. En cambio, es típicamente más eficiente si una primera entidad inicia el trabajo A y las otras entidades B y X escuchan el resultado.

[0039] Por ejemplo con Promesas, la Promesa A hace el trabajo y luego cada oyente hace A.luego(...) para obtener su propia Promesa dependiente del resultado de la tarea de A. Esto funciona bien si la tarea A se completa, sin embargo la cancelación de antemano presenta un problema. Más particularmente, si alguno de los oyentes como la tarea X decide cancelar, en los entornos contemporáneos la cancelación de cascadas y cancela la tarea original A. No sólo se anula la tarea A, sino que todos los demás oyentes (tarea B y tarea C en este ejemplo) también se cancelan, ya que dependen de A, que ya cancelado nunca se resolverá con éxito. Sin embargo, aunque la tarea B y la tarea C son oyentes independientes en relación con la tarea X, y pueden no tener conocimiento de la entidad cancelada X (ya que estaban unidos sólo por optimización), en los sistemas convencionales esas cancelaciones de burbuja arriba y abajo de una cadena, la tarea B y la tarea C se ven afectadas por la cancelación de la tarea X, lo cual es problemático.

[0040] Para superar este problema, se describe aquí un multiplexor 202 (FIG. 2) que envuelve una pieza de trabajo asincrónico compartido (tarea asincrónica A 204) y mientras está pendiente la tarea asincrónica A 204, puede distribuir nuevas tareas dependientes A para los oyentes (solicitantes, al menos tres de los cuales están ejemplificados, etiquetados 206-208) que dependen de la tarea asincrónica

original A 204, mientras que evitan que cualquier cancelación alcance la tarea asincrónica original A 204. Sólo si cada uno de los oyentes independientemente, cancela el multiplexor 202 cancela las tareas asincrónica A 204 subyacentes compartidas (a menos que A se cancele directamente). De este modo, la tarea asincrónica compartida A 204 se puede ejecutar y finalizar, con lo que se notifican todas las tareas (no canceladas) (bloque 210); e.g., cada resolución con el valor de éxito (o fracaso), y se puede acceder a cualquier dato acompañante (e.g., un resultado de cómputo guardado por la tarea asincrónica A 204).

[0041] Por lo tanto, como se entiende, si, por ejemplo, el oyente B 206 cancela antes de que la tarea asincrónica A 204 se complete, la cancelación burbujea hacia arriba desde el oyente B 206 hacia la tarea asincrónica compartida 204 (así como burbujas hacia abajo para cancelar la tarea asincrónica C 210). Sin embargo, debido a que el multiplexor 202 envuelve la tarea asincrónica 204 y conoce sus otros oyentes M 207 y X 208, cada uno de los cuales son independientes del oyente B 206 y entre sí, el multiplexor 202 intercepta la petición de cancelación y sabe que sólo el oyente B 206 fue cancelado. El oyente B 206 se elimina así de la lista de oyentes de notificación del multiplexor, sin embargo, el oyente M 207 y el oyente X 208 siguen listados como interesados en el resultado de la tarea asincrónica A 204. Observe que la "lista" puede comprender cualquier estructura de datos adecuada que rastrea la información de identidad de los oyentes y/o que "elimina" el oyente cancelado no tiene que ser una eliminación de tipo borrado real, sino que, por ejemplo, puede marcar la información de identidad para el oyente que ha sido cancelado, o realizar alguna acción similar. Dado que la cancelación de la tarea asincrónica compartida A es detenida por el multiplexor 202, la tarea asincrónica compartida A 204 se deja en cola, se ejecuta y se completa, y el oyente M 207 y el oyente X 208 todavía se notifican apropiadamente cuando se completa la tarea asincrónica compartida A 204.

[0042] La tarea asincrónica A 204 se puede cancelar de otras maneras, incluyendo directamente; por ejemplo, el objeto o similar que solicitó la tarea asincrónica A 204 puede cancelarla, tal como si el proceso es una instancia de objeto que se está eliminando y necesita cancelar sus tareas o si el objeto quiere volver a ejecutar una nueva tarea asincrónica A 204. Otra forma en que una tarea asincrónica compartida se puede cancelar es por una tarea asincrónica superior que burbujea hacia abajo su cancelación hasta la tarea asincrónica compartida, e.g., si la tarea asincrónica compartida A dependía de alguna tarea asincrónica A0 y la tarea asincrónica A0 cancela, también lo hace la tarea asincrónica A porque la tarea asincrónica A no se puede completar correctamente. Además, otra manera de cancelar la tarea asincrónica 204 es la mencionada anteriormente, es decir, cuando ya no hay oyentes que dependen de la tarea asincrónica 204. Por ejemplo, si cada uno de los oyentes 206-208 se cancela de forma independiente, luego cuando no hay más oyentes, la última cancelación básicamente se permite burbujear hacia arriba hasta cancelar la tarea asincrónica A 204.

[0043] La FIG. 3 es un diagrama de flujo que muestra algún ejemplo de lógica de un multiplexor. Las etapas 302 y 304 tienen lugar antes de que la tarea asincrónica compartida que se envuelve como un multiplexor esté en cola. La etapa 302 representa la verificación de una solicitud de cancelación, que si se recibe se maneja mediante las etapas de ejemplo de la FIG. 4, que se describe a continuación. La etapa 304 representa la verificación de cuándo se va a poner en cola la tarea asincrónica compartida, que si se cumple, se ramifica a la etapa 306 para hacer que la llamada a cola de la tarea asincrónica compartida a ejecutar, e.g., llama al procesador de tareas 112 para poner en cola la tarea de ejecución cuando el procesador de tareas 112 puede ejecutarlo. Como se entiende, aunque se muestra un bucle desde la etapa 304 de vuelta a la etapa 302 con fines de ilustración, el código de programa no bloquea esperando la cancelación o cola.

[0044] Una vez en cola, las etapas 306 y 308 representan un bucle "en espera" para que la tarea asincrónica multiplexada compartida se complete o se cancele, respectivamente, aunque como se entiende, la espera no bloquea, de modo que otro trabajo se puede realizar por el código de programa durante el estado pendiente de la tarea asincrónica multiplexada compartida. Si la tarea asincrónica multiplexada compartida se completa como se evaluó en la etapa 306, el multiplexor notifica al oyente(s) del resultado en la etapa 310. Si en su lugar hay una solicitud de cancelación en la etapa 308 con respecto a la tarea asincrónica multiplexada compartida, el multiplexor maneja la petición de cancelación, por ejemplo, usando el ejemplo lógico de la FIG. 4.

[0045] En la FIG. 4, La etapa 402 evalúa si la solicitud de cancelación es aquella que ha burbujado hacia arriba desde un oyente. Si no es así, la petición de cancelación es directa, e.g., desde la instancia de objeto a la que pertenece la tarea asincrónica multiplexada compartida y los oyentes/cadenas o desde una tarea superior (e.g., si la tarea asincrónica multiplexada es un oyente dependiente o de otro modo una tarea dependiente de otra tarea que fue cancelada) y por lo tanto la tarea asincrónica multiplexada compartida se cancela en la etapa 408. Observe que esta cancelación también cancela los oyentes inferiores de la tarea asincrónica, que a su vez pueden cancelar sus oyentes inferiores y/o tareas dependientes. Además, la cancelación de la tarea asincrónica multiplexada puede burbujear hacia arriba hasta una tarea asincrónica "superior", e.g., si la tarea asincrónica multiplexada compartida era un oyente/tarea en una cadena de dependencia, sin embargo, la "tarea asincrónica superior", si es un multiplexor, no se puede cancelar sino eliminar la tarea asincrónica cancelada de su lista de oyentes.

[0046] Si la solicitud de cancelación es aquella que ha burbujeado hacia arriba desde un oyente, la etapa 404 elimina ese oyente de la lista de aquellos a notificar. Cualquier confirmación o similar que el oyente ha cancelado necesita también poder ser devuelto en este momento.

[0047] La etapa 406 representa evaluar si existe al menos un oyente independiente restante. Si no es así, la tarea asincrónica multiplexada compartida se cancela en la etapa 408, (que, como se describió anteriormente, también puede burbujear hacia arriba la cancelación). Si queda al menos uno, la tarea asincrónica compartida no se cancela, y el proceso vuelve a la FIG. 3. Si la solicitud de cancelación se recibió "pre-cola", antes de que la tarea compartida fuera puesta en cola, el proceso de la FIG. 4 vuelve a la etapa 302 hasta que la tarea asincrónica multiplexada compartida se pone en cola o se recibe otra solicitud de cancelación. Si la solicitud de cancelación se recibió "después de la cola", después de que la tarea compartida se puso en cola, el proceso de la FIG. 4 vuelve a la etapa 308 hasta que se completa la tarea asincrónica multiplexada compartida o se recibe otra solicitud de cancelación.

[0048] Es factible tener una situación en la que una tarea es compartida por los oyentes, pero también se desea ejecutar independientemente de si tiene oyentes, e.g., el código desea ejecutar la tarea J, también J.luego(K) y J.luego(L). Esto puede ser optimizado para ejecutar la tarea compartida J con los oyentes K y L. Si K y L cancelan, sin embargo, entonces lo que se compartió de la tarea J no funcionará por sí mismo; para evitar esto, el optimizador puede reconocer esta situación y, por ejemplo, puede marcar J de alguna manera para que el multiplexor no cancele la tarea compartida J con base en la cancelación de todos sus oyentes, tal como para añadir un oyente NULL o los similares que no puede cancelar a

menos que J se cancele directamente, e.g., ejecute la tarea compartida J con los oyentes K, L y NULL.

[0049] Como se entiende, el multiplexado como se describe aquí es muy útil, incluyendo, porque la multiplexación permite abstraer el hecho de que diferentes oyentes pueden estar compartiendo el mismo trabajo asincrónico. Los oyentes no son conscientes de que la optimización del multiplexor está ocurriendo, y además, los oyentes no se pueden impactar entre sí.

[0050] Volviendo a otro aspecto, a saber, la limpieza automática de tareas asincrónicas, considera que hay una pieza de código que arranca una cadena de tareas asincrónicas. Cada vez que se ejecuta este código, el código hace la misma cadena asincrónica. Cada vez que se llama a la cadena asincrónica, es deseable cancelar primero la cadena asincrónica original (si está pendiente, es decir, tener al menos una tarea no completada), de modo que no se esté haciendo más trabajo de lo necesario y de modo que el resultado de la antigua cadena asincrónica no se tome accidentalmente.

[0051] Como otro ejemplo, si el código que ejecuta una cadena tiene algún concepto de vida asociado con el código, cuando el código ya no es necesario, también necesita cancelar cualquiera de sus cadenas pendientes. Por ejemplo, al eliminar una instancia de un objeto, las cadenas pendientes de la instancia también necesitan cancelación, al menos para evitar consumir innecesariamente recursos haciendo un trabajo innecesario.

[0052] Un problema con la anulación del trabajo asincrónico es que el acto de cancelar también es asincrónico (como es generalmente todo lo demás con estas tareas). En otras palabras, una solicitud de cancelación también es una tarea que

está en cola para ejecutarse cuando el procesador de la tarea 112 llega a esa tarea de cancelación. A menudo es la situación antes de que la tarea de solicitud de cancelación se deshace y procese, parte de la cadena de tareas asincrónicas se resuelve con éxito y está a punto de ejecutar más de la cadena, lo que puede desencadenar otro trabajo y/o causar otros problemas.

[0053] A modo de ejemplo, considere que existe una cadena de dependencia asincrónica tal como $A \rightarrow B \rightarrow C \rightarrow D$. Considérese además que la cadena acaba de terminar la tarea B satisfactoriamente, y por lo tanto la tarea en cola C cuando la cancelación se llama para la cadena. Debido a que la tarea C se puso en cola primero, la tarea C se deshace y se ejecuta, aunque se solicitó que se cancelara y ya no se desea ejecutar. Aunque la tarea de solicitud de cancelación se procesará después de la tarea C, y así evitar que ocurra la tarea D, todavía se desea detener la tarea C. Por ejemplo, recuerde que una cadena se puede cancelar para iniciar una nueva versión de ella, la nueva tarea A se está ejecutando en conjunción con la antigua tarea C, por ejemplo, puede suceder cosas impredecibles (y probablemente a veces malas), por lo que dicho código de programa no es robusto.

[0054] La tecnología descrita en el presente documento y representada en la FIG. 5 envuelve las tareas en el código que primero verifica ver si la cancelación fue llamada para esa tarea/su cadena. Para este fin, dado un conjunto ordenado 552 de tareas asincrónicas [A, B, C,...] (etiquetadas 554A-554C en la FIG. 5) que no están todavía enlazadas, una cadena 556 $S(A) \rightarrow S(B) \rightarrow S(C)...$ (etiquetados 554S(A)-554S(C) en la FIG. 5) se crea mediante un proceso de envoltura 558, donde S comprende el código que verifica la cancelación antes de ejecutar su tarea.

[0055] De este modo, independientemente de si una tarea se pone en cola antes de la solicitud de cancelación que está en cola para cancelarla, cuando se retira de la cola y se ejecuta, el código de verificación de cancelación envuelto añadido verifica con el código del programa para ver si está destinado a ser cancelado. Esto se representa generalmente en la FIG. 1 por la tarea de ejecución 116 que verifica de nuevo con el código del programa 102 (entidad solicitante 104 (a)) para verificar su estado cancelado o no; (esta información puede almacenarse en cualquiera de varias ubicaciones posibles, siempre y cuando la tarea sepa dónde buscarla). Si en el estado cancelado, la tarea se cancela en si misma no realizando ninguna de sus tareas, evitando así problemas potenciales con una nueva cadena de tareas. Por ejemplo, tal tarea auto-cancelada se puede completar con un estado de fallo, aunque se puede devolver otro estado para indicar que su finalización no tuvo éxito debido a esta condición auto-cancelada. Tenga en cuenta que el código de verificación de cancelación no tiene por qué omitir completamente el código de tarea, por ejemplo, ya que en su lugar puede saltar a la parte del código de tarea que devuelve el estado de falla (u otro).

[0056] La FIG. 6A ejemplifica el procesamiento de tareas generales, comenzando en la etapa 602, donde una tarea se deshace. La etapa 604 ejecuta la tarea (la lógica de tarea se muestra en la FIG. 6B) y la etapa 606 devuelve el resultado.

[0057] En la FIG. 6B, a través de las etapas 622 y 624, la tarea verifica primero si ha sido cancelada, e.g., la etapa 622 se comunica con su entidad solicitante o verifica una ubicación acordada para dichos datos de estado. Si se cancela, la etapa 624 se ramifica a la etapa 626, donde el estado se establece como falla (o algún otro valor que indica que la tarea no se completó satisfactoriamente). De lo

contrario, la tarea realiza sus operaciones y devuelve cualquier estado real resultante.

[0058] Para manejar los escenarios de cancelación automáticos descritos en el presente documento, se puede almacenar una cadena en la instancia de objeto o similar que ejecuta la cadena. A cada cadena se le da un nombre único por su instancia, e.g., "foo". Por lo tanto, si intenta ejecutar un nuevo "foo" mientras un viejo "foo" sigue funcionando, el viejo "foo" se cancela. Si se elimina la instancia y es hora de limpiar, cada una de las cadenas pendientes de la instancia se cancela.

[0059] La FIG. 7 es un diagrama de flujo que contiene etapas de ejemplo relacionadas con el funcionamiento de una cadena. La etapa 702 representa un objeto que decide ejecutar una determinada cadena de tareas, que, por ejemplo, puede estar con base en cualquier número de disparadores, e.g., interacción del usuario, un temporizador, una condición que se cumple, una decisión que se hace con base en algunos datos, y así sucesivamente. Se pueden ejecutar múltiples cadenas de tareas al mismo tiempo y las cadenas de tareas pueden formar parte de un conjunto optimizado de cadenas que comparten al menos una tarea; sin embargo, a efectos de explicación, consideran que la FIG. 7 se refiere a una cadena designada de un conjunto de una o más cadenas.

[0060] La etapa 704 evalúa si todavía se está ejecutando una cadena antigua del mismo nombre y, en caso afirmativo, se anula la cadena antigua (etapa 706), como se describió anteriormente. Esto puede incluir burbujear la cancelación a las otras tareas de la cadena antigua, incluyendo a cualquier multiplexor(s), como también se describió anteriormente.

[0061] La etapa 708 selecciona la primera tarea de la cadena y la etapa 710 llama a hacer cola esta tarea. Tenga en cuenta que si esta tarea se compartió previamente y se completó correctamente y si alguno de sus resultados sigue estando disponible y es válido con respecto a la nueva cadena, se puede utilizar el resultado anterior en lugar de volver a ejecutar la tarea.

[0062] La etapa 712 evalúa si la tarea se completa satisfactoriamente. Si no es así, la etapa 714 cancela el resto de la cadena. De lo contrario, las etapas 716 y 718 se mueven a la siguiente tarea de la cadena y así sucesivamente para que cada tarea se ejecute hasta que no quede nada o se produzca algo distinto de un éxito, e.g., una cancelación o un fallo. La etapa 720 representa el uso de los resultados de alguna manera deseada; e.g., una tarea de búsqueda de una imagen, otra tarea de texto buscado, otra tarea de formateo de un elemento UI con el texto y la imagen, otra tarea reproduce el elemento UI, y ahora el elemento UI puede interactuar con un usuario a través de otro código en el objeto.

[0063] La FIG. 8 resume algunas de las etapas relacionadas con tareas y cadenas, comenzando en la etapa 802 donde el objeto (o desarrollador) decide qué tareas se van a realizar, y nombra cualquier cadena. La etapa 804 envuelve las tareas con el código de manejo de cancelación como se describió anteriormente con referencia a las FIGS. 5, 6A y 6B. La etapa 806 optimiza las cadenas, e.g., comparte y multiplexa cualquiera de las tareas compartibles, e.g., ejemplificadas aquí con referencia a la FIG. 10. Observe que algunas o todas de estas etapas se pueden hacer antes del tiempo de ejecución, e.g., un objeto puede ser predefinido con sus tareas y cadenas con nombre, cada tarea envuelta con el código de manejo de cancelación y cualquier optimización realizada antes de que el objeto sea instanciado. Alternativamente, un objeto puede hacer al menos algunos de estos actos en tiempo de ejecución, e.g., un objeto puede tener

muchas cadenas diferentes y decidir cuál de las muchas combinaciones posibles de cadenas para ejecutar con base en algún conjunto actual de una o más condiciones, en cuyo caso una combinación dada puede tener que ser elegida dinámicamente para un conjunto de condiciones actuales y optimizada después de la elección.

[0064] Las FIGS. 9A y 9B están dirigidas hacia otro aspecto de la cancelación, e.g., si las especificaciones de la materia de verificación de cancelación, la cancelación se puede hacer asociando un número de "versión" con cada cadena. Como se representa en la FIG. 9A, cada vez que se llama cancelar a la cadena (etapa 902), se incrementa el número de versión para esa cadena (etapa 904). Como se representa en la FIG. 9B, cada vez que se llama a ejecución (etapa 924), se guarda el número de versión en ese momento (etapa 922). Al ejecutar las tareas de la cadena, en la etapa 926 se compara la versión actual con la versión de inicio y si no coinciden, el trabajo se detiene (la cadena se cancela) en la etapa 928. De lo contrario, la cadena continúa ejecutando sus tareas hasta completarse (etapa 930), o a través de la FIG. 9A se llama una cancelación (etapa 902) que cambia el número de versión (etapa 904) durante la ejecución de las tareas de la cadena.

[0065] Las FIGS. 10-12 se dirigen hacia un ejemplo de optimización de cadenas, en las que la FIG. 10 comprende etapas de optimización de ejemplo, y las FIGS. 11 y 12 son representaciones de cadenas de tareas que se convierten en un árbol de tareas que tienen tareas compartidas. Observe que hay muchas maneras de realizar la optimización, y la que se ejemplifica en la FIG. 10 es sólo una de estas formas. Además, tenga en cuenta que la FIG. 11 muestra cadenas no optimizadas 1178 transformadas por un optimizador 1180 en cadenas parcialmente optimizadas 1182, que se transforman adicionalmente en cadenas

1282 totalmente optimizadas en la FIG. 12. Observe también que la única tarea J se puede considerar una cadena aunque no tenga entidades dependientes.

[0066] La etapa 1002 selecciona el primer conjunto (más alto) de tareas, e.g., moviéndose de izquierda a derecha en las cadenas, que como se ve en las FIGS. 11 y 12 es el conjunto [A A A A J]. La etapa 1004 evalúa si hay alguna tarea compatible en este conjunto, que como se puede ver, incluye la tarea A en este ejemplo. La etapa 1006 selecciona tal tarea compatible y la etapa 1008 multiplexa esta tarea para compartir, e.g., envolviendo la tarea en código (e.g., la FIG. 4) que impide que la cancelación de un oyente impacte a cualquier otro oyente independiente. En la FIG. 11, esto está representado por el cuadro "Mux" alrededor de la tarea seleccionada A.

[0067] La etapa 1010 añade el siguiente nivel de tareas, e.g., sus identificadores (ID) o similares, a la lista de oyentes a notificar. En el ejemplo de las FIGS. 11 y 12, esta lista (por ahora) es el conjunto [B B F H]. La etapa 1012 hace que los elementos de este conjunto escuchen, e.g., eliminando básicamente las tareas comunes no seleccionadas de sus cadenas y enlazando la tarea compartida seleccionada a las cadenas, tal como se representa generalmente por la "salida X" tareas no seleccionadas A, y por las flechas de la tarea compartida A al siguiente nivel de tareas, el conjunto [B B F H]. Tenga en cuenta que si una de las tareas A no formaba parte de ninguna cadena, la etapa 1010 o 1012 puede agregar el oyente "NULL" o de alguna otra forma marcar el multiplexor de alguna manera para que la tarea A se ejecute incluso si todos sus otros oyentes dependientes cancelan.

[0068] La etapa 1014 repite el proceso para otras tareas de nivel superior compatibles. Obsérvese que no se muestran ninguno en el ejemplo de las FIGS.

11 y 12, sin embargo consideramos que el conjunto en otro escenario era [A A A A J V V V W W] (en lugar de [A A A A J]). Como se entiende, tales tareas V y W son también compartibles en este ejemplo alternativo, y por lo tanto serían procesadas de la misma manera, repitiéndose a través de la etapa 1014, primero para las tareas V y luego para las tareas W.

[0069] Una vez que no hay más tareas compartibles a este nivel, el proceso continúa a la etapa 1016 donde se evalúa el siguiente nivel, correspondiente al conjunto [B B F H]. Como puede verse, las tareas B (oyentes) se pueden compartir, y por lo tanto estas se seleccionan a través de la etapa 1016. El proceso se repite así, y tal como se representa en la FIG. 12 a través del bloque 1282, las tareas B se multiplexan y comparten. Tenga en cuenta que el conjunto de oyentes de la tarea A se actualiza para reflejar que sólo hay un oyente B, que puede ser parte de este proceso de optimización o realizado por separado.

[0070] Se debe observar que no siempre existe la necesidad de multiplexar una tarea compartida, aunque esto no se describe expresamente en la lógica de la FIG. 10. Por ejemplo, considere las tres cadenas $A \rightarrow B \rightarrow C$ y $A \rightarrow B \rightarrow D$ y $A \rightarrow B \rightarrow E$. Si no hay otras cadenas a optimizar, la tarea A no necesita ser multiplexada porque si el oyente cancela la tarea B, la tarea compartida A no tiene oyentes y por lo tanto también cancela. La tarea compartida B se debe multiplexar, de modo que la cancelación de la tarea C, D o E no afecte a las otras tareas no canceladas. Aunque no hay ningún inconveniente en tener la tarea compartida A multiplexada con un solo oyente B, si se desea, un proceso separado puede buscar tales situaciones y eliminar el código envuelto de multiplexación para la tarea A o las etapas de ejemplo de la FIG. 10 se pueden modificar para reconocer y hacer frente a esta situación.

[0071] Como puede verse, la tecnología aquí descrita permite el uso de cadenas optimizadas (e.g., cadenas prometedoras) y árboles. Mediante la multiplexación y el código de verificación de cancelación envuelto, las dependencias en tareas asincrónicas funcionan bien y de manera robusta.

[0072] Uno o más aspectos están dirigidos a multiplexar una tarea asincrónica que se identifica en al menos dos cadenas de tareas, incluyendo compartir resultados de la tarea asincrónica entre un conjunto de oyentes que comprende al menos dos oyentes dependientes de nivel inferior. La tecnología incluye el mantenimiento de la información de identidad de cada oyente de nivel inferior dependiente en asociación con la tarea asincrónica, e impide la cancelación de un oyente de cancelar la tarea asincrónica, siempre y cuando por lo menos un oyente permanezca dependiente de la tarea asincrónica con base en la identidad de información.

[0073] La prevención de la cancelación de un oyente de cancelar la tarea asincrónica puede comprender recibir una solicitud de cancelación correspondiente a un oyente de nivel inferior cancelado, acceder a la información de identidad y modificar la información de identidad incluyendo eliminar el oyente de nivel inferior cancelado del conjunto de los oyentes que dependen de la tarea asincrónica. La información de identidad se puede usar para determinar que al menos un oyente no cancelado permanece dependiente de la tarea asincrónica.

[0074] Cada oyente del conjunto de oyentes se puede notificar al término de la tarea asincrónica. Además, cada oyente del conjunto de oyentes puede estar en cola para ejecutar otra tarea asincrónica.

[0075] Las cadenas de tareas se pueden optimizar para que una tarea asincrónica en cada cadena de tareas sea una tarea compartida. El código de tarea asincrónica que realiza la tarea asincrónica puede estar envuelto en el código de cancelación de verificación. La ejecución de la tarea asincrónica puede incluir la ejecución del código de cancelación de verificación para determinar si la tarea asincrónica es destinada a ser cancelada, y si está destinada a ser cancelada, no ejecutar al menos parte del código de tarea asincrónica.

[0076] El oyente puede ser una tarea de oyente que incluye un código de tarea de oyente que realiza la tarea de oyente. El código de tarea de oyente puede estar envuelto en un código de verificación de cancelación que verifica, antes de que el código de tarea de oyente se ejecute, si la tarea de oyente está destinada a ser cancelada y si se pretende cancelar, no permite ejecutar al menos parte del código de tarea de oyente.

[0077] Una cadena de tareas puede estar asociada con un nombre. Una cadena de tareas pendientes se puede cancelar en función del nombre asociado con la cadena de tareas pendiente. Una cadena de tareas pendientes se puede cancelar con base en el nombre asociado con la cadena de tarea pendiente que coincide con el nombre de una nueva cadena de tareas que se ejecutará en lugar de la cadena de tareas pendiente.

[0078] Una cadena de tareas puede estar asociada con un nombre y un número de versión de cancelación actual que se basa en una llamada de cancelación para esa cadena de tareas. El número de versión de cancelación actual se puede guardar como un número de versión de cancelación guardado en asociación con el nombre. Mientras se ejecuta una cadena de tareas con un nombre coincidente, el número de versión de cancelación guardado se evalúa con

un número de versión de cancelación actual; la cadena de tareas se cancela si el número de versión de cancelación guardado no coincide con el número de versión de cancelación actual.

[0079] Uno o más aspectos están dirigidos hacia una entidad solicitante que incluye una primera cadena de tareas que tiene una tarea asincrónica compartida que se comparte con una segunda cadena de tareas, incluyendo proporcionar resultados de ejecutar satisfactoriamente la tarea asincrónica compartida a un oyente de la primera cadena de tareas y a un oyente de la segunda cadena de tareas. Un multiplexor mantiene una asociación de información de identidad del oyente de la primera cadena de tareas con la tarea asincrónica y la información de identidad del oyente de la segunda cadena de tareas con la tarea asincrónica. El multiplexor está configurado para notificar al oyente de la primera cadena de tareas y al oyente de la segunda cadena de tareas cuando se completa la tarea asincrónica. El multiplexor está configurado además para evitar la cancelación del oyente de la segunda cadena de tarea de cancelar la tarea asincrónica si el oyente de la primera cadena de tareas no se ha cancelado y para impedir que el oyente de la primera cadena de tarea cancele la tarea asincrónica si el oyente de la segunda cadena de tareas no ha cancelado.

[0080] La entidad solicitante puede ser una instancia de objeto. Un optimizador puede procesar una tarea asincrónica de la primera cadena de tareas y una tarea asincrónica de la segunda cadena de tareas en la tarea asincrónica compartida.

[0081] Un proceso de envoltura puede envolver al oyente de la primera cadena de tareas con código de verificación de cancelación que detiene al menos alguna tarea del oyente de la primera cadena de tareas si el oyente de la primera cadena de tareas está destinado a ser cancelado. El proceso de envoltura puede envolver

al oyente de la segunda cadena de tareas con código de cancelación de verificación que detiene al menos alguna tarea del oyente de la segunda cadena de tareas si el oyente de la segunda cadena de tareas está destinado a ser cancelado.

[0082] Uno o más aspectos están dirigidos a mantener una identidad de un primer oyente y una identidad de un segundo oyente en un conjunto de oyentes de uno o más oyentes asociados con una tarea asincrónica compartida. Al recibir una solicitud de cancelación correspondiente a la cancelación del segundo oyente, con base en la solicitud de cancelación, descrita en el presente documento está eliminando la identidad del segundo oyente del conjunto de oyentes asociados con la tarea asincrónica compartida. También se describe aquí si se determina si al menos un oyente permanece en el conjunto de oyentes y, de ser así, ejecutando la tarea asincrónica compartida y notificando a cada oyente en el conjunto de oyentes una vez completada la tarea asincrónica.

[0083] Cuando la determinación de si al menos un oyente permanece en el conjunto de oyentes determina que no hay oyentes, la tarea asincrónica se puede cancelar.

[0084] La tarea asincrónica se puede envolver con código de cancelación de verificación que detiene al menos un cierto trabajo de tarea de la tarea asincrónica si se pretende cancelar la tarea asincrónica. El primer oyente puede estar envuelto con código de cancelación de verificación que detiene al menos un trabajo de tarea del primer oyente si se pretende cancelar el primer oyente.

[0085] Mantener la identidad del primer oyente y la identidad de un segundo oyente en el conjunto de oyentes puede incluir la multiplexación de la tarea

asincrónica compartida, incluida la envoltura de la tarea asincrónica compartida con el código multiplexador 1) para eliminar la identidad del segundo oyente del conjunto de oyentes asociados con la tarea asincrónica compartida, y 2) para determinar si al menos un oyente permanece en el conjunto de oyentes.

EJEMPLO DISPOSITIVO DE COMPUTACIÓN

[0086] Las técnicas descritas aquí se pueden aplicar a cualquier dispositivo o conjunto de dispositivos (máquinas) capaces de ejecutar programas y procesos. Se puede entender, por lo tanto, que se contemplan computadores personales, computadores portátiles, dispositivos portátiles, portátiles y otros dispositivos informáticos y objetos informáticos de todo tipo, incluyendo teléfonos celulares, computadores de tableta/pizarra, consolas de juegos y entretenimiento y similares para uso en conexión con diversas implementaciones incluyendo aquellas ejemplificadas en este documento. Por consiguiente, el mecanismo de cómputo de propósito general descrito más adelante en la FIG. 13 es sólo un ejemplo de un dispositivo informático.

[0087] Las implementaciones se pueden implementar parcialmente a través de un sistema operativo, para ser utilizadas por un desarrollador de servicios para un dispositivo u objeto, y/o incluidas en el software de aplicación que funciona para realizar uno o más aspectos funcionales de las diversas implementaciones descritas en el presente documento. El software se puede describir en el contexto general de las instrucciones ejecutables por computador, tales como módulos de programa, que son ejecutados por uno o más computadores, tales como estaciones de trabajo para cliente, servidores u otros dispositivos. Los expertos en el arte apreciarán que los sistemas informáticos tienen una variedad de

configuraciones y protocolos que se pueden utilizar para comunicar datos y, por lo tanto, ninguna configuración o protocolo particular se considera limitante.

[0088] La FIG. 13 ilustra, por tanto, un ejemplo de un entorno del sistema informático 1300 adecuado en el que uno o varios aspectos de las implementaciones descritas en el presente documento se pueden realizar, aunque, como se ha explicado anteriormente, el entorno del sistema informático 1300 es sólo un ejemplo de un ambiente de computación adecuado y no pretende sugerir ninguna limitación en cuanto al ámbito de uso o funcionalidad. Además, el entorno del sistema informático 1300 no está destinado a ser interpretado como que tiene ninguna dependencia relacionada con una o una combinaciones de componentes ilustradas en el entorno del sistema informático 1300 de ejemplo.

[0089] Haciendo referencia a la FIG. 13, un dispositivo de ejemplo para implementar una o más implementaciones incluye un dispositivo de cómputo de propósito general en forma de un computador 1310. Los componentes del computador 1310 pueden incluir, pero no están limitados a, una unidad de procesamiento 1320, una memoria de sistema 1330 y un sistema de bus 1322 que acopla varios componentes del sistema que incluyen la memoria del sistema a la unidad de procesamiento 1320.

[0090] El computador 1310 incluye típicamente una variedad de medios legibles de máquina (e.g., computador) y puede ser cualquier medio disponible al que pueda acceder una máquina tal como el computador 1310. La memoria de sistema 1330 puede incluir medios de almacenamiento de computador en la forma de memoria volátil y/o no volátil, tal como memoria de sólo lectura (ROM) y/o memoria de acceso aleatorio (RAM), y medios de disco duro, medios de almacenamiento ópticos, medios de flash, y así sucesivamente; como se usa en el

presente documento, los medios de almacenamiento legibles por máquina/legibles por computador almacenan datos que no incluyen señales transitorias (aunque se pueden usar otros tipos de medios legibles por máquina/legibles por computador que no son medios de almacenamiento). A modo de ejemplo, y no limitación, la memoria de sistema 1330 también puede incluir un sistema operativo, programas de aplicación, otros módulos de programa y datos de programa.

[0091] Un usuario puede introducir comandos e información en el computador 1310 a través de uno o más dispositivos de entrada 1340. Un monitor u otro tipo de dispositivo de visualización está también conectado al sistema de bus 1322 a través de una interfaz, tal como la interfaz de salida 1350. Además de un monitor, los computadores también pueden incluir otros dispositivos de salida periféricos, como altavoces y una impresora, que se pueden conectar a través de la interfaz de salida 1350.

[0092] El computador 1310 puede funcionar en un entorno en red o distribuido utilizando conexiones lógicas a uno o más computadores remotos, tal como el computador remoto 1370. El computador remoto 1370 puede ser un computador personal, un servidor, un enrutador, una red PC, un dispositivo parecido u otro nodo de red común, o cualquier otro dispositivo de consumo o transmisión de medios remoto, y puede incluir cualquiera o todos los elementos descritos anteriormente con relación al computador 1310. Las conexiones lógicas representadas en la FIG. 13 incluyen una red 1372, tal como una red de área local (LAN) o una red de área extensa (WAN), pero también pueden incluir otras redes/buses. Tales entornos de red son comunes en hogares, oficinas, redes informáticas de toda la empresa, intranets e Internet.

[0093] Como se mencionó anteriormente, aunque se han descrito implementaciones de ejemplo en conexión con diversos dispositivos de cómputo y arquitecturas de red, los conceptos subyacentes se pueden aplicar a cualquier sistema de red y cualquier dispositivo o sistema informático en el que sea deseable implementar dicha tecnología.

[0094] También hay múltiples formas de implementar la misma o similar funcionalidad, e.g., una API apropiada, un conjunto de herramientas, un código de controlador, un sistema operativo, un control, un objeto de software independiente o descargable, etc., que permite que las aplicaciones y los servicios tomen ventaja de las técnicas aquí proporcionadas. De este modo, las implementaciones de la presente invención se contemplan desde el punto de vista de una API (u otro objeto de software), así como de un objeto de software o hardware que implementa una o más implementaciones como se describe en el presente documento. De este modo, varias implementaciones descritas aquí pueden tener aspectos que están completamente en hardware, en parte en hardware y parcialmente en software, así como completamente en software.

[0095] La palabra "ejemplo" se usa aquí para significar servir como un ejemplo, instancia o ilustración. Para evitar dudas, la materia objeto de la presente invención no está limitada por tales ejemplos. Además, cualquier aspecto o diseño descrito en el presente documento como "ejemplo" no se debe interpretar necesariamente como preferente o ventajoso con respecto a otros aspectos o diseños, ni pretende excluir estructuras de ejemplo equivalentes y técnicas conocidas por los versados en el arte. Además, en la medida en que los términos "incluye", "tiene", "contiene" y otras palabras similares se usan, para evitar dudas, dichos términos tienen la intención de ser inclusivos de una manera similar al término "que comprende" como una palabra de transición abierta sin excluir

cualquier elemento adicional u otros elementos cuando se emplea en una reivindicación.

[0096] Como se ha mencionado, las diversas técnicas descritas aquí se pueden implementar en conexión con hardware o software o, cuando sea apropiado, con una combinación de ambos. Tal como se utiliza en el presente documento, los términos "componente", "módulo", "sistema" y similares se refieren asimismo a una entidad relacionada con el computador, hardware, una combinación de hardware y software, software o software en ejecución. Por ejemplo, un componente puede ser, pero no está limitado a ser, un proceso que se ejecuta en un procesador, un procesador, un objeto, un ejecutable, un subproceso de ejecución, un programa y/o un computador. A modo de ilustración, tanto una aplicación que se ejecuta en un computador como en el computador puede ser un componente. Uno o más componentes pueden residir dentro de un proceso y/o subproceso de ejecución y un componente se puede localizar en un computador y/o distribuir entre dos o más equipos.

[0097] Los sistemas mencionados anteriormente se han descrito con respecto a la interacción entre varios componentes. Se puede apreciar que tales sistemas y componentes pueden incluir aquellos componentes o subcomponentes especificados, algunos de los componentes o subcomponentes especificados, y/o componentes adicionales, y según diversas permutaciones y combinaciones de los anteriores. Los subcomponentes también se pueden implementar como componentes acoplados comunicativamente a otros componentes en lugar de incluirse en componentes primarios (jerárquicos). Además, se puede observar que uno o más componentes se pueden combinar en un único componente que proporciona funcionalidad agregada o dividido en varios subcomponentes separados, y que se puede proporcionar una o más capas intermedias, tal como

una capa de gestión, a medios comunicativamente se unen a tales subcomponentes para proporcionar una funcionalidad integrada. Cualquier componente descrito en el presente documento también puede interactuar con uno o más de otros componentes no descritos específicamente en el presente documento pero generalmente conocidos por los versados en el arte.

[0098] A la vista de los sistemas de ejemplo descritos en el presente documento descriptivo, también se pueden apreciar metodologías que se pueden implementar de acuerdo con la materia descrita, con referencia a los diagramas de flujo/diagramas de flujo de las diversas figuras. Aunque con fines de simplicidad de explicación, las metodologías se muestran y describen como una serie de bloques, se debe entender y apreciar que las diversas implementaciones no están limitadas por el orden de los bloques, ya que algunos bloques pueden ocurrir en órdenes diferentes y/o concurrentemente con otros bloques de lo que se representa y describe en el presente documento. Cuando se ilustra un flujo no secuencial o ramificado a través de diagramas de flujo/diagramas de flujo, se puede apreciar que se pueden implementar otras diversas ramas, trayectorias de flujo y órdenes de los bloques que consiguen el mismo resultado o un resultado similar. Además, algunos bloques ilustrados son opcionales en la implementación de las metodologías aquí descritas.

CONCLUSIÓN

[0099] Aunque la invención es susceptible de diversas modificaciones y construcciones alternativas, ciertas implementaciones ilustradas de esta se muestran en los dibujos y han sido descritas en detalle anteriormente. Se debe entender, sin embargo, que no hay intención de limitar la invención a las formas específicas descritas, sino que, por el contrario, se pretende cubrir todas las

modificaciones, construcciones alternativas y equivalentes que caen dentro del espíritu y alcance de la invención.

[00100] Además de las diversas implementaciones descritas en el presente documento, se debe entender que se pueden utilizar otras implementaciones similares o se pueden hacer modificaciones y adiciones a la implementación o ejecuciones descritas para llevar a cabo la misma función o función equivalente de la implementación correspondiente(s) sin desviarse de ello. Además, múltiples chips de procesamiento o dispositivos múltiples pueden compartir el rendimiento de una o más funciones descritas en el presente documento, y de manera similar, el almacenamiento se puede efectuar a través de una pluralidad de dispositivos. En consecuencia, la invención no se limita a ninguna implementación única, sino que se debe interpretar en amplitud, espíritu y alcance de acuerdo con las reivindicaciones adjuntas.