(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 401 826**

(51) Int. CI.:

G06F 17/30 (2006.01)

(12) TRADUCCIÓN DE PATENTE EUROPEA T3

(96) Fecha de presentación y número de la solicitud europea: **04.02.2009 E 09152079 (1)**

(97) Fecha y número de publicación de la concesión europea: **10.10.2012 EP 2166461**

(54) Título: **Sincronización de información**

(30) Prioridad:

17.09.2008 GB 0817022

(45) Fecha de publicación y mención en BOPI de la traducción de la patente:
24.04.2013

(73) Titular/es:

SAGE TECHNOLOGIES LIMITED (100.0%)
UNIT 3096 CITYWEST BUSINESS PARK
24 DUBLIN, IE

(72) Inventor/es:

MILENCOVICI, TUDOR y
JOUHIER, BRUNO

(74) Agente/Representante:

CARPINTERO LÓPEZ, Mario

DESCRIPCIÓN

Sincronización de información

Esta invención se refiere a un método y un aparato para sincronizar información a través de un conjunto de aplicaciones informáticas.

La información utilizada en una aplicación informática puede adoptar muchas formas, como por ejemplo datos financieros o de contabilidad. La información puede ser información de pedidos, información de facturas, información de existencias o cualquier otro tipo de información financiera que se utilice.

Cada entrada de información se conoce como un objeto y los objetos se almacenan en dispositivos de almacenamiento de datos.

Existe un problema cuando se modifican objetos que están duplicados en varios dispositivos de almacenamiento de datos. La situación ideal es que cualquier modificación se refleje inmediatamente en todas las réplicas, de manera que la información esté actualizada en todas partes. Debido a limitaciones objetivas, tales como la fiabilidad de la conexión y la latencia, la tolerancia a los errores o el deseo de mantener una alta disponibilidad de los objetos, no resulta práctico (y en muchos casos no es necesario) mantener toda la red de aplicaciones actualizada todo el tiempo.

Por lo tanto, se considera que las aplicaciones interactúan e intercambian información con el objetivo de obtener un contenido subyacente coherente. Esta interacción se denomina sincronización.

Un método existente para sincronizar datos utiliza una sincronización de reloj vectorial, que utiliza un algoritmo para generar una ordenación parcial de eventos en un sistema distribuido y detectar violaciones de causalidad. Mensajes entre procesos contienen el estado del reloj lógico del proceso emisor. Un reloj vectorial de un sistema de N procesos es un conjunto de N relojes lógicos, uno por proceso, donde una copia lógica de los cuales se mantiene en cada proceso con las siguientes reglas para las actualizaciones de reloj.

Inicialmente todos los relojes están a cero.

Cada vez que un proceso experimenta un evento interno, incrementa en uno su propio reloj lógico en el vector.

Cada vez que un proceso se prepara para enviar un mensaje, envía su vector completo junto con el mensaje que está enviándose.

Cada vez que un proceso recibe un mensaje, compara el reloj vectorial incluido en el mensaje con su propio reloj vectorial. Si el reloj vectorial incluido en el mensaje es estrictamente mayor que el reloj vectorial del proceso, el proceso acepta el nuevo estado contenido en el mensaje y sustituye su reloj vectorial por el reloj vectorial contenido en el mensaje. Si el reloj vectorial incluido en el mensaje es estrictamente menor que el reloj vectorial del proceso, el proceso ignora el mensaje. Si los dos relojes vectoriales no pueden compararse (cada uno tiene una entrada que es estrictamente mayor que la entrada correspondiente en el otro), el proceso de recepción indica que hay un conflicto y aplica un regla de manejo de conflictos para actualizar su estado.

El reloj vectorial está asociado a cada objeto de un dispositivo de almacenamiento de datos. La sincronización se consigue utilizando el algoritmo descrito anteriormente.

El documento US 2007/0282915 da a conocer un sistema para identificar relaciones causales entre conjuntos de datos en diferentes ordenadores centrales.

Un objeto de la presente invención es proporcionar un método más eficiente de sincronización de conjuntos de datos.

Según la presente invención, se proporciona un aparato y un método como los descritos en las reivindicaciones adjuntas. Otras características de la invención resultarán evidentes a partir de las reivindicaciones dependientes y de la siguiente descripción.

Según un primer aspecto de la presente invención, se proporciona un método de sincronización de al menos una primera y una segunda instancia de un conjunto de información que comprende una pluralidad de entradas, donde dichas primera y segunda instancias están ubicadas en diferentes nodos de un sistema, comprendiendo el método:

- asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

- generar un resumen para cada una de las al menos primera y segunda instancias del conjunto de información,

donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las al menos primera y segunda instancias con otra de las al menos primera y segunda instancias del conjunto de información; y

- sincronizar las al menos primera y segunda instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias;

en el que un par existente formado por un valor de nodo y un valor actual de reloj lógico de una entrada se elimina y se sustituye con un nuevo par formado por un valor de nodo y un valor actual de reloj lógico cuando se modifica dicha entrada.

Puede asociarse un valor de marca de tiempo a cada entrada de datos cuando se modifica dicha entrada, además del par formado por un valor de nodo y un valor actual de reloj lógico.

El valor actual de reloj lógico puede denominarse como un tic.

Preferentemente, la sincronización se lleva a cabo de manera bidireccional, por ejemplo desde la primera instancia hasta la segunda instancia y después desde la segunda instancia hasta la primera instancia.

Preferentemente, la sincronización se lleva a cabo para un conjunto de información a la vez. Esta característica es ventajosa con respecto a la sincronización de una entrada de datos a la vez.

Cada resumen está ubicado preferentemente en un nodo con el que está relacionado.

El resumen puede comprender un par formado por un valor de nodo y un valor de reloj lógico para cada nodo del sistema.

El resumen puede incluir además un valor de prioridad de conflicto para cada nodo del sistema.

El par formado por un valor de nodo y un valor actual de reloj lógico y/o el resumen pueden transmitirse mediante un mecanismo de suministro de datos o un mecanismo de sindicación, tales como un mecanismo RSS o un mecanismo Atom.

Los nodos pueden ser puntos finales que contienen una instancia del conjunto de información. Los valores de nodo pueden ser URL.

Pueden proporcionarse dos resúmenes para cada evento de sincronización, resúmenes que pueden ser un resumen de valores iniciales y un resumen de valores finales, correspondientes al estado de un nodo antes de un evento de modificación y después de un evento de modificación, respectivamente.

La invención se extiende a un dispositivo de sincronización que puede hacerse funcionar para llevar a cabo el método del aspecto anterior.

El dispositivo de sincronización puede ser una aplicación que se ejecute en un sistema informático.

Según otro aspecto de la invención, se proporciona un sistema informático que comprende una pluralidad de nodos, donde cada nodo incorpora una instancia de un conjunto de información, pudiendo hacerse funcionar el sistema informático para sincronizar las instancias del conjunto de información, y que comprende al menos un medio lógico de sincronización que puede hacerse funcionar para:

- asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

- generar un resumen para cada una de las instancias del conjunto de información, donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las instancias con otra de las al menos instancias del conjunto de información; y

- sincronizar las instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias.

El medio lógico de sincronización puede comprender un punto final de sincronización, que puede ser una interfaz o un servicio, para cada nodo o para cada instancia del conjunto de información.

El medio lógico de sincronización puede comprender uno o más motores de sincronización. Los motores de sincronización pueden hacerse funcionar para llevar a cabo intercambios de mensajes entre puntos finales de sincronización. Los motores de sincronización y los puntos finales de sincronización pueden residir en diferentes

ordenadores. Como alternativa, un motor de sincronización puede residir en el mismo ordenador como un punto final de sincronización.

Cada punto final de sincronización puede hacerse funcionar preferentemente para llevar a cabo la asociación y la generación de resúmenes para su nodo dado.

Preferentemente, los motores de sincronización pueden hacerse funcionar para la comunicación con puntos finales para llevar a cabo la función de sincronización.

Los nodos pueden ser aplicaciones informáticas que se ejecutan en un ordenador o pueden ser aplicaciones que se ejecutan en diferentes ordenadores.

Según otro aspecto de la invención, se proporciona un producto de programa informático que comprende instrucciones que, cuando se ejecutan en un ordenador, se hacen funcionar para implementar un método de sincronización de al menos una primera y una segunda instancia de un conjunto de información que comprende una pluralidad de entradas, donde dichas primera y segunda instancias están ubicadas en diferentes nodos de un sistema informático, comprendiendo el método:

- asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

- generar un resumen para cada una de las al menos primera y segunda instancias del conjunto de información, donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las al menos primera y segunda instancias con otra de las al menos primera y segunda instancias del conjunto de información; y

- sincronizar las al menos primera y segunda instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias.

Para un mejor entendimiento de la invención y para mostrar cómo pueden llevarse a cabo realizaciones de la misma, a continuación se hará referencia, a modo de ejemplo, a los dibujos esquemáticos que se acompañan, en los que:

la figura 1 es una vista esquemática de un sistema para sincronizar una pluralidad de dispositivos de almacenamiento de datos;

la figura 2 es una vista esquemática de una disposición alternativa del sistema mostrado en la figura 1, en la que un motor de sincronización está incluido en un nodo;

la figura 3 es una vista esquemática de la misma arquitectura que la figura 2, pero con el origen y el destino invertidos;

la figura 4 es una vista esquemática de un sistema para una sincronización bidireccional de una pluralidad de dispositivos de almacenamiento de datos;

la figura 5 es una vista esquemática de una arquitectura similar a la de la figura 4, pero con un administrador de eventos ubicado en el proveedor B; y

la figura 6 es una vista esquemática que muestra una arquitectura de ejemplo para una sincronización de modo inmediato.

Con el fin de afrontar los problemas relacionados con la sincronización mencionados anteriormente, se han identificado varios atributos para un esquema de sincronización.

La solución debe ajustarse a las necesidades de los diversos casos de uso que surgen en la práctica.

Algunos de los atributos beneficiosos son los siguientes.

Dadas suficientes interacciones entre aplicaciones y en ausencia de modificaciones, debe conseguirse un estado sincronizado entre dispositivos de almacenamiento de datos.

El esquema debe soportar una interacción casual entre nodos.

El sistema utilizado debe permitir una implementación que sea independiente de la topología.

El esquema debe ser robusto y coherente.

El esquema debe ofrecer una sincronización bidireccional.

El esquema debe manejar conflictos entre los estados de los objetos de manera algorítmica y rigurosa.

El esquema debe incluir una solución para el registro y el manejo de errores.

El esquema debe soportar escenarios de sincronización de equiparación e inmediata.

El esquema debe ofrecer independencia de ubicación para un motor de sincronización.

El esquema debe ofrecer un soporte de sincronización parcial.

Debe permitirse la capacidad de resincronización total.

Debe poder conseguirse una implementación sencilla.

Se ha desarrollado un esquema para proporcionar una sincronización de conjuntos de información que tiene en cuenta los objetivos expuestos anteriormente.

1.1 Sincronización de reloj vectorial

Un protocolo de sincronización está basado en una sincronización de "reloj vectorial", una técnica ampliamente conocida a la que se ha hecho referencia anteriormente y utilizada por varios sistemas de sincronización/duplicación: redes P2P, sistemas de archivos duplicados, etc.

Los relojes vectoriales proporcionan una elegante solución al problema de mantener sincronizadas varias copias de un objeto dado y de detectar conflictos.

Suponiendo que un objeto dado O está duplicado en varios nodos N1, N2,…, en un sistema de aplicaciones informáticas conectadas, los relojes vectoriales se construyen de la siguiente forma:

- Cada nodo mantiene un “reloj lógico”: C1 en N1, C2 en N2, etc.

- Cada nodo incrementa su propio tic de reloj cuando O se modifica.

- Un reloj vectorial para O es una lista de pares (N1 T1), (N2 T2),…, donde T1 es el tic de C1 cuando O se modificó por última vez en N1, T2 es el tic de C2 cuando O se modificó por última vez en N2, etc. Si O no se modificó nunca en Ni, su reloj vectorial no contiene ningún par de valores para Ni.

‘O’ puede tener diferentes relojes vectoriales en diferentes nodos. El objetivo del sistema de sincronización es sincronizar O en todos los nodos. Cuando O está totalmente sincronizado, su reloj vectorial es el mismo en todos los nodos.

Comparando los relojes vectoriales de O en dos nodos, el algoritmo de sincronización puede decidir si las versiones pueden sincronizarse (y cómo) o si hay un conflicto. La siguiente tabla explica cómo funciona esta comparación (VOj se refiere al reloj vectorial de O en el nodo j):

Caso	Ejemplo	Estado	Acción
VO1 = VO2	VO1=(N1 5)(N2 7)(N3 8) VO2=(N1 5)(N2 7)(N3 8)	N1 y N2 tienen la misma versión de O	Ninguna
VO1 < VO2	VO1=(N1 5)(N2 7)(N3 8) VO2=(N1 5)(N2 8)(N3 8)	La versión de O de N1 es un antecesor de la versión de N2.	Sustituir la versión de O de N1 por la versión de N2
VO1 > VO2	VO1=(N1 6)(N2 7)(N3 9) VO2=(N1 5)(N2 7)(N3 8)	La versión de O de N2 es un antecesor de la versión de N1.	Sustituir la versión de O de N2 por la versión de N1
No comparables	VO1=(N1 6)(N2 7)(N3 9) VO2=(N1 5)(N2 8)(N3 8)	N1 y N2 tienen un antecesor común pero están en diferentes ramas	Conflicto. Utilizar una regla arbitraria para decidir qué lado gana

Este esquema de sincronización tiene propiedades interesantes:

- Es un algoritmo distribuido. Cada nodo administra su propio reloj y sus propios metadatos de sincronización (los

relojes vectoriales de sus objetos). Cualquier par de nodos puede sincronizarse entre sí sin tener que establecer contacto con otros nodos. No es necesario que haya un coordinador central.

- No limita la topología. La sincronización puede llevarse a cabo a través de subconjuntos arbitrarios (normalmente pares) de nodos. Evidentemente, lo complejo incluye lo sencillo, de manera que se permiten topologías de "estrella", pero también topologías más complejas.

- Consigue una “coherencia final". Si la regla de manejo de conflicto es coherente en todo el sistema, todos los nodos llegarán al mismo estado si la modificación de los datos se detiene y se ejecutan suficientes pasadas de sincronización ("suficientes" depende de la topología).

- Es robusto. Si un nodo se restaura a un estado anterior, no afectará al proceso de sincronización y llegará al estado de los otros nodos (en realidad, si el nodo se había sincronizado después del punto de restauración, los cambios que se produjeron entre el punto de restauración y la última sincronización se recuperarán durante la sincronización después de la restauración del nodo). Por supuesto, esto supone que se realiza una copia de seguridad de los metadatos de sincronización (los relojes vectoriales) junto con los datos.

1.2 Gestión de tics

En la descripción del algoritmo anterior de relojes vectoriales, el tic se incrementa cada vez que se modifica un objeto. En realidad, esto no es un requisito y el algoritmo funciona igual de bien si el tic se incrementa cada vez que se ejecuta una pasada de sincronización en lugar de cada vez que se modifica un objeto. Por tanto, los nodos pueden utilizar cualquiera de las estrategias para administrar sus relojes.

Para ser precisos y para obtener el mismo algoritmo para las dos estrategias, el reloj debe administrarse realmente de la siguiente manera:

- Incrementarse inmediatamente “después” de que se modifique O (el reloj vectorial de O contiene el valor “antes” de que se incremente).

- O alternativamente, incrementarse al “principio” de la pasada de sincronización (de modo que los objetos no se pasan por alto si la pasada examina varios objetos y si los objetos se modifican cuando está ejecutándose la pasada de información).

Con esta convención, el tic puede interpretarse como el “primer valor que no se ha sincronizado” en lugar de el "ultimo valor que se ha sincronizado" cuando se ejecuta la pasada de sincronización. Ésta puede no ser la convención más natural, pero hace que el algoritmo funcione independientemente de si el tic se incrementa en el momento de la modificación o en el momento de la sincronización.

1.3 Resolución de conflictos

El algoritmo utiliza un mecanismo simple basado en prioridades y marcas de tiempo para resolver conflictos. La idea es la siguiente:

- A cada nodo se le proporciona una “prioridad de conflicto”. Normalmente, las prioridades serán estáticas, pero el esquema también soporta escenarios en los que las prioridades se modifican dinámicamente cuando el sistema está en ejecución. Gana el nodo con el valor de prioridad más pequeño.

- Si el conflicto se produce entre nodos de la misma prioridad, se utiliza la marca de tiempo de la modificación del objeto para resolver el conflicto. Gana el nodo en el que el objeto se modificó por última vez.

- En el muy improbable caso en que las marcas de la modificación sean iguales, se aplica alguna regla arbitraria (por ejemplo, gana el id de nodo más pequeño).

La resolución de conflictos puede implementarse opcionalmente añadiendo un valor de prioridad a los pares de valores del reloj vectorial. El reloj vectorial es ahora una lista de tripletes (Ni, Ti, Pi) donde Ni es el id de nodo, Ti es el tic de Ni cuando O se modificó por última vez en Ni, y Pi es la prioridad de conflicto de Ni. Además, los metadatos de sincronización deben contener la marca de tiempo de la última modificación de cada objeto. Por tanto, los metadatos de sincronización para el objeto O son la combinación de su reloj vectorial extendido (lista de tripletes) y la marca de tiempo de su última modificación.

El algoritmo de resolución de conflictos funciona como se describe en la siguiente tabla (la marca de tiempo se reduce a hh:mm para no saturar la tabla):

Caso	Ejemplo	Ganador	Descripción
Las prioridades difieren (estáticas)	VO1=(N1 6 1)(N2 7 2)(N3 9 3) 10:23 VO2=(N1 5 1)(N2 8 2)(N3 8 3) 10:25	VO1	El conflicto implica a N1, N2 y N3. N1 tiene la prioridad de conflicto más baja (1) y VO1 es la versión más reciente para N1
Las prioridades difieren (dinámicas)	VO1=(N1 6 1)(N2 7 2)(N3 9 3) 10:23 VO2=(N1 5 3)(N2 8 2)(N3 8 3) 10:25	VO1	El conflicto implica a N1, N2 y N3. La prioridad de N1 ha cambiado (de 3 a 1 cuando el tic cambió de 5 a 6). En este caso se utiliza la prioridad más reciente de N1. Por tanto, N1 tiene la prioridad de conflicto más baja (1) y VO1 gana.
Las prioridades son idénticas	VO1=(N1 61)(N2 7 1)(N3 9 3) 10:23 VO2=(N1 5 3)(N2 8 1)(N3 8 3) 10:25	VO2	El conflicto implica a N1, N2 y N3. N1 y N3 tienen la prioridad más baja (1). Por tanto se utilizan las marcas de tiempo para resolver el empate. VO2 gana. Nota: el hecho de que la prioridad de N1 sea dinámica en el ejemplo no importa, solo se utiliza la prioridad más reciente.

1.4 Resumen

Normalmente se sincronizan conjuntos de objetos (tablas) en lugar de objetos individuales (registros). Por tanto, el esquema utiliza una variante del algoritmo en la que los nodos realizan un seguimiento de los relojes vectoriales a nivel de "conjunto" en lugar de a nivel de "objeto".

En el nivel de objeto, el algoritmo del esquema sólo realiza un seguimiento de la última modificación: un único par (Ni, Ti) + marca de tiempo de la última modificación.

Esto es menos caro y más fácil de implementar ya que los metadatos de sincronización pueden almacenarse directamente en la tabla de datos (solo se necesitan 3 columnas adicionales para el id de nodo, el tic y la marca de tiempo y, normalmente, la columna de la marca de tiempo ya estará presente) o pueden almacenarse en un tabla de datos sincronizados diferente con una simple correspondencia 1 a 1 con la tabla de datos.

El reloj vectorial a nivel de "conjunto" se denomina "resumen" en la terminología del esquema. El resumen del esquema es realmente un reloj vectorial "extendido" formado por tripletes (id de nodo, tic, prioridad de conflicto).

Esta variante supone que la sincronización se lleva a cabo en un conjunto a la vez en lugar de en un objeto a la vez (aunque posteriormente se muestra que también puede utilizarse para propagar cambios en objetos individuales, bajo determinadas condiciones).

El resumen de un nodo también puede interpretarse como un "compendio" de los estados de sincronización de todos los objetos del conjunto en ese nodo. Por ejemplo, si el resumen para el conjunto S es (N1 6 1)(N2 7 2)(N3 9 3) en el nodo N1, esto significa que el conjunto S de N1 se ha sincronizado con el conjunto S de N2 en el tic 7 (el estado de N1 tiene en cuenta todos los cambios realizados en N2 con tic < 7) y con el conjunto S de N3 en el tic 9 (el estado de N1 tiene en cuenta todos los cambios realizados en N3 con tic < 9).

La sincronización se basa ahora en los siguientes principios:

- Un conjunto puede tener diferentes resúmenes en diferentes nodos. El objetivo del sistema de sincronización es sincronizar el conjunto en todos los nodos. Cuando un conjunto está totalmente sincronizado, su resumen es el mismo en todos los nodos.

- Comparando los resúmenes de un conjunto dado en dos nodos, el algoritmo de sincronización puede seleccionar los objetos que necesitan sincronizarse en ambas direcciones (véase el siguiente apartado 1.5).

- El algoritmo no tiene relojes vectoriales completos a nivel de objeto pero tiene 4 fragmentos de información disponibles cuando sincroniza un objeto: los dos resúmenes + los dos tripletes a nivel de objeto. Esto es suficiente para detectar y resolver conflictos (véase el apartado 1.6).

1.5 Selección de objetos

El esquema utiliza dos pasadas para sincronizar dos nodos Nj y Nk: una pasada para propagar los cambios de Nj a Nk y una segunda pasada en la otra dirección para propagar los cambios de Nk a Nj.

La siguiente tabla describe cómo el algoritmo selecciona objetos de N1 durante la sincronización de N1 a N2, y después selecciona objetos de N2 durante la sincronización en la otra dirección.

Dirección	**Nodo**	**Resumen antes**	**Resumen después**	**Cláusula de selección**	**Comentarios**
N1->N2	N1	(N1 6 1) (N2 7 2) (N3 9 3)	(N1 6 1) (N2 7 2) (N3 9 3)	SELECCIONAR * DE N1.K DONDE (nodo = N1 y tic en [5, 6]) o (nodo = N3 y tic en [8, 9])	Los criterios de selección sólo tienen en cuenta los pares que tienen un tic mayor en N1.
	N2	(N1 5 1) (N2 8 2) (N3 8 3)	(N1 6 1) (N2 8 2) (N3 9 3)		Para estos pares, el tic de N2 es el umbral de selección.
N2->N1	N1	(N1 6 1) (N2 7 2) (N3 9 3)	(N1 6 1) (N2 8 2) (N3 9 3)	SELECCIONAR * DE N2.K DONDE (nodo = N2 y tic en [7, 8])	Lo mismo que antes pero invirtiendo los roles de N1 y N2. Los resúmenes son iguales al final de la operación.
	N2	(N 1 6 1) (N2 8 2) (N3 9 3)	(N1 6 1) (N2 8 2) (N3 9 3)		

1.6 Detección de conflictos con resúmenes

La detección de conflictos es ligeramente más compleja que en el algoritmo original (véase el apartado 1.1) ya que los objetos individuales no constituyen un reloj vectorial completo, sino que constituyen solamente un par de valores. Sin embargo, los conflictos todavía pueden detectarse de manera fiable ya que hay disponibles 4 fragmentos de información cuando se sincroniza un objeto: los pares de valores (Nj, Tj) del objeto en ambos lados, y además los relojes vectoriales de los conjuntos en ambos lados. La detección de conflictos funciona de la siguiente manera:

1. En primer lugar se comparan los pares de valores de los objetos en ambos lados. Si llevan el mismo id de nodo, no hay conflicto y el objeto que tiene el mayor tic es el más reciente.

2. Si los pares de valores de objeto tienen id de nodo diferentes, el par de valores de objeto en un lado se compara con el resumen en el otro lado. Si el tic asociado al objeto es estrictamente inferior al tic correspondiente en el resumen del otro lado, no hay conflicto y el lado del resumen tiene la última versión. En caso contrario, todavía no puede tomarse una decisión sobre conflicto.

3. Si el test anterior no ha tenido éxito, se realiza de nuevo el mismo test en la otra dirección (el otro objeto se compara con el otro resumen). De nuevo, si el tic asociado al objeto es estrictamente inferior al tic correspondiente en el resumen, no hay conflicto y el lado del resumen tiene la última versión.

4. Si fallan los tests anteriores, hay un conflicto.

Este algoritmo funciona en algunos ejemplos mostrados a continuación (sincronización de N1 a N2):

Caso	N1		N2		Resultado	Comentarios
	Objeto	Resumen	Objeto	Resumen		
a	(N1 5)	(N1 6 1) (N2 7 2) (N3 9 3)	(N1 4)	(N1 5 1) (N2 8 2) (N3 8 3)	Sin conflicto N1 es más reciente	Caso 1: los pares de valores tienen el mismo id de nodo
b	(N1 5)	(N1 6 1)	(N2 6)	(N1 5 1)	Sin conflicto	Caso 2 ó 3 (dependiendo

		(N2 7 2) (N3 9 3)		(N2 8 2) (N3 8 3)	N1 es más reciente	de cómo se empiece): par de valores en el lado de N2 es estrictamente inferior al par correspondiente en el resumen del lado de N1.
c	(N1 5)	(N1 6 1) (N2 7 2) (N3 9 3)	(N2 7)	(N1 5 1) (N2 8 2) (N3 8 3)	Conflicto N1 gana porque su prioridad de conflicto (1) es inferior a la de N2 (2)	Caso 4: las dos comparaciones cruzadas entre el par de valores y el resumen han fallado.
d	(N1 5)	(N1 6 1) (N2 7 2) (N3 9 3)	(N3 7)	(N1 5 1) (N2 8 2) (N3 8 3)	Sin conflicto N1 es más reciente	Caso 2 ó 3: el par de valores en el lado de N2 es estrictamente inferior al par de valores correspondiente en el resumen en el lado de N1.
e	(N38)	(N1 6 1) (N2 7 2) (N3 9 3)	(N2 7)	(N1 5 1) (N2 8 2) (N3 8 3)	Conflicto N1 gana porque su prioridad de conflicto (1) es inferior a la de N3 (3)	Caso 4: las dos comparaciones cruzadas entre el par de valores y el resumen han fallado.

A continuación se proporcionan escenarios que pueden dan lugar a los casos anteriores y que justifican los resultados proporcionados en la tabla:

a) El objeto se modificó en N1 en el tic 4 y después se sincronizó con respecto a N2. Después el objeto se modificó en N1 en el tic 5 pero se sabe que no se ha modificado en N2 porque todavía está etiquetado con (N1 4) en el lado de N2. Por lo tanto, no hay conflicto y la versión 5 de N1 necesita propagarse a N2.

b) El objeto se modificó en N2 en el tic 6. Esta modificación se ha propagado a N1 (esto se sabe porque el resumen de N1 contiene (N2 7 2)). Después, el objeto se ha modificado en N1 en el tic 5. Por lo tanto, no hay conflicto y la versión 5 de N1 necesita propagarse a N2.

c) El objeto se modificó en N1 en el tic 5. Esta modificación no ha propagado antes a N2 porque el resumen de N2 contiene (N1 5 1). Asimismo, el objeto se modificó en N2 en el tic 7 y esta modificación no se ha propagado a N1 porque el resumen de N1 contiene (N2 7 2). Por lo tanto, hay un conflicto puesto que las dos modificaciones se han llevado a cabo de manera independiente.

d) El objeto se modificó en N3 en la versión 7 y después se sincronizó con respecto a N2 y N1 (no se sabe ni en qué orden ni a través de qué trayectoria, pero sí se sabe que esta sincronización se ha producido gracias a las entradas de resumen de N3). El objeto se modificó entonces en N1 en el tic 5.

No hay conflicto y la versión de N1 es más reciente porque se sabe que esa versión (N3 7) se ha propagado antes a N1.

e) El objeto se modificó en N3 en la versión 8. Esta modificación se ha propagado a N1 pero no a N2 (esto se sabe a partir del tic de N3 en los dos resúmenes). El objeto también se modificó en N2 en la versión 7 y esta modificación no ha propagado a N1 (esto se sabe porque el resumen de N1 contiene (N2 7 2)). Por lo tanto, hay conflicto porque el objeto se ha modificado de manera independiente en dos nodos (N3 y N2).

2 Arquitectura

2.1 Preámbulo

La sección anterior describe un algoritmo de sincronización de manera abstracta. No pone muchas limitaciones en la arquitectura real en la que se implantará este algoritmo. Por ejemplo, no dice si las pasadas de sincronización se ejecutan desde uno de los nodos o de manera externa a los nodos. Si se ejecuta desde uno de los nodos, no dice si

todas las pasadas se ejecutan desde el mismo nodo o si las pasadas pueden ejecutarse desde diferentes nodos. Tampoco dice cómo se accede a los datos: conexión directa a las bases de datos (arquitectura C/S clásica), interfaz de servicio (SOAP o REST (transferencia de estado representacional)) o de otro modo.

El algoritmo puede implantarse de varias formas (proceso externo/interno, único/múltiples procesos, CS/SOA (arquitectura orientada a servicios), etc.). En esta segunda sección se proporciona un ejemplo de una posible implantación.

2.2 Ajustes en la terminología

A continuación se describe una implementación específica del esquema, que es la especificación de un protocolo REST en las solicitudes del solicitante.

Es necesario adaptar la terminología utilizada en la sección anterior a la terminología utilizada para este ejemplo de implementación. La siguiente tabla proporciona tal correspondencia:

Algoritmo	**Implementación de ejemplo**
objeto	recurso
conjunto	entidad
nodo	punto final
tic	tic
resumen	resumen

Hubiera sido posible utilizar la terminología del ejemplo de implementación de la sección anterior. No se ha utilizado para resaltar que el hecho de que el algoritmo es independiente de la arquitectura, de manera que puede analizarse y describirse independientemente de la arquitectura.

2.3 Punto final

La correspondencia entre nodo y punto final de la tabla anterior necesita una explicación adicional. Cuando se describió el algoritmo, la naturaleza de los nodos se explicó de manera superficial: un nodo identifica una ubicación y, por ejemplo, un nodo puede administrar varios conjuntos.

En el lado de la implementación se lleva a cabo un enfoque ligeramente restrictivo: un punto final representa un conjunto en una ubicación (una entidad en un proveedor de servicios dado) en lugar de la propia ubicación (el proveedor de servicios). Por lo tanto, un punto final es en realidad una URL de entidad de un proveedor de servicios dado. Esta interpretación más restrictiva de “nodo” no afecta al algoritmo: si el algoritmo funciona para un nodo que administra varios conjuntos, también funciona para el caso más sencillo en el que un nodo administra un único conjunto.

La siguiente tabla ofrece ejemplos de puntos finales. Tal y como se muestra en la tabla, las URL de puntos finales pueden incluir filtros. Esta característica debe utilizarse con precaución. Para garantizar la seguridad, los filtros siempre deben crear una partición de la entidad (por ejemplo, un punto final por país) para que los conjuntos de dos puntos finales del mismo proveedor no se solapen. Si los filtros se solapan, la detección de conflictos puede generar falsos positivos.

Punto final	Descripción
http:/sdata.acme.com/sdata/app1/test/accounts	Entidad *"account*s" en proveedor de servicios http://sdata.acme.com/sdata/app1/test
http://sdata.acme.com/sdata/app2/test/accounts	Misma entidad (*accounts*) en un proveedor diferente (app2)
http://sdata.acme.com/sdata/appl/test/products	Diferente entidad (*product*s) en el mismo proveedor (app1)
http://sdata.acme.com/sdata/app1/test/accounts ? where=country eq 'UK'	Pueden utilizarse filtros en las URL de puntos finales. Filtros diferentes se tratan como puntos finales diferentes

2.4 Motor

En la arquitectura de sincronización implementada, los puntos finales son las puertas de entrada a las aplicaciones, son las URL que un “motor” de sincronización utiliza para leer y escribir desde la aplicación.

Los procesos que intercambian datos entre aplicaciones para que se sincronicen se ejecutan por componentes denominados "motores de sincronización".

La arquitectura implementada es muy flexible en el lado del motor. Soporta escenarios centralizados en los que un motor central lleva a cabo todas las interacciones, así como escenarios descentralizados en los que varios motores actúan conjuntamente. Este punto se desarrollará en el apartado 2.9 posterior.

2.5 Mensajes

Un motor de sincronización interactúa con una aplicación a través de los puntos finales (URL) que la aplicación expone. El motor utiliza solamente tres tipos de mensajes para comunicarse con las aplicaciones, descritos en la siguiente tabla, donde Atom se refiere a un lenguaje XML utilizado para mecanismos de suministro web:

Tipo de mensaje	URL	Formato	Descripción
Resumen	PuntoFinal/$SincrResumen	Entrada Atom que contiene datos útiles del resumen	Este mensaje es una entrada Atom simple que contiene el resumen. El motor lo utiliza para leer resúmenes de los puntos finales
Mecanismo de suministro de Delta	PuntoFinal/$SincrOrigen PuntoFinal/$SincrDestino	Mecanismo de suministro Atom con lote de implementación de ejemplo y extensiones de sincronización	Éste es el mensaje de sincronización principal. Se utiliza para transferir los recursos que se han modificado de un punto final a otro. El mensaje es un mecanismo de suministro Atom que contiene lotes de implementación y extensiones de sincronización.
Diagnosis	URL de diagnosis	Mecanismo de suministro Atom con datos útiles de diagnosis de implementación de ejemplo	Este mensaje se utiliza para notificar errores y avisos a un componente que los registra para que el usuario pueda revisarlos y actuar en consecuencia. Esta implementación no impone una URL ya que escenarios de integración diferentes tienen requisitos diferentes sobre la notificación de errores. En cambio, la URL de diagnóstico debe configurarse antes de ejecutar una pasada de sincronización.

2.6 Proceso

De manera genérica, la figura 1 muestra cómo los componentes interactúan entre sí en una pasada de sincronización. El proceso requiere 4 etapas numeradas de 1 a 4. Todas las etapas se inician por el motor de sincronización y esta figura muestra el caso más general en que la aplicación origen, la aplicación destino, el motor y el registrador de eventos son componentes diferentes que pueden residir eventualmente en diferentes ordenadores.

El proceso se lleva a cabo de la siguiente manera:

En 1, el motor obtiene el resumen destino.

En 2, el motor envía el resumen destino al origen. El origen utiliza los dos resúmenes (el suyo propio y el resumen destino) para seleccionar los recursos que deben trasferirse al destino (utiliza el algoritmo descrito en el apartado 1.5). El origen devuelve el mecanismo de suministro de delta al motor.

En 3, el motor envía el mecanismo de suministro de delta al destino. El mecanismo de suministro de delta es un lote de instrucciones de creación/actualización/borrado que el proveedor destino ejecuta para sincronizar su dispositivo de almacenamiento de datos. El mecanismo de suministro de delta también contiene los dos resúmenes que se han utilizado para seleccionar recursos en el lado origen. El destino utiliza estos resúmenes para detectar y resolver conflictos (utiliza el algoritmo descrito en el apartado 1.6). Cuando el lote se ha procesado totalmente, el destino actualiza su resumen para reflejar el hecho de que se ha aplicado el delta. El destino devuelve información de diagnóstico acerca del éxito/fracaso de las operaciones que se han enviado.

En 4, el motor envía al administrador de eventos los diagnósticos que acaba de recibir. El administrador de eventos los registra y avisa a los usuarios si está configurado para ello.

El delta es un mecanismo de suministro que está dividido en páginas (utilizando el protocolo de paginación del ejemplo de implementación). Por lo tanto, el motor repite las etapas 2, 3 y 4 en cada página del mecanismo de suministro. En la etapa 3, el resumen destino solo se actualiza al final de la última página (esta operación podría

haberse realizado como una etapa diferente #5 para aislar claramente el bucle en las etapas 2, 3 y 4, pero entonces el motor realizaría un recorrido de ida y vuelta adicional al destino).

2.7 Observaciones

Esta arquitectura tiene algunas propiedades interesantes:

Todas las comunicaciones están basadas en HTTP y URL. Como resultado, el motor puede ejecutarse desde cualquier parte siempre y cuando pueda acceder a las URL (el motor es un consumidor de servicios que está conectado a varios proveedores).

La inteligencia está en los puntos finales y en los mensajes, no en el motor. El motor es solamente un proceso sencillo que intercambia mensajes. Necesita conocer las URL de los 3 participantes (el origen, el destino y el administrador de eventos) pero solo necesita conocer muy poco acerca de los mensajes que están intercambiándose (solo necesita detectar el final del lote para detener su bucle). Implementar un motor es una tarea bastante trivial (no puede decirse lo mismo de los puntos finales).

El motor no tiene estados. Puede encapsularse como un objeto sencillo que se instancia al principio de cada pasada. Las URL deben estar configuradas antes de iniciar la pasada de sincronización. Una vez que se haya completado la pasada, el objeto motor puede eliminarse. Por lo tanto, el motor no está ligado a ninguna ubicación particular y las pasadas de sincronización pueden ejecutarse desde diferentes lugares. Todo lo que se necesita es que el motor conozca las URL y esto es más un problema de configuración/repositorio que un problema de sincronización.

2.8 Implantaciones simplificadas

La implantación puede simplificarse integrando el motor en uno de los puntos finales. En este caso, las comunicaciones entre el motor y su punto final integrador pueden cortocircuitarse (no es necesario que se realicen mediante HTTP y URL). Por ejemplo, si el motor se ejecuta desde la aplicación origen y si el origen también maneja los diagnósticos, la figura 1 se convierte en la configuración mostrada en la figura 2, en la que los números se refieren a las mismas funciones mencionadas anteriormente con relación a la figura 1.

La figura 2 muestra una implantación simplificada del protocolo de sincronización. En esta configuración, el motor y el registrador de eventos están integrados en la aplicación origen. La arquitectura se reduce a dos componentes: una aplicación origen que lleva a cabo las pasadas de sincronización y una aplicación destino que responde a solicitudes de sincronización. La sincronización está limitada a una dirección: los cambios se propagan desde el origen al destino pero no desde el destino al origen.

En esta arquitectura simplificada, el origen ya no es un proveedor de servicios, sino que es una aplicación que consume servicios proporcionados por otra aplicación.

Si es necesaria una sincronización bidireccional entre las dos aplicaciones, todavía puede utilizarse la misma arquitectura simplificada. Para la sincronización en la otra dirección, la figura 3 muestra una arquitectura en la que, de nuevo, los números se refieren a las mismas funciones mencionadas anteriormente con relación a la figura 1.

La figura 3 es similar a la figura 2, pero con los roles del origen y el destino invertidos. Al igual que en la figura 2, la sincronización está limitada a una dirección, pero esta vez el motor de sincronización está en el lado destino y está trayendo los cambios desde un origen remoto, mientras que en la figura 2 estaba en el lado origen y llevaba los cambios a un destino remoto.

La figura 4 muestra la arquitectura simplificada y bidireccional obtenida combinando las figuras 2 y 3. Ésta es una configuración de “concentrador de sincronización” típica en la que hay un concentrador central (aplicación A) que ejecuta el motor, administra el registro de eventos y lleva a cabo la sincronización con un satélite (proveedor B). La figura sólo muestra un satélite pero puede haber varios. En esta configuración, el concentrador dirige las pasadas de sincronización y los satélites son más pasivos, ya que solo responden a solicitudes del concentrador.

La figura 5 muestra un escenario típico de un dispositivo portátil desconectado. Los roles están invertidos con respecto a la figura 4. El servidor (concentrador) es el proveedor B y el dispositivo portátil (satélite) es la aplicación A. El motor de sincronización se ejecuta en el dispositivo portátil cuando puede establecer una conexión con el servidor. El registro de eventos está centralizado en el servidor. Esta figura solo muestra un dispositivo portátil pero la arquitectura permite cualquier número de dispositivos portátiles.

2.9 Otros escenarios de implantación

El "ejercicio de simplificación” anterior muestra que la sincronización del ejemplo de implantación puede permitir tanto un escenario tipo "conjunto" en el que un motor central coordina las pasadas de sincronización entre servicios

como un escenario de "dispositivo portátil desconectado" en el que cada dispositivo portátil ejecuta su propio motor de sincronización.

Pero la arquitectura permite muchas más variantes ya que todas las interacciones se realizan a través de URL y la ubicación de los componentes no es relevante (todo lo que se necesita es que el motor pueda acceder a las URL de los otros componentes). La flexibilidad es particularmente importante en el lado de notificación de errores, y nada impide que el motor envíe los diagnósticos a más de un punto final de diagnóstico. En el contexto de un conjunto de aplicaciones, normalmente esto no es necesario ya que el conjunto tiene una utilidad central de administración de eventos, pero cuando las aplicaciones no pertenecen a un conjunto, el ejemplo de implementación permite que el sistema se configure para notificar errores en una de las aplicaciones, en la otra, en ambas, o algunas veces en una y algunas veces en la otra (dependiendo de la dirección, por ejemplo). Esta decisión debe tomarse teniendo en cuenta consideraciones del producto y la arquitectura del ejemplo de implementación debe poder satisfacer las diferentes necesidades de los escenarios de integración del producto.

2.10 Sincronización inmediata y de equiparación

El motor descrito anteriormente funciona en modo de “equiparación". La sincronización se realiza en pasadas que pueden ejecutarse por un planificador o lanzarse manualmente. Las pasadas de sincronización utilizan la siguiente lógica:

- Leer el resumen destino.

- Consultar el delta del origen (el resumen destino se pasa como una entrada de datos ya que el origen lo necesita para calcular el delta).

- Aplicar el delta al destino (que resolverá conflictos y actualizará su resumen al final).

- Éxito/fracaso de registro.

En un cierto número de escenarios, es importante propagar los cambios tan pronto como sea posible en lugar de tener que esperar una pasada de sincronización. Este modo "inmediato" implica un proceso ligeramente diferente:

- Detectar que un recurso ha cambiado en el lado origen.

- Preparar un delta que representa el cambio.

- Aplicar el delta al destino.

- Éxito/fracaso de registro.

El protocolo de sincronización del ejemplo de implementación está diseñado para manejar los dos modos bajo un protocolo común. Evidentemente, la implementación del origen será diferente en los dos modos (en el modo de equiparación, el origen necesita proporcionar un punto de entrada para consultar el delta; en el modo inmediato, el origen necesita poder detectar cambios), pero el protocolo está diseñado para que el lado destino pueda manejar ambos modos de manera uniforme (el destino no necesita saber qué modo está utilizándose).

Además, el protocolo de sincronización del ejemplo de implementación está diseñado para soportar una mezcla de los dos modos. Esta característica es importante ya que el modo inmediato supone un desafío. Por ejemplo, una de las aplicaciones puede estar fuera de línea o la aplicación destino puede estar inactiva cuando el origen trata de propagar los cambios. Esto puede solucionarse introduciendo una cola para garantizar la entrega pero esto tiene algunas desventajas (la cola necesita permanecer activa si la aplicación origen está cerrada, la cola puede acabar desperdiciando muchos recursos, especialmente si un destino nunca está disponible, etc.). Una alternativa es considerar que el modo inmediato es solamente un “esfuerzo razonable” para propagar cambios y utilizar el modo de “equiparación" para garantizar que los sistemas se sincronicen periódicamente incluso si se han perdido algunas solicitudes de sincronización inmediata.

En el modo inmediato, la arquitectura es normalmente como la mostrada en la figura 6, que muestra una configuración típica para una sincronización inmediata. La aplicación origen lleva sus cambios a una aplicación destino. La principal diferencia con el modo de equiparación es la eliminación de un recorrido de ida y vuelta para consultar el resumen destino. Esta configuración es más eficaz para propagar cambios que la configuración de equiparación ilustrada en la figura 2, pero un sistema real probablemente combinará ambas configuraciones, de modo que puede volver al modo de equiparación cuando falle el modo inmediato.

La principal diferencia con el modo de equiparación es que el motor ya no lee el resumen destino (esto implicaría un recorrido de ida y vuelta adicional), si solo lleva el delta al destino.

El protocolo del ejemplo de implementación maneja el modo inmediato y la mezcla de modos utilizando un mensaje de delta "rico" (que se describe a sí mismo). El mensaje de delta del ejemplo de implementación contiene:

- Un lote de acciones para crear/actualizar/eliminar recursos.

- Los metadatos de sincronización para los recursos individuales en el lote (punto final + tic + triplete de marca de tiempo de modificación).

- Dos resúmenes que se han utilizado para generar el lote.

El proceso incluye dos resúmenes que delimitan el delta (cambio) en el mecanismo de suministro. Estos dos resúmenes se denominan resumen de "valores iniciales" y resumen de "valores finales". Se establecen de la siguiente forma:

- En el caso de “equiparación”, el resumen de valores iniciales es simplemente el resumen destino y el resumen de valores finales es el resumen origen. Esta combinación corresponde a los criterios utilizados para seleccionar recursos en el lado origen (véase el apartado 1.5) (dicho de otro modo, el lado origen tiene esta versión del objeto, pero el lado destino aún no lo ha visto).

- En el caso “inmediato”, el resumen de valores iniciales es el resumen origen antes de que el recurso se modificara, y el resumen de valores finales es el resumen origen después de que el recurso se modificara (igual que los valores iniciales pero el tic para el punto final origen está incrementado en 1).

Con este refinamiento, el destino sabe de manera precisa qué contiene el delta. El destino comprueba si los intervalos de tic definidos por los valores iniciales y los valores finales son contiguos a sus propios valores de tic (el resumen destino). Si los intervalos son contiguos, el destino puede actualizar su resumen. En caso contrario, si hay un espacio entre el intervalo de tics definido por el resumen de valores iniciales y el resumen destino, el destino sabe que al menos se ha pasado por alto una sincronización inmediata y no actualiza su resumen. La siguiente sincronización de equiparación cerrará el espacio.

La razón de tener dos resúmenes en el mensaje es hacer que los mecanismos de suministro se "describan más a sí mismos”. No contienen solamente los cambios; también contienen información sobre los dos resúmenes que incluyen los cambios. Esto es útil en el modo inmediato porque permite al destino comprobar si el intervalo de cambios enviado por el origen es contiguo a los cambios que ya se han incorporado en el destino. Si el intervalo de cambios es contiguo, el destino puede incorporarlos y actualizar su resumen. Si no lo son (esto sucede si no se ha entregado al destino un mensaje anterior de sincronización inmediata, el destino debe descartar la solicitud.

El protocolo de implementación está diseñado para que el lado destino pueda manejar el modo de equiparación y el modo inmediato con la misma lógica. Por eso se utiliza el mismo mensaje (mecanismo de suministro) para ambos modos, con dos resúmenes en el mensaje. Esto hace que el protocolo sea más robusto (por ejemplo, en el improbable escenario en que el nodo destino se restaura a partir de una vieja copia de seguridad entre el momento en que el origen consultó su resumen y el momento en que el origen envía sus cambios).

La siguiente tabla muestra cómo funciona este algoritmo.

Valores iniciales	Valor final	Destino	Nuevo destino	Comentarios
(N1 6 1) (N2 7 2) (N3 9 3)	(N1 7 1) (N2 7 2) (N3 9 3)	(N1 6 1) (N2 9 2) (N3 8 3)	(N1 7 1) (N2 9 2) (N3 8 3)	N1 propaga un cambio a N2 en el modo inmediato. Por tanto, los valores iniciales y los valores finales solo difieren en N1 (véase anteriormente). El tic de valores iniciales para N1 (6) coincide con el tic destino para N1 (6). Por tanto, el destino sabe que ha recibido antes todas las versiones de N1 hasta el tic 6 y que ahora está recibiendo cambios entre 6 y 7 de N1, por lo que ahora tiene todas las versiones de N1 hasta el tic 7. El destino puede actualizar el tic de N1 a 7 en su resumen.
(N1 7 1) (N2 7 2) (N3 9 3)	(N1 8 1) (N2 7 2) (N3 9 3)	(N1 7 1) (N2 9 2) (N3 8 3)		N1 propaga un segundo cambio a N2 en el modo inmediato, en el momento en que N2 está fuera de línea. Este mensaje no llega al destino. Por tanto, no sucede nada y el resumen destino no se actualiza.
(N1 8 1) (N2 7 2)	(N1 9 1) (N2 7 2)	(N1 7 1) (N2 9 2)	(N1 7 1) (N2 9 2)	N1 propaga un tercer cambio a N2 en el modo inmediato (se supone que N2 está de nuevo en línea). El tic de

(N3 9 3)	(N3 9 3)	(N3 8 3)	(N3 8 3)	valores iniciales para N1 (8) no coincide con el tic destino para N1 (7). El destino no puede actualizar su tic de resumen para N1 porque no recibió los cambios entre los tics 7 y 8.
(N1 7 1) (N2 9 2) (N3 8 3)	(N1 10 1) (N2 7 2) (N3 9 3)	(N1 7 1) (N2 9 2) (N3 8 3)	(N1 10 1) (N2 9 2) (N3 9 3)	Una pasada de equiparación se ejecuta de N1 a N2. En el modo de equiparación, el resumen de valores iniciales es el resumen destino y el resumen de valores finales es el resumen origen (véase anteriormente). Por tanto, esta pasada selecciona los recursos que tiene tics de N1 entre 7 y 10 o tics de N3 entre 8 y 9. Los intervalos de tics que el destino recibe son adyacentes a los intervalos de tics que tenía antes (esto siempre se cumple en el modo de equiparación porque los resúmenes de valores iniciales y destino son idénticos). Por tanto, el destino puede fijar su tic de N1 a 10 y su tic de N3 a 9 al final de esta pasada de sincronización. La anomalía queda resuelta. Nota: el tic de N2 es 7 en el origen y 9 en el destino al final de esta pasada. Esto no es una anomalía. Otra pasada de sincronización en la otra dirección se necesaria para hacer que el tic de N2 esté al mismo nivel.

Sumario

El esquema descrito anteriormente permite una reducción ventajosa en los datos que se añaden a cada registro u objeto, ya que solamente se añade un último par nodo/tic, y el resto del reloj vectorial se envía con el resumen. Este esquema permite la misma eficacia y detección de conflictos que los esquemas actuales, pero con una sobrecarga mucho más reducida.

Todas las características dadas a conocer en esta memoria descriptiva (incluyendo cualquier reivindicación, resumen y dibujos adjuntos) y/o todas las etapas de cualquier método o proceso dado a conocer pueden combinarse en cualquier combinación, excepto en combinaciones en las que al menos parte de tales características y/o etapas sean mutuamente excluyentes.

Cada característica dada a conocer en esta memoria descriptiva (incluyendo cualquier reivindicación, resumen y dibujos adjuntos) puede sustituirse por características alternativas que ofrezcan la misma finalidad, u otra equivalente o similar, a no ser que se indique expresamente lo contrario. Por tanto, a no ser que se indique expresamente lo contrario, cada característica dada a conocer es solamente un ejemplo de una serie genérica de características equivalentes o similares.

La invención no está limitada a los detalles de la(s) anterior(es) realización(es). La invención se extiende a cualquier característica novedosa o a cualquier combinación novedosa de las características dadas a conocer en esta memoria descriptiva (incluyendo cualquier reivindicación, resumen y dibujo adjuntos), o a cualquier etapa novedosa o a cualquier combinación novedosa de las etapas de cualquier método o proceso dado a conocer que esté dentro del alcance de las reivindicaciones adjuntas.

REIVINDICACIONES

1.- Un método para sincronizar al menos una primera y una segunda instancia de un conjunto de información que comprende una pluralidad de entradas, en el que dichas primera y segunda instancias están ubicadas en diferentes nodos de un sistema, comprendiendo el método:

asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

caracterizado por generar un resumen para cada una de las al menos primera y segunda instancias del conjunto de información, donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las al menos primera y segunda instancias con otra de las al menos primera y segunda instancias del conjunto de información; y

sincronizar las al menos primera y segunda instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias;

en el que un par existente formado por un valor de nodo y un valor actual de reloj lógico de una entrada se elimina y se sustituye con un nuevo par formado por un valor de nodo y un valor actual de reloj lógico cuando se modifica dicha entrada.

2.- El método según la reivindicación 1, en el que un valor de marca de tiempo se asocia a cada entrada de datos cuando se modifica dicha entrada, además del par formado por un valor de nodo y un valor actual de reloj lógico.

3.- El método según cualquier reivindicación anterior, en el que la sincronización se lleva a cabo de manera bidireccional.

4.- El método según cualquier reivindicación anterior, en el que cada resumen está ubicado en un nodo con el que está relacionado.

5.- El método según cualquier reivindicación anterior, en el que el resumen comprende un par formado por un valor de nodo y un valor de reloj lógico para cada nodo del sistema.

6.- El método según la reivindicación 5, en el que el resumen incluye además un valor de prioridad de conflicto para cada nodo del sistema.

7.- El método según cualquier reivindicación anterior, en el que se proporcionan dos resúmenes para cada evento de sincronización.

8.- Un dispositivo de sincronización que puede hacerse funcionar para llevar a cabo el método según una cualquiera de las reivindicaciones 1 a 8.

9.- Un sistema informático que comprende una pluralidad de nodos, donde cada nodo incorpora una instancia de un conjunto de información, haciéndose funcionar el sistema informático para sincronizar las instancias del conjunto de información, y que comprende al menos un elemento lógico de sincronización que puede hacerse funcionar para:

asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

caracterizado por generar un resumen para cada una de las instancias del conjunto de información, donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las instancias con otra de las al menos instancias del conjunto de información; y

sincronizar las instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias;

en el que un par existente formado por un valor de nodo y un valor actual de reloj lógico de una entrada se elimina y se sustituye con un nuevo par formado por un valor de nodo y un valor actual de reloj lógico cuando se modifica dicha entrada.

10.- El sistema informático según la reivindicación 9, en el que el elemento lógico de sincronización comprende un punto final de sincronización para cada nodo o para cada instancia del conjunto de información.

11.- El sistema informático según la reivindicación 9 o la reivindicación 10, en el que el elemento lógico de sincronización comprende uno o más motores de sincronización que pueden hacerse funcionar para llevar a cabo intercambios de mensajes entre puntos finales de sincronización.

12.- El sistema informático según la reivindicación 11, en el que los motores de sincronización y los puntos finales de sincronización residen en diferentes ordenadores.

13.- El sistema informático según una cualquiera de las reivindicaciones 10 a 12, en el que cada punto final de sincronización puede hacerse funcionar para llevar a cabo la asociación y la generación de resúmenes para su nodo dado.

14.- El sistema informático según una cualquiera de las reivindicaciones 10 a 13, en el que la pluralidad de motores de sincronización puede hacerse funcionar para la comunicación con los puntos finales para llevar a cabo la función de sincronización.

15.- Un producto de programa informático que comprende instrucciones que, cuando se ejecutan en un ordenador, se hacen funcionar para implementar un método de sincronización de al menos una primera y una segunda instancia de un conjunto de información que comprende una pluralidad de entradas, donde dichas primera y segunda instancias están ubicadas en diferentes nodos de un sistema informático, comprendiendo el método:

asociar un par formado por un valor de nodo y un valor actual de reloj lógico a cada entrada del conjunto de información de una instancia dada cuando se modifica dicha entrada de la instancia dada;

caracterizado por generar un resumen para cada una de las al menos primera y segunda instancias del conjunto de información, donde dichos resúmenes comprenden información de reloj lógico de un último evento de sincronización de una de las al menos primera y segunda instancias con otra de las al menos primera y segunda instancias del conjunto de información; y

sincronizar las al menos primera y segunda instancias del conjunto de información comparando los resúmenes y reconciliando las al menos primera y segunda instancias;

en el que un par existente formado por un valor de nodo y un valor actual de reloj lógico de una entrada se elimina y se sustituye con un nuevo par formado por un valor de nodo y un valor actual de reloj lógico cuando se modifica dicha entrada.

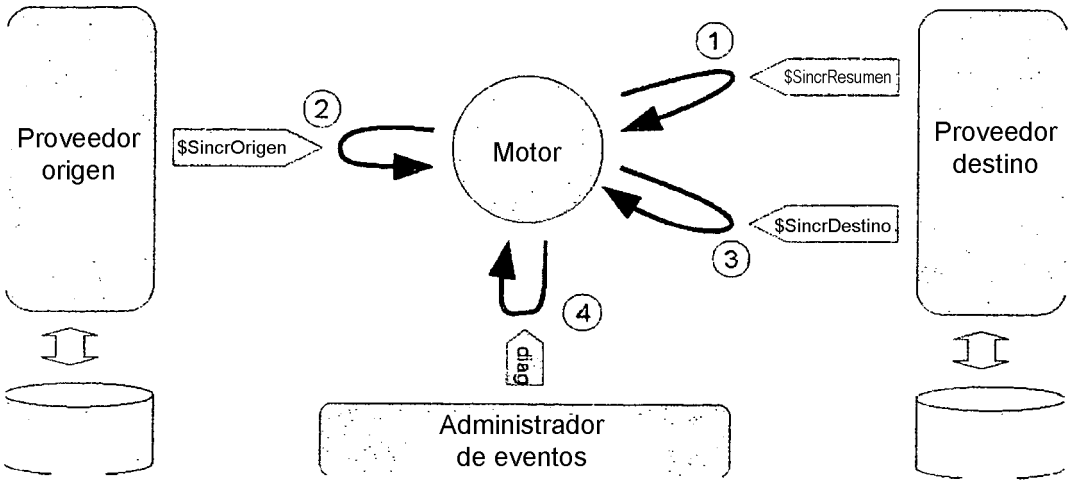


Figura 1

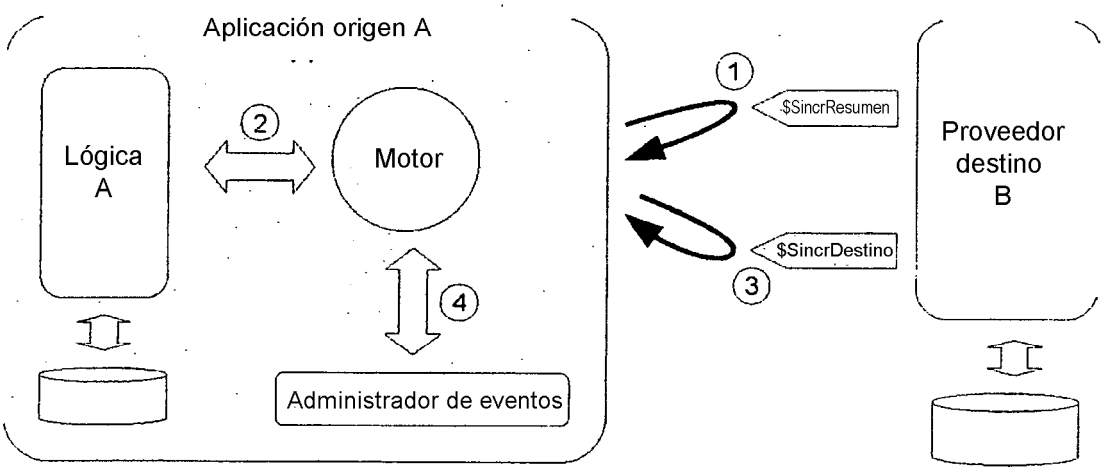


Figura 2

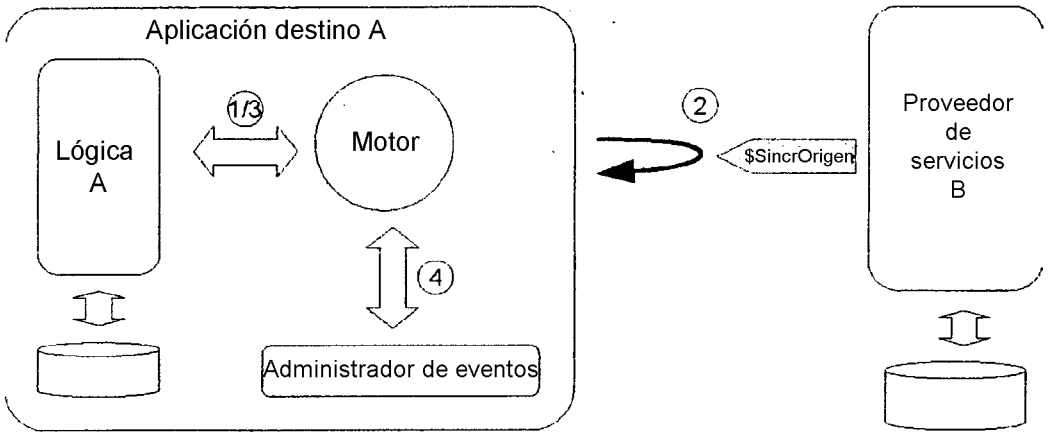
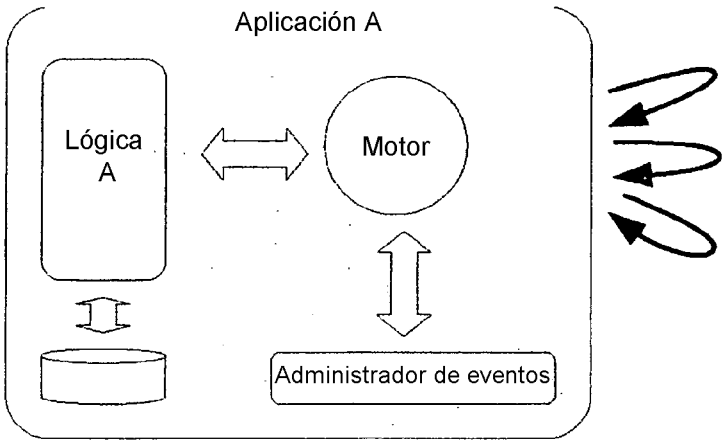


Figura 3



También es posible la siguiente variante:

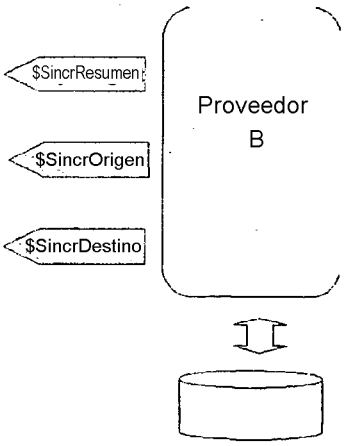


Figura 4

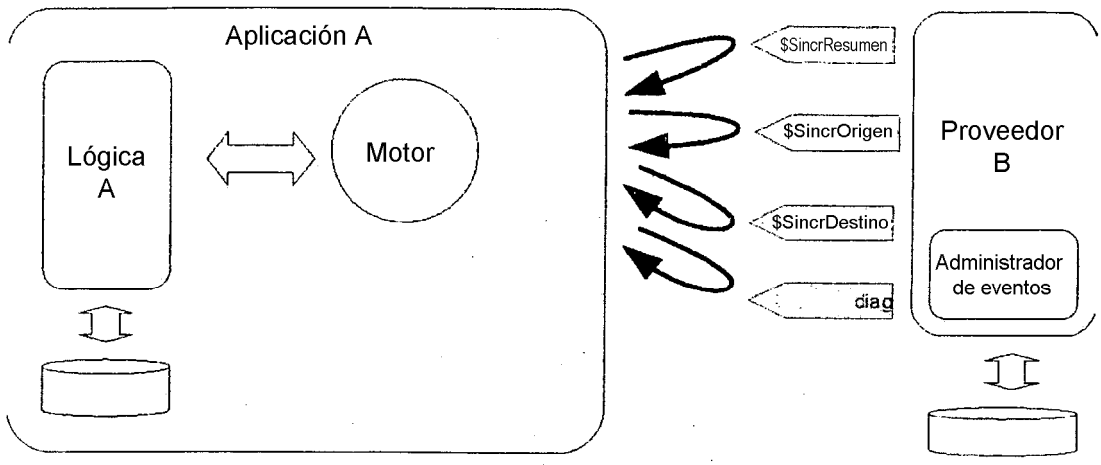


Figura 5

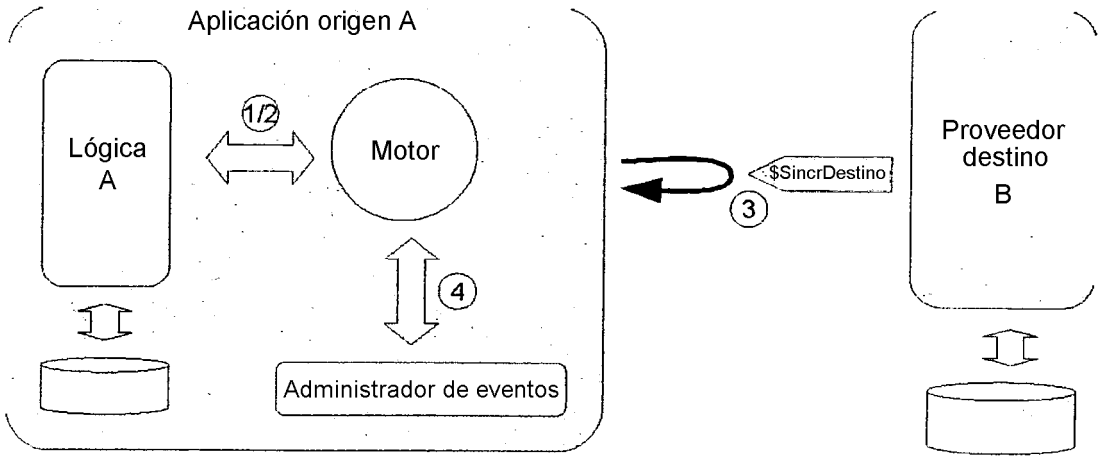


Figura 6

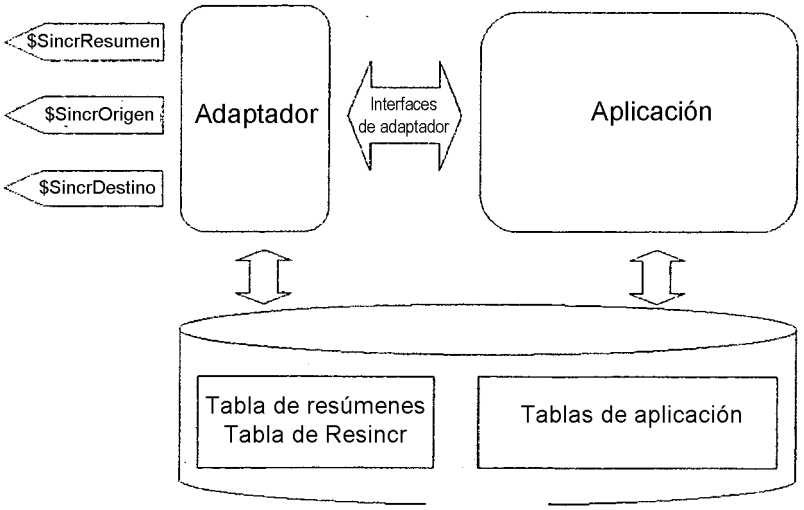

$SincrResumen
$SincrOrigen
$SincrDestino
Adaptador
Interfaces de adaptador
Aplicación
Tabla de resúmenes
Tabla de Resincr
Tablas de aplicación

Figura 7