(19) OFICINA ESPAÑOLA DE PATENTES Y MARCAS

ESPAÑA

(11) Número de publicación: **2 375 026**

(21) Número de solicitud: 200803715

(51) Int. Cl.:
G06F 9/54 (2006.01)
H04Q 9/00 (2006.01)

(12) SOLICITUD DE PATENTE A1

(22) Fecha de presentación: **26.12.2008**

(43) Fecha de publicación de la solicitud: **24.02.2012**

(43) Fecha de publicación del folleto de la solicitud: **24.02.2012**

(71) Solicitante/s: **Universidad de Castilla-La Mancha**
Campus Universitario-Pabellón de Gobierno
Plaza de la Universidad, 2
02071 Albacete, ES

(72) Inventor/es: **Villa Alises, David;**
Moya Fernández, Francisco;
Villanueva Molina, Félix Jesús;
Barba Romero, Jesús;
Rincón Calle, Fernando y
López López, Juan Carlos

(74) Agente: **Carpintero López, Mario**

(54) Título: **Módulo y métodos para la interconexión de equipos heterogéneos con recursos limitados por medio de middlewares de comunicaciones orientados a objeto.**

(57) Resumen:
Módulo y métodos para la interconexión de equipos heterogéneos con recursos limitados por medio de middlewares de comunicaciones orientados a objeto.
Módulo y métodos para la integración de objetos distribuidos en sistemas empotrados que pueden ser implementados mediante dos alternativas complementarias: software empotrado y síntesis de hardware. El método permite fabricar dispositivos para sensorización y actuación remota más baratos y pequeños, manteniendo además un muy alto grado de interoperabilidad con el middleware de comunicaciones, lo que facilita sensiblemente sus posibilidades de operación y su integración con otros sistemas preexistentes.

ES 2 375 026 A1

DESCRIPCIÓN

Módulo y métodos para la interconexión de equipos heterogéneos con recursos limitados por medio de middlewares de comunicaciones orientados a objeto.

Antecedentes de la invención

Campo técnico

La presente invención tiene relación con el sector técnico del control automático de procesos industriales, telecontrol y telemedida y miniaturización de dispositivos detectores y de accionamiento. También tiene una importante relación con los sistemas de Domótica, Inmótica, *Pervasive computing* (Informática penetrante) y *Ambient intelligence* (Inteligencia ambiental).

Algunos ejemplos de posibles aplicaciones de la presente invención son:

- Redes de dispositivos de bajo coste para la automatización de tareas en el hogar, la oficina, explotaciones agropecuarias, instalaciones industriales, etc.
- Integración de dispositivos para la provisión de servicios de valor añadido independientes de tecnología (vídeo-vigilancia, prevención de accidentes, monitorización, tele-asistencia, etc.).
- Implementación de dispositivos hardware que interactúan con software estándar sin necesidad de controladores o firmware específicos.

Descripción de la técnica relacionada

Tradicionalmente, la palabra *middleware* (software intermedio) se define como la capa de software situada entre el sistema operativo y las aplicaciones; su finalidad es proporcionar los mecanismos para la comunicación entre elementos de un sistema distribuido.

En general, cualquier *middleware* define interfaces de programación, protocolos de comunicación, y un modelo de información para objetos y servicios. Todo ello para permitir la construcción de aplicaciones distribuidas heterogéneas. Algunos ejemplos de este tipo de plataformas son CORBA (véase referencia [1]), ZeroC Ice (referencia [2]), Web Services (referencia [3]), Jini (referencia [4]), etc.

Muchos de los *middlewares* para sistemas distribuidos heterogéneos permiten también que aplicaciones escritas en diversos lenguajes y sobre diferentes plataformas puedan interactuar, a base de definir de forma precisa tanto los protocolos de comunicación como las interfaces de programación para cada lenguaje.

Las ventajas de las técnicas de orientación a objetos son bien conocidas en arquitecturas de software, y los *middlewares* distribuidos orientados a objeto tienen una larga historia de aplicaciones exitosas en muchas áreas de negocio de las tecnologías de la información. Además, muchas de las plataformas distribuidas orientadas a objetos están apoyadas por la industria y sus herramientas, protocolos y servicios han crecido en consenso con ella. El gran número de aplicaciones desarrolladas otorgan a los servicios, protocolos y aplicaciones ciertas garantías de fiabilidad, fruto de la experiencia y es uso continuado en aplicaciones sensibles.

Muchas iniciativas previas se han marcado como objetivo la miniaturización de los *middlewares* existentes. Por ejemplo, el propio *Object Management Group* publicó la especificación MinimumCORBA (referencia [5]), una versión ligera de su popular arquitectura CORBA (referencia [1]). MinimumCORBA elimina las prestaciones más costosas del núcleo de comunicaciones manteniendo un buen grado de interoperabilidad con objetos CORBA estándar.

Tal como apunta la referencia [6], existen principalmente tres tipos de propuestas para miniaturización de objetos distribuidos:

1. *Eliminar las prestaciones más costosas*, pero manteniendo la generalidad. Esta primera estrategia es la utilizada por dynamicTAO (referencia [7]) y sus descendientes: LegORB (referencia [8]) y UIC-CORBA (referencia [9]). LegORB es un ORB modularizado que tiene la capacidad de poderse configurar dinámicamente. La librería monolítica de TAO (referencia [10]) se descompone en un conjunto de bloques funcionales independientes que se pueden omitir en la aplicación objetivo. Según sus autores un cliente CORBA ronda los 20 KB. en un HP Jornada 600 que utiliza Windows CE como S.O., y en un cliente CORBA sobre PalmPilot con PalmOS 3.0. ronda los 6 KB.

   UIC (*Universally Interoperable Core*) define un esqueleto de *middleware* basado en componentes. Cada componente encapsula un conjunto reducido de características que se puede cargar dinámicamente dependiendo de la plataforma, dispositivo y red concreta. UIC, como su nombre indica, se puede usar para implementar núcleos de comunicaciones para diferentes *middlewares* además de CORBA, como Java RMI o DCOM. Un servidor CORBA estático ocupa aproximadamente 35 KB en un SH3 con Windows CE.

Un producto comercial similar es e*ORB (referencia [11]), un núcleo de comunicaciones con características de tiempo real capaz de ejecutarse en una HP iPAQ o en un DSP Texas Instruments TMS320C64X DSP.

Otro desarrollo representativo de esta primera estrategia es MicroQoSCORBA (referencia [12]), un núcleo de comunicaciones ajustable a medida que se puede generar a partir de una piezas predefinidas para implementar servidores y clientes para aplicaciones y dispositivos específicos (ha sido probado con SaJe (referencia [14]) y TINI (referencia [15]).

nORB (referencia [13) implementa un conjunto de protocolos de transporte como *plug-ins*, que incluyen protocolos específicos del entorno (ESIOP en tecnología CORBA). Comparte muchas ideas de from MicroQoSCORBA, como el hecho de utilizar una versión simplificada del protocolo GIOP estándar, llamada GIOPLite.

2. *Adaptar el middleware a dispositivos específicos*. Una muestra representativa de esta segunda estrategia para el desarrollo de *middlewares* ligeros es TINIORB (referencia [16]), un núcleo de comunicación basado en MinimumCORBA que ha sido adaptado especialmente para el dispositivo TINI de Dallas Semiconductor. PalmORB (referencia [17]) también es otro ejemplo de esta estrategia.

3. *Utilizar una pasarela*. Es decir, construir dispositivos con un protocolo propio muy simple, pero requiere que un dispositivo intermedio materialice la interoperabilidad con objetos del *middleware* estándar. Este es el enfoque usado en UORB (referencia [6]) y una de las alternativas de integración que propone SENDA (referencia [18]).

   Otra propuesta interesante del mismo tipo es (referencia [19]). Este trabajo muestra cómo un conjunto de pequeños microcontroladores de 8 bits pueden aparecer como un conjunto de objetos CORBA. El *host* ejecuta un proxy para cada dispositivo conectado y las comunicaciones entre los dispositivos y el proxy utilizan un protocolo específico no estándar.

La presente invención está fundamentada en algunos artículos previos del grupo ARCO. En particular, los artículos [A] y [B] sientan las bases del concepto que sustenta la presente invención, Los artículos [C] y [D] describen un protocolo de descubrimiento de servicios que puede aplicarse a los módulos de la presente invención.

En cualquier caso, el presente texto introduce mejoras y aportaciones no triviales en el módulo, procedimientos de operación del mismo y en el método para su construcción y despliegue. Dichas aportaciones constituyen el núcleo de las reivindicaciones que se solicitan.

Descripción de la invención

Problemas técnicos planteados

Todos los trabajos previos siguen las mismas reglas básicas: Eliminar los métodos de invocación e instanciación dinámica, simplificar el lenguaje de definición de interfaces (OMG IDL en el caso de CORBA) , eliminar los tipos de datos complejos o grandes, eliminar algunos campos del protocolo de comunicaciones, eliminar o simplificar los tipos de mensajes usados en el protocolo, eliminar el soporte a referencias indirectas y a servicios comunes, modularizar el núcleo de comunicaciones e instanciar sólo aquellos componentes que realmente se necesitan en cada caso.

Es importante señalar que todos los núcleos de comunicaciones que se citan arriba requieren muchas utilidades de soporte adicionales: serialización de tipos de datos, primitivas de comunicaciones, sistema operativo, etc. De modo que los requerimientos de memoria reales son órdenes de magnitud mayores que los tamaños citados.

Incluso la implementación de objeto distribuido más pequeña es mucho mayor de lo que resultaría factible para el entorno objetivo del presente documento. Requerir de un dispositivo TINI (que tiene un coste actual de unos 30 euros) para cada componente del sistema ubicuo implicaría precios astronómicos para un sistema mínimamente útil. Pensar en una máquina virtual Java con soporte RMI para cada bombilla o interruptor de un edificio es algo completamente inadmisible en muchos aspectos, no sólo el económico.

El primer problema que se plantea a la hora de intentar empotrar un *middleware* tal como Ice o CORBA en un nodo típico para una red de sensores y actuadores (SAN, Sensor/Actuator Networks) es la cantidad de recursos de cómputo (sobre todo memoria) que se necesita para los núcleos de estos *middlewares*.

En la figura 3 se muestra el diagrama simplificado de la invocación de un método remoto (RMI) en un *middleware* orientado a objetos estándar como CORBA.

En un bus de objetos convencional (ORB), una invocación remota implica uno o más Adaptadores de Objetos, un Mapa de Objetos que mantiene la correspondencia de pares objeto-sirviente y un conjunto de sirvientes, los cuales implementan el comportamiento de los objetos.

Hay dos elementos importantes en el mensaje de petición:

- *El identificador del objeto*, que determina a qué objeto va dirigido el mensaje y en última instancia qué sirviente la atenderá.
- *La operación*, que determina qué método del sirviente se debe ejecutar.

La comunicación estándar se consigue por medio de un protocolo como GIOP (*General Inter-ORB Protocol*) en el caso de CORBA o IceP (*Ice Protocol*) en el caso de ZeroC Ice.

Para garantizar interoperabilidad básica entre los componentes del *middleware* debe respetarse cierto grado de compatibilidad, independientemente si dichos objetos se realizan en software convencional, empotrado o implementaciones hardware. Esto se consigue respetando las normas que impone el *middleware* concreto y el protocolo de comunicaciones asociado, como puede ser GIOP o IceP.

CORBA es una arquitectura bastante veterana y de la que se ha escrito mucho sobre sus posibilidades de implementación en sistemas empotrados. Gran parte del trabajo realizado hasta la fecha está muy influenciado por MinimumCORBA. Esta especificación busca acotar la complejidad del ORB eliminando capacidades especialmente complejas. MinimumCORBA sigue siendo conforme a CORBA pero con capacidades reducidas. La ventaja principal es que los objetos MinimumCORBA son completamente portables, pueden ejecutase sin problemas sobre un ORB completo.

Objeto de la invención

Dado que el éxito de una tecnología está determinado en parte por las aplicaciones a las que pueda dar soporte, en el presente documento se considera que cuanto más sencillo sea el desarrollo de nuevas aplicaciones, más aplicaciones y con más variedad y rapidez estarán disponibles en el mercado. En este sentido, la solución de la presente invención permite desarrollar aplicaciones con nuevas tecnologías sin tener que aprender nuevas metodologías, sistemas operativos, interfaces o lenguajes de programación.

Muchos trabajos anteriores se han encaminado hacia la reducción de estos *middlewares* estándar, de manera que puedan ser integrados en dispositivos de menor coste que un ordenador personal. La presente solución también explora esta tendencia pero utiliza una perspectiva mucho más radical. En lugar de reducir un *middleware* estándar, en esta documento se analiza la generación automática de la mínima cantidad de software necesaria para mantener la interoperabilidad, sin incluir nada del *middleware* con el que se pretende ser compatible.

El objeto de la invención es un módulo y un método para la integración de objetos distribuidos en sistemas empotrados. La presente invención involucra a uno o más módulos que permiten que un dispositivo de sensorización o actuación cualquiera pueda ser accedido remotamente mediante algún *middleware* de comunicaciones orientado a objetos. Dichos módulos podrán ser manipulados de forma individual o conjunta dependiendo de las características del *middleware* y del sistema de comunicaciones disponible.

Este tipo de módulos puede ser utilizado en una gran variedad de aplicaciones tanto domésticas como industriales, ofreciendo la posibilidad de acceso con la granularidad necesaria a grandes despliegues de elementos sensores y actuadores de toda Índole y por una gran variedad de medios de comunicación de datos, incluyendo redes inalámbricas.

Como principales ventajas respecto de invenciones previas cabe destacar la posibilidad de utilizar dispositivos de cómputo mucho más pequeños, simples y baratos, debido a que los métodos de integración redundan en una sensible reducción en los requerimientos de memoria y capacidad de cómputo. Otras importantes ventajas son la autonomía de los módulos y la utilización de *middlewares* de comunicaciones estándar, con el consiguiente ahorro en herramientas de desarrollo y formación de personal.

Breve descripción de los dibujos

La figura 1 muestra el módulo constituido por tres componentes cuando se utiliza como medio de acceso a un sensor/actuador.

La figura 2 muestra el módulo como pasarela entre redes con tecnologías diferentes.

La figura 3 se ilustra la implementación de un *middleware* convencional en el marco de un servidor, donde una involucra a un adaptador de objetos, un mapa de objetos y un sirviente.

La figura 4 ilustra el mismo escenario que en la figura 3, cuando se utiliza uno de los módulos objeto de la invención.

La figura 5 ilustra un escenario básico de uso de un objeto activo donde un sensor notifica a un segundo objeto "observador" cualquier cambio que ocurra en la magnitud que mide.

La figura 6 muestra otro escenario de uso para objetos activos donde, en este caso, el objeto "observador" es un canal de eventos.

La figura 7 representa la secuencia de operaciones que se produce durante el establecimiento del mecanismo de consulta de un grupo arbitrario de objetos sensores/actuadores por medio del uso de canales de eventos y cómo la aplicación recibe los resultados.

La figura 8 se muestra el uso de un módulo con la configuración de la figura 2 para lograr la integración de un sensor remoto, que utiliza una tecnología distinta.

La figura 9 muestra los roles del programador de servicios (lado derecho) y del programador de módulos (lado izquierdo) y su relación con las especificaciones de interfaces y código necesario.

La figura 10 representa el flujo de trabajo mediante el cual se instala la aplicación en el módulo.

La figura 11. muestra el esquema de un compilador concreto como ejemplo de implementación del proceso descrito en la figura 10 para el *middleware* ZeroC Ice.

La figura 12 muestra el flujo de trabajo del proceso de generación de HwO (objetos hardware), de acuerdo con el segundo modo de realización de la invención.

La figura 13 es un ejemplo de *objeto hardware* para un conjunto (*buffer*) de enteros, mostrando las señales de sincronización necesarias para su integración en el diseño.

La figura 14 muestra la plataforma HwO proporcionando accesibilidad a los HwO desde y hacia el exterior, y

La figura 15 muestra la configuración habitual de un proxy, incluyendo la adaptación de los datos procedentes del bus por medio de las *Port Acquisition Units* (PAU) y el *Port Delivery Unit* (PDCJ).

Descripción detallada de la invención

La finalidad principal del módulo objeto de la invención es proporcionar acceso transparente a cualquier sensor/actuador por medio de un *middleware* de comunicaciones de uso general como puede ser CORBA, Jini, Web Services o ZeroC Ice.

También forman parte de la invención los procedimientos definidos para lograr la interoperabilidad entre dispositivos diferentes por medio de sendos módulos, por ejemplo la interconexión funcional transparente de un sensor y un actuador de distinta tecnología empleando un *middleware* de comunicaciones dado.

Debe hacerse una interpretación extensiva del concepto de *sensor* como todo aquel dispositivo capaz de recoger el valor de una magnitud física ya sea de modo pasivo (cuando se le interroga) o de modo activo, es decir, por iniciativa propia por ejemplo cuando cambia el valor de la magnitud medida. No está limitado el ámbito de uso de dichos sensores ni la naturaleza de la magnitud física. Esta definición involucra por tanto a sensores tan variados como un pulsador, un termómetro o una cámara de vídeo. Y del mismo modo, se puede aplicar la misma pauta para el concepto de *actuador*, pero teniendo presente que la operación sobre ciertos actuadores puede implicar consecuencias en el funcionamiento de sistemas complejos como, por ejemplo, la activación del motor de un ascensor, una electroválvula o un grupo electrógeno.

Cada módulo está formado por tres componentes principales: un elemento de cómputo, una interfaz de red y un transductor como interfaz con el dispositivo sensor/actuador.

A diferencia de las invenciones previas sobre miniaturización de *middlewares*, la presente invención utiliza un enfoque completamente innovador. La comunicación con el *middleware* se basa en el reconocimiento de mensajes estándar mediante autómatas de estados finitos y las respuestas se construyen a base de plantillas parametrizables garantizando mensajes coherentes con el protocolo utilizado por el *middleware*, por ejemplo GIOP(General Inter-ORB Protocol) en el caso de CORBA, o IceP en el caso de ZeroC Ice.

El primer paso para desarrollar cualquier aplicación distribuida utilizando un *middleware* de comunicaciones orientado a objetos es diseñar y declarar las interfaces entre los objetos y sus clientes. Dicha especificación se realiza normalmente en un lenguaje de programación específico, como pueden ser IDL (Interface Definition Language) en el caso de CORBA, Slice en el caso de ZeroC Ice o Java en el caso de Jini. Después, el ORB (o núcleo de comunicaciones) genera los mensajes para cada método, y con los tipos de datos correspondientes, en el protocolo adecuado. Obviamente la tarea de identificar un mensaje y construir un autómata reconocedor para los mensajes de la interfaz especificada puede ser una tarea realmente tediosa por lo que debe y puede ser automatizada.

La realización, por un lado, de la implementación empotrada en los módulos y por otro, de las aplicaciones que accederán a ellos, marca la frontera entre dos disciplinas claramente distintas con herramientas, procesos y responsabilidades diferentes.

La implementación de los módulos normalmente está estrechamente relacionada con uno o más dispositivos sensores o actuadores, de modo que generalmente tienen el rol de servidor. Las aplicaciones, que toman el rol de clientes, se diseñan e implementan de un modo idéntico al utilizado en un flujo de diseño estándar para aplicaciones distribuidas. Para el programador de aplicaciones que utiliza una red de sensores o actuadores accesibles por medio de módulos de integración, todo el proceso que ocurre dentro del módulo resulta transparente e indistinguible de un objeto distribuido convencional.

Por tanto, queda fuera del ámbito de esta invención cualquier herramienta o tecnología software necesaria para el desarrollo de la aplicación cliente, que habitualmente se instalará en un computador de propósito general. Por contra, sí se consideran como parte de esta invención los mecanismos que permiten a un módulo enviar el valor la magnitud de un sensor o actuador a un objeto remoto por medio de una invocación a través del *middleware* de comunicaciones. Por ello, aunque el objetivo principal del método descrito en el presente documento es la implementación de objetos distribuidos (servidores), ocasionalmente estos dispositivos necesitarán invocar mensajes sobre otros objetos bien conocidos; típicamente canales de eventos. Esta precisión es necesaria puesto que en esos casos el módulo actúa como cliente de un objeto remoto.

Interoperabilidad

En esta sección se describen brevemente algunas de las características de los protocolos de aplicación de CORBA (GIOP) e Ice (IceP). Estas características deben respetarse para lograr un grado de compatibilidad suficiente como para garantizar interoperabilidad básica de los picoObjetos con el *middleware*. Sirvan como ejemplo de las cuestiones a considerar a la hora de adaptar la invención a otros protocolos.

picoCORBA

PicoCORBA va mucho más lejos que MinimumCORBA en lo referente al recorte de capacidades. Los objetos picoCORBA no son en absoluto portables porque normalmente se implementan directamente en el lenguaje ensamblador de la plataforma concreta. Incluso utilizando C o algún lenguaje de bajo nivel, no se respeta en absoluto el *mapping* (mapa de correspondencia o de asociación) de OMG para esos lenguajes (si es que existe) puesto que no hay ninguna librería con la que enlazar ese código. Obviamente éste es un precio razonable a cambio de la sensible reducción de tamaño que se consigue a cambio.

El prototipo realizado es capaz de obtener un flujo de bytes procedente de la red y devolver una respuesta. El protocolo de transporte que se utilice para hacerlo posible puede ser muy variado: TCP, SLIP, SNAP, LonWorks o cualquier otro protocolo de transporte confiable o incluso, no confiable, si el *middleware* concreto lo permite.

Tal como se explica anteriormente, sólo hay dos puntos importantes en los que calcular sumas de verificación: cuando se ha recibido completamente el identificador del objeto y durante la recepción del nombre del método. Para simplificar aún más este proceso, los *object_key* de todos los picobjetos de un servidor tienen el mismo tamaño por lo que se sabe de antemano el byte exacto del mensaje en el que se debe realizar el cálculo. Utilizar *object_key* de tamaño fijo no causa problemas de interoperabilidad. Esa cadena aparece en la IOR (CORBA) y la norma dice que el cliente debe respetarla tal cual la lee.

Versión GIOP

La versión actual de picoCORBA es conforme a GIOP 1.0. Esto no implica restricción alguna pues la especificación de GIOP obliga a que todo cliente "hable" con el servidor en una versión de GIOP igual o inferior a la que aparezca reflejada en la referencia publicada por el servidor (su IOR), es decir, el servidor es quién decide.

Si el objeto picoCORBA soporta GIOP 1.0, y así aparece reflejado en la IOR correspondiente, cualquier cliente conforme al estándar debe poder interoperar con él usando la versión, y el servidor no está obligado a implementar nada más.

Ordenamiento de bytes

La especificación GIOP indica que el que inicia la comunicación determina cuál será el ordenamiento de bytes (*big endian* o *little endian*) mientras dure la conexión. En el caso de GIOP 1.0, el iniciador de la conexión siempre es el cliente. El servidor debe realizar la conversión cuando sea necesario y responder con el ordenamiento que indique el cliente.

En principio la aplicación de los picobjetos está orientada a entornos acotados en los que este aspecto se puede controlar, de modo que soportar sólo uno de los ordenamientos no plantea problemas de interoperabilidad. Para picoCORBA, dar soporte a los dos ordenamientos implica duplicar la cantidad de mensajes que puede reconocer y generar el dispositivo. Supone un sensible aumento de coste en lo referente a cantidad de recursos consumidos pero, para ciertas aplicaciones que puedan necesitarlo, se puede considerar oportuno.

Interfaz CORBA::Object

La interfaz *CORBA::Object* define un conjunto de métodos que todos los objetos CORBA deben soportar. Sin embargo, la mayor parte de estos métodos son atendidos por el proxy remoto del objeto o por el ORB local, de modo que no se materializan en invocaciones remotas al servidor.

El solicitante considera que las únicas operaciones realmente necesarias para un funcionamiento adecuado son non_existent() e is_a(); la primera permite comprobar si el objeto está vivo, la segunda ofrece capacidades de introspección muy necesarias. Como no hay ningún otro componente es el dispositivo, deberá ser la implementación generada por picoCORBA la que provea al dispositivo de la facultad para responder a estos dos métodos.

Mensajes GIOP

A continuación se explican las restricciones aplicables a los distintos mensajes, cuáles de ellos se soportan y en qué medida. Todas las puntualizaciones se refieren al capitulo 15 de la especificación CORBA, titulada, *General Inter-ORB Protocol*. Hay mensajes no soportados, pero los que se soportan no sufren ninguna modificación en el formato.

GIOP versión 1.0 incluye los siguientes mensajes:

- Petición (del cliente al servidor).
- Respuesta (del servidor al cliente).
- Cancelación de petición (del cliente).
- Petición de localización (del cliente).
- Respuesta de localización (del servidor).
- Cierre de conexión (del servidor).
- Mensaje de error (ambos).

Cabecera GIOP. No hay ninguna diferencia con la cabecera estándar salvo que el valor del campo GIOP_version es siempre 1,0 y el campo byte_order es constante en el caso de la implementación mínima.

Mensajes de petición. Corresponden a invocaciones de métodos del objeto remoto. Se toman las siguientes consideraciones:

- El campo *service_context* se ignora.
- El valor del campo *response_expected* se ignora. Los picobjetos siempre generan respuestas aunque no se soliciten. Es un comportamiento conforme al estándar porque los clientes ignoran a su vez respuestas no solicitadas.
- El campo *object_key* identifica al objeto sobre el que se invoca el método. Los picobjetos utilizan claves de objeto de tamaño constante. Esto es transparente para los clientes y no afecta a la interoperabilidad, pero resulta muy conveniente para el reconocimiento de los mensajes.
- El campo *requesting_principal* está obsoleto y se ignora.

Los mensajes de petición incluyen el valor de los parámetros del método y pueden ser reconocidos por el servidor picoCORBA conforme a la especificación de la interfaz IDL.

Mensajes de respuesta. Los mensajes de respuesta se generan como resultado de la invocación de un método. Los picoObjetos siempre generan un mensaje de respuesta si el mensaje de petición es correcto, en caso contrario no devuelven nada. El contenido del mensaje de respuesta depende del mensaje de petición y del resultado de la invocación del procedimiento de usuario correspondiente. Las consideraciones tomadas en este caso son:

- Los campos *service_context* y *request_id* son copias del valores recibidos en la petición.
- El campo *reply_status* siempre contiene *NO_EXCEPTION* ya que picoCORBA no soporta excepciones ni información de redirección de localización (*location forward*).

Mensajes de cancelación de petición. La especificación GIOP dice que los servidores pueden hacer caso omiso a este tipo de mensajes y aún habiéndolos recibido procesar la petición y generar un mensaje de respuesta. Por tanto, picoCORBA los ignora completamente y sin embargo cumple perfectamente la especificación.

Mensajes de localización. Este tipo de mensajes los utilizan los ORB clientes para localizar objetos. PicoCORBA no soporta este mecanismo y por tanto ignora estos mensajes.

Mensajes de finalización de conexión. Sirve para notificar a un cliente la desconexión de un servidor. Dado que los objetos picoCORBA se implementan en dispositivos empotrados, se entiende que son "always on" por lo que nunca necesitarán enviar este tipo de mensaje.

Mensajes de error. Se utilizan para que el receptor pueda informar al emisor de que está enviando mensajes incorrectos. Dado que picoCORBA ignora cualquier mensaje no soportado, también ignora los mensajes mal formados. PicoCORBA tampoco puede enviar mensajes incorrectos dado que los genera mediante un autómata.

Si eso llegara a ocurrir, significaría que hay un fallo en la implementación y debe reemplazarse. En cualquier caso, de nada serviría procesar informes de error dado que el servidor picoCORBA no tiene capacidad para recuperarse ante un problema de este tipo. Por eso, los picoObjetos ignoran los mensajes de error y nunca los generan.

picoIce

ZeroC Inc. ha desarrollado un núcleo de comunicaciones de objetos distribuidos llamado Ice (*Internet Communication Engine*) construido a partir de la experiencia en CORBA, pero libre de restricciones legales y burocráticas. Implementa más funcionalidades que cualquier otra plataforma de objetos distribuidos (persistencia y migración de objetos, autenticación, seguridad, replicación, servicios de despliegue, pasarelas firewall, etc.). En la referencia [2] hay disponible un resumen de las diferencias entre Ice y CORBA.

A pesar de la escasez de soporte para plataformas empotradas, Ice ofrece varias ventajas sobre CORBA en cuanto a la reducción de la cantidad de recursos necesarios. El protocolo IceP es más simple que GIOP debido a varias decisiones de diseño:

- Todos los mensajes son siempre *little endian* de modo que no es necesario preocuparse del ordenamiento de bytes.
- Tiene soporte para protocolos de transporte no confiables como, por ejemplo, UDP (mucho más fácil de implementar en dispositivos empotrados de bajo coste).
- Hay menos tipos de mensajes y es posible conseguir plena interoperabilidad a pesar de no implementar todos ellos.
- Los campos no procesados de los mensajes se pueden descartar fácilmente porque suelen estar precedidos de un campo que indica la longitud total.
- No hay requisitos de alineamiento de datos para los mensajes "en el cable".

Interfaz Ice::Object

Del mismo modo que en el caso de CORBA, todos los objetos Ice implementan por defecto la interfaz Ice::Object, que se muestra a continuación:

```
interfaceObject {
  void ice_ping();
  bool ice_isA(stringtypeID);
  string ice_id();
  StrSeq ice_ids();
};
```

Los objetos picoIce implementan todos estos métodos, por lo que está garantizado un comportamiento correcto también a nivel de interfaz estándar.

Mensajes IceP

El protocolo Ice utiliza los siguientes mensajes:

- Petición (del cliente al servidor).
- Petición por lotes (del cliente al servidor).
- Respuesta (del servidor al cliente).
- Validación de la conexión (del servidor al cliente).
- Cierre de conexión (ambos sentidos).

Los mensajes de validación y cierre de conexión sólo se utilizan en protocolos de transporte orientados a conexión.

PicoIce reconoce todos estos mensajes a excepción de la *Petición por lotes* sin que ello implique problema alguno de interoperabilidad, ya que dicho mensaje se utiliza como un mecanismo de optimización para baterías de peticiones.

Concretamente, el objeto PicoIce (como servidor) es capaz de reconocer mensajes de petición y cierre de conexión, y es capaz de generar mensajes de respuesta y validación.

Cabecera de mensajes. PicoIce utiliza cabeceras IceP estándar salvo por una restricción: la compresión de mensajes no está soportada. El campo *compressionStatus* siempre vale 0. Sin embargo, eso no plantea ningún problema de interoperabilidad pues la compresión de mensajes es opcional según la especificación del propio *middleware*.

Mensaje de petición. Son los mensajes enviados por el cliente que desencadenan la ejecución de un método en un objeto del servidor. El mensaje tiene los siguientes campos:

```
structRequestData {
  int requestId;
  Ice::Identify id;
  Ice::StringSeq facet;
  string operation;
  byte mode;
  Ice::Context context;
  Encapsulation params;
};
```

- El campo *requestid* es un entero que identifica el mensaje. Su valor se copia en el mensaje de respuesta.
- El campo *id* identifica al objeto sobre el que se invoca el método, de modo similar al *object_key* de GIOP. Por la misma razón también se utilizan claves de objeto de tamaño constante.
- El campo *facet*.

Los mensajes de petición incluyen el valor de los parámetros del método y pueden ser reconocidos por el servidor picoIce conforme a la especificación de la interfaz Slice.

Mensaje de respuesta. El mensaje de respuesta contiene el resultado de una invocación twoway, es decir, los parámetros de salida y el valor de retorno. Los campos de la respuesta son:

```
structReplyData {
  int requestId;
  byte replyStatus;
  Encapsulation body;
};
```

- *requestId* es el valor obtenido para el campo homónimo.
- El campo *replyStatus* indica el éxito de la operación o algún tipo de error. El picoObjeto mínimo cubre 2 de las 8 respuestas posibles. Un valor 0 indica éxito en la localización del objeto y ejecución del método. En caso de un valor erróneo para el identificador del objeto, el nombre del método u otro tipo de error el valor de este campo será Excepción Desconocida (7).

Es posible implementar una gestión más sofisticada para el tratamiento de errores que incluya reconocimientos de errores o disparo de excepciones. Evidentemente, ello es posible a costa de un aumento en el consumo de memoria. Como en otros casos, esta funcionalidad adicional se dará únicamente cuando la plataforma lo permita.

Mensaje de Validación de Conexión

Cuando un objeto (servidor) recibe una nueva conexión de un cliente (protocolo orientado a conexión) lo primero que hace el servidor es enviar un mensaje especial para validar la conexión. Sólo cuando el cliente ha recibido dicho mensaje empieza a enviar mensajes de petición. Se trata de un mensaje pequeño formado únicamente por una cabecera (14 bytes) y todo su contenido es completamente estático por lo que no representa ningún problema para ser implementado en picoObjetos.

Mensaje de cierre de conexión

Cuando un cliente ya no desea realizar más peticiones envía el mensaje *CloseConnection*. El objeto debe responder a su vez con otro mensajes *CloseConnection*. Un servidor Ice también puede enviar este mensaje para indicar al cliente que no va a seguir atendiendo sus peticiones. Dado que los picoObjetos se consideran “always on” nunca enviarán un mensaje de este tipo.

Características específicas de la invención

Se describen a continuación las características más importantes tanto del módulo como de los métodos que permiten obtener ventajas prácticas relevantes mediante su uso, todos ellos partes constituyentes de la invención que se describe en el presente documento.

Objetos activos

Los objetos activos son objetos distribuidos implementados mediante módulos que tienen la cualidad de notificar su estado de forma asíncrona, es decir, sin que exista una solicitud explícita por parte del que recibirá la información. Todo objeto activo dispone de una referencia a un segundo objeto que actúa como “observador”. En el caso general, este objeto “observador” es una canal de eventos, de modo que una cantidad arbitraria de *consumidores* (otros objetos distribuidos) se pueden subscribir para recibir las modificaciones que sufra el estado del objeto activo.

Se establecen dos tipos de objetos activos (no excluyentes):

- Los objetos reactivos envían mensajes al observador asociado cada vez que reciben un mensaje estándar ping, independientemente de quién sea el emisor de dicho mensaje.
- Los objeto proactivos envían mensajes al observador asociado cada vez que el valor de su estado cambia independientemente del motivo por el que ese hecho ocurra.

El *objeto activo* se emplea frecuentemente en la implementación de módulos sensores, de modo, que los potenciales interesados en el valor del sensor puedan tener información actualizada de dicho valor sin necesidad de realizar solicitudes continuamente. En cualquier caso, es posible realizar módulos actuadores que actúan como objetos activos. Ello permite a sus observadores disponer de información actualizada sobre los cambios que sufre el actuador, presumiblemente a consecuencia de terceros, normalmente clientes.

Soporte para consulta de grupos

Un problema típico en las redes de sensores en la necesidad de leer el valor de un conjunto de ellos en el mismo instante. Para proporcionar esta funcionalidad se utilizan también *objetos reactivos*. Cualquier servicio interesado en un conjunto de sensores, puede realizar la siguiente secuencia de operaciones:

- Obtiene las referencias a todos los objetos.
- Crea un canal de eventos de consulta, o localiza un módulo que ofrezca la misma funcionalidad.
- Subscribe a los objetos en el citado canal.
- Obtiene las referencias a los canales de eventos (observadores) asociados a cada objeto.
- Crea un segundo canal de recogida de resultados.
- Enlaza los observadores de los objetos sensores con el canal de resultados.
- El servicio se subscribe a sí mismo en el canal de resultados.
- Invoca un mensaje ping() sobre el canal de consulta.

En ese momento empezará a recibir invocaciones procedentes de cada sensor. Esos mensajes incluyen el valor de la magnitud medida y el identificador del objeto, de modo que el servicio puede averiguar de qué sensor procede cada valor siendo de gran ayuda para generar información de alto nivel como identificación de “zonas calientes”, gradientes, etc.

Anunciamientos

Los módulos, independientemente del uso al que estén destinados, tienen la posibilidad de darse a conocer en el entorno por medio de la invocación de un método específico en un objeto remoto conocido o en un canal de eventos si se dispone de él. Se definen dos alternativas básicas (no excluyentes) para su implementación:

- El módulo envía periódicamente un mensaje de anunciamiento. Tiene algunos inconvenientes: Reduce la vida de las baterías en el caso de sensores o actuadores independientes como es el caso de las redes de sensores inalámbricos. Por otro lado, genera tráfico innecesario a partir del momento en que todos los posibles interesados conocen su existencia.
- El módulo responde con un mensaje de anunciamiento cuando algún cliente o servicio interesado envía previamente un mensaje de descubrimiento.

El mensaje de anunciamiento incluye la información necesaria para localizar el dispositivo y enviar mensajes al objeto anunciado. Normalmente implica la dirección lógica o física en la red de comunicaciones que permite enviar mensajes al dispositivo y algún tipo de identificador único que permite localizar dentro del dispositivo al objeto concreto que ha de atender dicho mensaje. En el caso de CORBA, esta información es una IOR (*Interoperable Object Reference*) mientras que el caso de ZeroC Ice es un *proxy*.

El anunciamiento hace posible una instalación muy simple de nuevos dispositivos pues no se requiere configuración previa. El sistema puede asumir y configurar nuevos dispositivos tan pronto como aparezcan tanto de forma automática o guiada por un operador humano según el caso.

Métodos de integración

La utilización de un módulo asociado a un sensor o actuador facilita sensiblemente la integración en cualquier sistema automatizado o sistema de información, puesto que el acceso al sensor/actuador se hace por medio de un API local proporcionada por el *middleware*, de un modo predecible y estandarizado. Los sensores/actuadores son entidades básicas en el modelado del sistema simplificado los procesos de captura de requisitos, análisis y diseño, y sobre todo

durante la implementación. El personal involucrado en el desarrollo no requiere formación específica, sea cual sea la tecnología empleada por el fabricante de los dispositivos o la red de datos utilizada.

Integración de pares

Es posible "enlazar" un sensor con un actuador remoto de modo que una variación en la magnitud medida por el sensor pueda desencadenar una acción concreta en un actuador. En esta configuración tanto el sensor como el actuador disponen de un módulo conectado a una red común. El escenario más básico es aquel en el que el sensor dispara la ejecución de un método del actuador al detectar un cambio en su entrada.

Esto solo tiene sentido cuando el tipo del sensor y el actuador son compatibles, por ejemplo, un sensor de presencia todo/nada que activa la iluminación de una habitación. Si bien se trata de un servicio extremadamente simple, tiene una gran ventaja: no existe ningún intermediario y sin embargo, no es un sistema aislado.

- Tanto el sensor como el actuador pueden ser operados por terceros sin romper la relación que existe entre ambos.
- La relación es persistente pero se puede cambiar remotamente. Un operador del sistema puede modificar la configuración del sensor por medio de invocaciones remotas para elegir otro actuador distinto.

Este modo de integración "uno a uno" corresponde con un direccionamiento *unicast*. El sensor solo conoce la dirección lógica de un actuador en un momento dado.

Direccionamiento

Por medio de canales de eventos es posible flexibilizar el escenario anterior. Un canal de eventos se puede implementar por medio de direcciones de red especiales cuando la tecnología de red lo soporta. Cuando las limitaciones de dicha tecnología lo impidan, se pueden utilizar servicios proporcionados por el *middleware* para este fin.

El canal de eventos es un objeto distribuido más en el que se pueden registrar todos aquellos interesados en el valor de un sensor. El sensor alimenta el canal con el valor de la magnitud medida y los actuadores se registran en el canal, de modo que el canal de eventos ejerce como cliente de los actuadores propagando la información original. De ese modo se pueden conseguir direccionamientos *broadcast* (todos), *multicast* (todos los de un grupo) o *any-cast* (uno del grupo).

Sin embargo, es posible que la implementación del *middleware* utilizado no soporte tecnologías de red presentes en la instalación previa. Incluso en ese caso, sin la disponibilidad de los servicios estándar del *middleware*, es posible llevar a cabo estos modos de direccionamiento.

La solución pasa por implementar un objeto distribuido mediante un módulo que carece de sensores o actuadores. Este objeto presenta una interfaz remota similar a la de un canal de eventos estándar como lo que proporciona el *middleware*. Por medio de este interfaz, los consumidores pueden registrarse en dicho canal. El sensor obtiene una referencia al objeto que tiene el rol de canal y la utiliza para realizar invocaciones que serán propagadas a todos los actuadores registrados. Asimismo, estos canales-módulo pueden proporcionar interfaces remotas que permitan gestionar el canal o aplicar filtros de forma dinámica.

Integración transparente

En algunas tecnologías de red específicas en ambientes industriales es habitual que se presenten limitaciones eléctricas o topológicas que impidan o dificulten el uso de ciertos tipos de sensores/actuadores por sus características técnicas o por circunstancias específicas de un despliegue concreto.

En estos casos, es posible integrar un módulo que proporcione una interfaz compatible con la tecnología previa pero que utilice invocaciones a un objeto remoto para conseguir la finalidad del sensor/actuador sobre el que se desea operar realmente. Este módulo actúa como cliente de un objeto remoto, que es el que realmente tiene acceso físico al sensor/actuador. De ese modo se consigue integración transparente con la red y la tecnología del fabricante pero independizando la tecnología real del sensor/actuador. Además, otros clientes pueden compartir dicho sensor/actuador sin interferir en el funcionamiento del sistema previo, siempre que se respeten sus restricciones temporales.

Interconexión lógica dinámica

La utilización de un *middleware* común proporciona un entorno en el que resulta sencillo ofrecer conectividad lógica de cualquiera a cualquiera. Un sistema que incorpore módulos activos sensores/actuadores puede instalarse sin configuración previa. Después, un operador o administrador puede establecer relaciones directas entre módulos sen-

sores y actuadores, o de estos con servicios del sistema implementados sobre computadores convencionales utilizando las librerías proporcionadas por el fabricante del *middleware*. Dicha configuración se realiza también utilizando las interfaces remotas que ofrecen los módulos como objetos distribuidos que son.

Este método puede resultar muy conveniente en edificios o instalaciones industriales en las que se deben satisfacer escenarios de uso muy variados. Por ejemplo, la iluminación de un recinto de exposiciones puede estar controlada por medio de objetos distribuidos individuales mediante la integración de un módulo a cada punto de luz. El operador puede asociar uno o varios interruptores físicos (sensores) con uno o varios puntos de luz (actuadores) (ver apartado de *Direccionamiento*). Estas relaciones pueden establecerse muy fácilmente y cambiar frecuentemente, adaptándose a las necesidades concretas de cada evento y stand. También existe la posibilidad de disponer de una consola central para poder operar sobre cualquiera de los puntos de luz (módulos actuadores) o ver el estado de cada interruptor o pulsador (módulos sensores). El sistema completo puede ser autónomo y no requerir ninguna infraestructura adicional aparte de los propios módulos y la red de comunicaciones. Este método de interconexión lógica de módulos puede verse como un "cableado virtual".

Modos de realización de la invención

La implementación del módulo integra dispositivos sensores y/o actuadores por medio de los transductores adecuados, una o más interfaces de red y un dispositivo de cómputo. La elección de este dispositivo de cómputo determina dos modos de realización de la invención bien diferenciados:

Estrategia software

En ese caso, el dispositivo de cómputo empleado está basado en un microcontrolador o microprocesador de gama baja. La funcionalidad del módulo está fundamentada principalmente en un aplicativo software instalado en dicho microcontrolador.

La forma más simple para conseguir un comportamiento coherente en cada picoObjeto es utilizar un autómata reconocedor. En ese contexto, el conjunto de mensajes posibles para cada picoObjeto constituye una gramática BNF definida por:

- El formato de mensajes utilizado por el protocolo de comunicación del middleware.
- La identidad de cada objeto. Hay que tener en cuenta que varios objetos pueden compartir el mismo código funcional. Esta técnica se llama "sirviente por defecto" (*default servant*).
- La interfaz o conjunto de interfaces que ofrece cada objeto. Eso incluye el nombre, el tipo de los argumentos y el valor de retorno de cada método.
- El procedimiento de serialización de datos (CDR en el caso de CORBA).
- Las interfaces estándar heredadas del núcleo de comunicaciones (CORBA::Object en CORBA).
- Restricciones específicas de la plataforma objetivo.

Primero se define un conjunto de elementos léxicos básicos (*tokens*), es decir, campos obligatorios o no que aparecen en cada mensaje y que tienen un formato y tamaño conocido tales como el nombre del objeto, el método, la interfaz. A continuación, se genera el conjunto de reglas que describen, la forma en que se pueden combinar dichos *tokens* (la gramática). Con esta información es posible generar un autómata reconocedor funcional completo. El flujo completo de desarrollo se muestra en la figura 10.

Todo picoObjeto debe incluir un conjunto de procedimientos de usuario que materializan la implementación de los métodos de los objetos. Estos fragmentos de código debe aportarlos el programador del dispositivo y son específicos de cada arquitectura concreta, al igual que ocurre en cualquier middleware tradicional. Cuando el autómata de un picoObjeto identifica un mensaje de petición completo, se dispara automáticamente la ejecución del procedimiento de usuario asociado. Como consecuencia se genera y envía un mensaje de respuesta. Sí el autómata no es capaz de identificar un mensaje de petición válido, el flujo de entrada se descarta hasta el siguiente punto de sincronización.

Tanto los mensajes de entrada (peticiones) como los de salida (respuestas) se pueden manejar *al vuelo* usando un procesamiento a nivel de bytes específicamente diseñado. Esta es una solución muy conveniente para dispositivos con limitaciones de memoria muy severas (sólo unos cuantos cientos de bytes de memoria de programa y poco más de una docena de registros de propósito general) . Es estas condiciones no es posible almacenar un mensaje de entrada completo. El mensaje de petición se procesa conforme va llegando y el mensaje de respuesta también se genera parcialmente copiando parte de los datos de entrada. La última parte del mensaje de respuesta se genera por medio del procedimiento de usuario asociado a cada método.

Para reducir los requerimientos de memoria en la implementación del autómata, utilizamos una suma de verificación para la comprobación de los tokens. Incluso cuando los tokens son arbitrariamente largos, el compilador de picoObjetos los sustituye por un único byte. Mientras el picoObjeto está reconociendo un mensaje de entrada, va calculando un suma de comprobación de modo incremental. Ese cálculo se comprueba cuando el número de bytes consumidos alcanza el esperado. Realmente no es necesario comprobar en el limite de cada token si el cálculo obtenido y el esperado corresponden. Si el conjunto de mensajes posibles para un objeto se organiza como un árbol lexicográfico sólo se necesita comprobar los puntos de bifurcación para decidir qué rama seguir. En muchas ocasiones el compilador puede optimizar aún más esta tarea y calcular las sumas de verificación sólo para los mensajes completos, exceptuando los argumentos.

Nuestra estrategia es muy diferente a las aproximaciones previas de miniaturización de middlewares como MicroQoSCORBA, que intenta una miniaturización de código basada en la construcción de implementaciones de ORB específicos a partir de librerías predefinidas. Los picoObjetos usan una estrategia con mayor granularidad. Genera completamente el código que reconoce y los mensajes para cada aplicación.

El enfoque anterior se ha aplicado a dos middlewares existentes: CORBA y ZeroC Ice, que se denominan picoCORBA y picoIce respectivamente. Las restricciones que impone cada middleware particular implican decisiones de diseño diferentes. Estas decisiones han sido comentadas con detalle en la sección “Interoperabilidad”.

Los dos prototipos se han desarrollado en lenguaje ensamblador del Microchip PIC. Java para un PC empotrado estándar, Java sobre un dispositivo Dallas Semiconductors TINI y C sobre un PC convencional.

Estrategia hardware

La segunda alternativa de realización de la invención propone la creación e integración con objetos hardware. Un objeto hardware (a partir de ahora HwO) es un circuito integrado específico que realiza una tarea compleja en una red, como una red de sensores y actuadores.

La figura 12 muestra el flujo de trabajo del proceso de generación de HwO. La entrada a este proceso es una descripción de la interfaz del objeto escrita en lenguaje IDL o Slice, de modo que otros objetos del sistema puedan ver el componente hardware. Como se ha dicho, un objeto hardware ofrece una interfaz característica que determina el modo en que se puede acceder a su funcionalidad (protocolo de acceso al objeto). Esa información se utiliza para generar:

- La infraestructura de comunicaciones dentro del chip.
- La implementación del objeto hardware final. Esta implementación puede ser el resultado de un proceso de integración, reutilizando diseños existentes. A veces será necesario implementar nuevos diseños si la aplicación concreta lo requiere. Señalar que este es un proceso completamente automático. Finalmente, se sintetiza el código VHDL (lenguaje estándar de descripción de hardware) resultante.

Las redes de sensores se modelan como un conjunto de objetos que se comunican por medio de paso de mensajes. Conceptos tales como atributos, métodos o parámetros son habituales en la comunidad de desarrollo de software dado que han sido ampliamente aplicados al desarrollo de todo tipo de proyectos durante décadas; ese no es el caso del hardware. Por eso es obligatorio establecer un modelo de objetos hardware para reducir la brecha conceptual entre el mundo del software y el hardware. Los beneficios de ver los componentes hardware como objetos son:

- Permite una integración transparente entre objetos software y hardware sin afectar al diseño del sistema.
- Hace posible la generación automática de infraestructura de comunicaciones debido a la aportación semántica que implica dentro y fuera del chip.

Para implementar objetos lógicos, como componentes hardware, se definen:

- Una interfaz estándar para automatizar la generación de adaptadores y,
- Un mecanismo de invocación de métodos que debe darle soporte.

Un objeto hardware ofrece una interfaz hardware característica:

- Señales comunes en el sistema: reloj, reset, etc.
- Una señal de entrada por cada método, para activar tu ejecución.

- Una señal de salida por cada método para indicar el fin de una operación, que se usa para propósitos de sincronización.

- Varios puertos de datos de entrada/salida.

Nuestro modelo de objetos hardware utiliza el sistema de tipos de datos de Ice para asegurar una semántica de comunicación ínter-componente. En el momento de escribir este documento es posible utilizar en los HwO todos los tipos básicos (*bool, short, int, float*, etc.), estructuras, secuencias de tamaño fijo y cualquier combinación de ellos.

La figura 13 ilustra las interfaces para un objeto que implementa un buffer de enteros. El solicitante ha desarrollado una herramienta (HAP, HwO Access Protocol) que genera la información de sincronización sobre el protocolo de invocación de métodos hardware y el código VHDL del autómata de estados finitos que reconoce el protocolo de los mensajes de entrada.

Plataforma de Objetos Hardware

Un HwO únicamente implementa la funcionalidad para la que ha sido diseñado. En este trabajo se desacoplan intencionadamente las comunicaciones del comportamiento para hacer módulos reutilizables en futuros diseños. Aislando los HwO de los detalles de implementación de las comunicaciones se consigue que no se vean afectados por cambios imprevistos en la infraestructura de comunicaciones. Para que los HwO sean accesibles desde fuera del chip, definimos una plataforma de objeto hardware (figura 14).

Se pueden agrupar varios HwOs en una única plataforma HwO, compartir recursos y reducir costes de implementación. Cada objeto hardware del sistema tiene un identificador único diferente del OID. Realmente, este identificador corresponde con la dirección base del módulo que implementa el esqueleto. El HwOA es responsable de la traducción: de OID externos a direcciones válidas para el bus interno. Tanto los proxies como los esqueletos coinciden en la forma en la que las invocaciones a métodos se traducen en una secuencia de acciones de bajo nivel sobre el bus para activar la ejecución de la operación. Las reglas de codificación/decodificación son las mismas que en la versión software. Por tanto, la invocación de un método se descompone en primitivas de lectura/escritura. Las lecturas y escrituras son los servicios básicos que ofrecen la mayoría de los buses de modo que se puede implementar en casi cualquier plataforma, independientemente de la tecnología.

Generación de la Infraestructura de Interconexión

La generación de esqueletos (los adaptadores de objetos hardware) y el HwOA que implica la generación de *proxies* es un paso crítico en el flujo de diseño hardware. La implementación final está optimizada para cumplir los requisitos de bajo coste manteniendo el proceso automático para ahorrar tiempo de diseño.

Se definen en esta memoria dos plantillas que se especializarán de acuerdo al número y tipo de métodos a implementar. La figura 15 muestra la arquitectura general de un *proxy*. La figura no incluye el esqueleto por cuestiones de espacio. Los módulos más importantes son las Port Acquisition units (PAU) y el Port Delivery Unit (PDU) que son responsables de la adaptación de los datos recibidos (HwO o bus) de acuerdo con el protocolo de acceso al objeto impuesto por la interfaz HwO.

La *Combinational Adaptation Logic* (CAL) traduce las señales de control en comandos del bus específicos y viceversa. La correspondencia entre las señales de control del FSM y las señales de control del bus se definen por separado en ficheros de configuración.

La lógica de control implementa el FSM materializando la adaptación al protocolo local (acceso al objeto) y el remoto (bus de comunicaciones).

Sólo se produce un esqueleto para cada objeto hardware que exporte al menos un método. Sólo se genera un *proxy* (realmente un esqueleto) para cada método que pueda ser invocado remotamente por el cliente. Esto reduce la lógica usada en el *proxy* dado que no se genera nada para los métodos que no se usan.

Se han introducido otras optimizaciones reutilizando la lógica para dos o más definiciones de métodos (que no tienen porqué corresponder al mismo objeto/clase) que tengan la misma implementación hardware física.

Descripción detallada con referencia a las figuras

En la implementación de un *middleware* convencional, una invocación remota (figura 3) involucra a un adaptador de objetos, un mapa de objetos y un sirviente, todo ello en el marco de un servidor.

Por contra, el mismo escenario cuando se utiliza uno de los módulos objeto de la invención (figura 4) requiere únicamente el reconocimiento del mensaje de invocación y la generación de la respuesta, sin intervención de ningún otro elemento software.

Cuando el módulo se utiliza como medio de acceso a un sensor/actuador está constituido por tres componentes (figura 1). El primero es una interfaz de red de cualquier tecnología, el segundo es un dispositivo de cómputo y por último un transductor que adapta y controla las señales que se envían y proceden del sensor/actuador.

Una configuración alternativa del módulo (figura 1) se emplea para envío de mensajes uno a muchos a modo de canal de eventos sencillo.

La figura 5 ilustra un escenario básico de uso de un objeto activo. Se trata de un sensor que notifica a un segundo objeto "observador" cualquier cambio que ocurra en la magnitud que mide. El objeto "observador" puede ser un objeto convencional implementado con las librerías que proporciona el fabricante del *middleware*, pero también puede tratarse de objeto empotrado que utiliza uno de los módulos descritos en este documento.

La figura 6 muestra otro escenario de uso para objetos activos. En este caso, el objeto "observador" es un canal de eventos, es decir, un objeto que puede reenviar a otros las invocaciones que recibe, y de ese modo materializar las posibilidades de propagación de eventos descritas en el apartado *Objetos Activos*, mencionado anteriormente. Como antes, este *canal de eventos* lo puede proporcionar el *middleware* o se puede construir por medio de un módulo refsec:direccionamiento.

El uso de objetos activos (en su modalidad de reactivos) permite implementar un método de consulta de un grupo arbitrario de objetos sensores/actuadores por medio del uso de canales de eventos. La figura 7 representa la secuencia de operaciones que se produce durante el establecimiento del mecanismo y cómo la aplicación recibe los resultados.

En la figura 8 se muestra el uso de un módulo con la configuración de la figura 2 para lograr la integración de un sensor remoto, que utiliza una tecnología distinta. Se trata de integrar en una red de propósito específico (como una red control industrial) un dispositivo en principio incompatible. Se utiliza un módulo que hace de puente entre redes distintas y otro módulo para integrar el sensor/actuador. Cuando la red de propósito específico direcciona el dispositivo, el módulo realiza automáticamente una invocación remota al objeto remoto empotrado que realmente está en disposición de realizar el acceso al sensor/actuador.

La figura 9 muestra los roles del programador de servicios (lado derecho) y del programador de módulos (lado izquierdo) y su relación con las especificaciones de interfaces y código necesario.

La aplicación que se instala en el módulo se obtiene mediante el flujo de trabajo descrito en la figura 10. Las entradas corresponden con los rectángulos con fondo gris. El desarrollador debe aportar:

- Un fichero de definición de interfaces (Slice, IDL, etc.).
- Un fichero que indica los objetos que albergará el módulo y las interfaces que implementa cada uno de ellos.
- Un fichero con la implementación de los métodos de los objetos, en el lenguaje nativo del dispositivo de cómputo utilizado en el módulo.

La figura 11 muestra el esquema de un compilador concreto como ejemplo de implementación del proceso descrito en la figura 10 para el middleware ZeroC Ice. Se parte de un conjunto de ficheros Slice en los que se encuentran las interfaces de los objetos, y un fichero en lenguaje IcePick, hasta la generación del *bytecode*, que identifica las instancias concretas y sus tipos.

El frontal del compilador (*frontend*) procesa los ficheros de entrada y genera un especificación intermedia que incluye los objetos descritos, sus tipos y demás peculiaridades. Otro proceso se alimenta de esta especificación y genera un fichero intermedio por medio de un juego de instrucciones virtual (*Virtual ISA*). Este lenguaje llamado FSM (*Finite State Machine*) es una especificación concreta de las máquinas de estados que reconocen los mensajes de petición y generan los mensajes de respuesta.

En el siguiente paso, un compilador toma dicho fichero FSM como entrada y genera código en algún lenguaje de implementación, como por ejemplo C. Por último se enlaza el fichero generado junto con otro fichero proporcionado por el usuario (llamado *usercode* y escrito en el mismo lenguaje de implementación. El fichero *usercode* debe escribirse explícitamente para la plataforma destino e incluye el código que se ejecutará como respuesta a las invocaciones recibidas, lo que normalmente implica acceder a hardware especifico que permite leer/modificar el valor del sensor/actuador conectado al módulo.

El flujo de trabajo para el segundo modo de realización de la invención, llamados HwO (objetos hardware), está representado en la figura 12. El proceso es similar al de la figura 9 si bien los resultados son diferentes.

La figura 13 es un ejemplo de objeto *hardware* para un conjunto (*buffer*) de enteros. Se muestran las señales de sincronización necesarias para su integración en el diseño.

Los HwO no tienen capacidad de comunicación con el exterior. Para ello se sirven de la "Plataforma de Objetos Hardware" (figura 14). De ese modo se consigue un desacoplo muy conveniente ante cambios en la infraestructura.

La conexión del HwO al bus se realiza por medio de un proxy. La figura 15 muestra la configuración habitual de un proxy, incluyendo la adaptación de los datos procedentes del bus por medio de las *Port Acquisition Units* (PAU) y el *Port Delivery Unit* (PDU).

Descripción de las referencias numéricas de las figuras

Figura 1

1.- Comunicaciones (eth, BT, zigbee...).

2.- Control (PIC, TINI, MICA...).

3.- Transductor (sensor o actuador).

4.- Red.

5.- Entorno físico.

Figura 2

1.- Interfaz de red tecnología A.

2.- Control (PIC, TINI, MICA...).

3.- Interfaz de red tecnología B.

4.- Red tecnología A.

5.- Red tecnología B.

Figura 3

1.- Cliente.

2.- Proxy.

3.- Adaptador de objetos.

4.- Mapa de objetos activos.

5.- Sirviente.

Figura 4

1.- Cliente.

2.- Proxy.

3.- PicoObjeto.

Figura 5

1.- Sensor.

2.- Control.

3.- Interfaz de red.

4.- Objeto distribuido, ya sea estándar o implementado como módulo.

Figura 6

1.- Sensor.

2.- Control.

3.- Interfaz de red.

4.- Canal de eventos, sea convencional o implementado como módulo.

5.- Objeto distribuido, convencional o módulo.

Figura 7

1.- Aplicación.

2.- Canal de respuesta.

3.- Canal de consulta.

4.- Objeto, sensor/actuador.

5.- Observador del objeto.

Figura 8

1.- Red de propósito específico.

2.- Interfaz de red.

3.- Control.

4.- Interfaz de red.

5.- Interfaz de red.

6.- Control.

7.- Sensor/actuador.

Figura 9

1.- Limitaciones del dispositivo, Consumo de energía, Cobertura de RF, Plataformas...

2.- Definición de objetos notificando las interfaces que implementan.

3.- Definición de interfaces (IDL, Slice, etc..).

4.- Objetos distribuidos, Servicio básicos.. Características del sistema, Escalabilidad.

5.- Compilador de interfaces para picoObjetos.

6.- Compilador Slice estándar.

7.- Implementación de los métodos (C, asm, vhdl).

8.- Especificación del autómata reconocedor.

9.- “Cabos” para el cliente.

10.- Código de la aplicación (C++, Java, C#).

11.- PicoObjeto (servidor).

12.- Aplicación (cliente).

Figura 10

1.- Definición de objeto, indicando las interfaces que implementan.

2.- Definición de interfaces (IDL, Slice, etc..).

3.- Controlador de interfaces para picoObjetos.

4.- Implementación de los métodos (C, asm, vhdl).

5.- Especificación del autómata reconocedor.

6.- PicoObjeto (servidor).

7.- Máquina virtual portable para picoObjetos.

8.- Plataforma concreta.

Figura 11

1A.- SLICE source.

1B.- Icepick source.

2.- Slice Parser.

3.- Slice symbol table.

4.- Icepick parser.

5.- Icepick symbol table.

6.- Symbol table <fachada>.

7.- Otros backends.

8.- Plantillas FSM.

9.- Binario (otros).

10.- Binario ELF (PC).

11.- Binario MICA2 (Tiny OS).

12.- Compilador C.

13.- Dependiente de la plataforma.

14.- FSM backend.

15.- FSM file.

16.- Máquina virtual C.

17.- Máquina virtual Pitón.

18.- Máquina virtual PIC.

19.- bytecode asm PIC.

20.- bytecode Pitón.

21.- bytecode C.

22.- Makefile.

23.- FSM compiler.

24.- Restricciones de la plataforma

Entradas proporcionadas por el programador:

Referencias numéricas 9, 10, 11, 15, 19, 20, 21, 22

Resultados generados por el compilador: Resto de referencias numéricas.

Figura 12

1.- Descripción Slice.

2.- Generación del interfaz para el HWO.

3.- Object access Protocol Wrapper.

4.- Repositorio de IP.

5.- Biblioteca de plantillas.

6.- Integración de IP.

7.- Desarrollo de la lógica del objeto.

8.- Generación esqueleto/proxy/AO.

9.- Código VHDL.

Figura 14

1.- Sensor/actuador.

2.- Adaptador HwO.

3.- Tablas de conversión OID.

4.- Interfaz de red.

Figura 15

1.- Interfaz con IP.

2.- PDU.

3.- FIFO de retorno.

4.- LÓGICA DE CONTROL.

5.- PAU.

6.- FIFO de paráms.

7.- Lógica de adaptación de combinaciones.

8.- DIRECCIÓN DEL MÉTODO.

9.- Interfaz con el bus/red.

Referencias

[1] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, ed. 2.3, June 1999. Available in http://www.omg.org/, document id: 98-12-01.

[2] M. **Henning**, M. **Spruiell**. Distributed Programming with Ice, 2006, available online at http://www.zeroc.com/ (Last visited 26-2-2007).

[3] Web Services Architecture, available online at *http://www.w3.org/TR/ws-arch/*.

[4] Sun Microsystems, *Jini Architecture Specification*, ed. 1.2, available online at http://www.sun.com/ (Last visited 26-2-2007).

[5] Object Management Group, *Minimum CORBA Specification*, ed. 2.3, August 2002, available online at (Last visited 26-2-2007) http://www.omg.org/, document id: 02-08-01.

[6] **Rodrigues**, G., **Ferraz**, C. *A CORBA-Based Surrogate Model on IP Networks*, In Proceedings of the 21st Brazilian Symposium on Computer Networks, 2003.

[7] Fabio **Kon**, F. **Costa**, G **Blair**, Roy **Campbell**. *The Case for Refiective Middleware*. Communications of the ACM: special issue in adaptive middleware, 2002; Volume 45, Issue 6, pp 33-38.

[8] Manuel **Román**, M. **Dennis**, Mickunas, Fabio **Kon** and Roy **Campell**. *LegORB and Ubiquitous CORBA*, In Proceedings of IFIP/ACM Middleware '2000 Workshop on Reflective Middleware, Feb 2000; pp. 1-2.

[9] M. **Román**, Fabio **Kon**, Roy H. **Campbell**, *Reflective Middleware: From Your Desk to Your Hand*, IEEE Distributed Systems Online, 2001; 2(5).

[10] The ACE ORB, available online at *http://www.theaceorb.com/*.

[11] PrismTech, OpenFusion e*ORB Real-time Embedded Whitepaper, available online at (Last visited 26-2-2007) http://www.prismtechnologies.com/.

[12] **Haugan**, Olav. *Configuration and Code Generation Tools for Middleware Targeting Small, Embedded Devices*, M.S. Thesis, Dec 2001, Washington State University.

[13] V. **Subramonian**, G. **Xiang**. *Middleware Specification for Memory-Constrained Networked Embedded Systems*, In 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2004; pp 306-313.

[14] SaJe, Real-Time Native Java Execution. Available online at *http://saje.systronix.com/*.

[15] Tiny Internet Interface. Available online at *http://www.ibutton.com/TINI/index. html*.

[16] J. **Morena**, F. **Moya**, J.C. **López**. *Implementación de un ORB para Dispositivos Empotrados*, Symposium on Informatics and Telecommunications, Sevilla, Sep 2002, p 25-27.

[17] M. **Román**, A. **Singhai**, *Integrating PDAs into Distributed Systems: 2K and PalmORB*, In Proceedings of International Symposium on Handheld and Ubiquitous Computing, 1999, pp. 137-149.

[18] F. **Moya**, J.C. **López**. *SENDA: an alternative to OSGi for large-scale domotics*, Networks, The Proceedings of the Joint International Conference on Wireless LANs and Home Networks (ICWLHN 2002) and Networking (ICN 2002), World K Scientific Publishing, pp 165-176, Aug 2002.

[19] W. **Nagel**, N. **Anderson**. *A Protocol for Representing Individual Hardware Devices as Objects in a CORBA Networt*, Real-time and Embedded Distributed Object Computing Workshop, Julio 2002.

[A] F. **Moya** D. **Villa** F.J. **Villanueva** J. **Barba** F. **Rincón** J.C. **López** J. **Dondo** "Embedding Standard Distributed Object-Oriented Middlewares in Wireless Sensor Networks Wireless". Communications and Mobile Computing Journal, March 11, 2007.

[B] D. **Villa** F.J. **Villanueva** F. **Moya** F. **Rincón** J. **Barba** J.C. **López** "Embedding a general purpose middleware for seamless interoperability of networked hardware and software components" International Conference on Grid and Pervasive Computing, May 1, 2006.

[C] D. **Villa** F.J. **Villanueva** F. **Moya** J. **Barba** F. **Rincón** J.C. **López** "Minimalist Object Oriented Service Discovery Protocol for Wireless Sensor Networks" International Conference on Grid and Pervasive Computing, May 13, 2007.

[D] D. **Villa** F.J. **Villanueva** F. **Moya** F. **Rincón** J. **Barba** J.C. **López** "ASDF: An Object Oriented Service Discovery Framework for Wireless Sensor Networks" International Journal of Pervasive Computing and Comunications, December 2008.

REIVINDICACIONES

1. Módulo para la interconexión de objetos distribuidos, que aloja al menos un objeto distribuido accesible por un *middleware* de comunicaciones de propósito general que mediante una máquina de estados finitos, reconoce mensajes de petición procedentes de clientes remotos convencionales y genera respuestas de acuerdo a los mensajes de petición, **caracterizado** porque está implementado en un circuito integrado específico y comprende:

- un elemento de cómputo implementado en un microcontrolador del circuito configurado para procesar los mensajes de petición y generar las respuestas,
- al menos un dispositivo de comunicaciones conectable a una red de datos a través del que el elemento de cómputo recibe los mensajes y envía respuestas hacia otros módulos o hacia objetos distribuidos remotos,
- una máquina de estados finitos específica para el reconocimiento de los mensajes necesarios para interactuar con los objetos distribuidos alojados en el módulo,
- un transductor para acceder al estado de uno o varios objetos hardware asociados a objetos distribuidos alojados en el módulo.

2. Módulo según la reivindicación 1, **caracterizado** porque el transductor se conecta a al menos un objeto hardware que es un sensor para leer su estado.

3. Módulo según la reivindicación 1, **caracterizado** porque el transductor se conecta a al menos un objeto hardware que es un actuador para leer o modificar su estado.

4. Módulo según la reivindicación 1, **caracterizado** porque el al menos un dispositivo de comunicaciones tiene definidos al menos un punto de acceso local para los objetos alojados en el módulo y al menos un punto de acceso remoto para interoperar con al menos un objeto distribuido remoto independientemente del tipo de objeto hardware asociado al objeto distribuido remoto.

5. Módulo según cualquiera de las reivindicaciones anteriores, **caracterizado** porque el dispositivo de comunicaciones envía mensajes de invocaciones asíncronas a un objeto distribuido remoto, configurado previamente, para informar del estado del objeto distribuido alojado en el módulo.

6. Módulo según cualquiera de las reivindicaciones anteriores, **caracterizado** porque la máquina de estados finitos está configurada para reconocer, dentro de los mensajes de peticiones, mensajes de invocaciones a objetos distribuidos alojados en el módulo y, de acuerdo a dichos mensajes de invocaciones, ordenar al elemento de cómputo la generación de una respuesta.

7. Módulo según cualquiera de las reivindicaciones anteriores, **caracterizado** porque el dispositivo de comunicaciones usa un canal de eventos, que es un objeto distribuido en el que se registran todos los objetos hardware y se vinculan al estado de un determinado objeto hardware asociado a un objeto distribuido alojado en el módulo, vinculando objetos hardware de igual o diferente fabricante y tecnología.

8. Módulo según la reivindicación 7, **caracterizado** porque el dispositivo de comunicaciones usa direccionamiento, que se selecciona entre *broadcast*, multicast o any-cast, para enviar un mensaje con datos procedentes de un objeto distribuido alojado en el módulo hacia todos, un grupo o al menos uno de los objetos remotos direccionados.

9. Módulo según cualquiera de las reivindicaciones anteriores, **caracterizado** porque el dispositivo de comunicaciones envía mensajes a través de una red de datos local a módulos o a objetos hardware conectados a otra red de datos remota.

10. Módulo según la reivindicación 9, **caracterizado** porque el dispositivo de comunicaciones del módulo se comunica con un módulo puente conectado a una red de datos remota para enviar mensajes de invocación a través del módulo puente a otro módulo que aloja un objeto distribuido asociado a cierto objeto hardware.

11. Módulo según la reivindicación 10, **caracterizado** porque aloja un objeto distribuido asociado un objeto hardware que se selecciona entre sensor y actuador conectado a una red de control industrial.

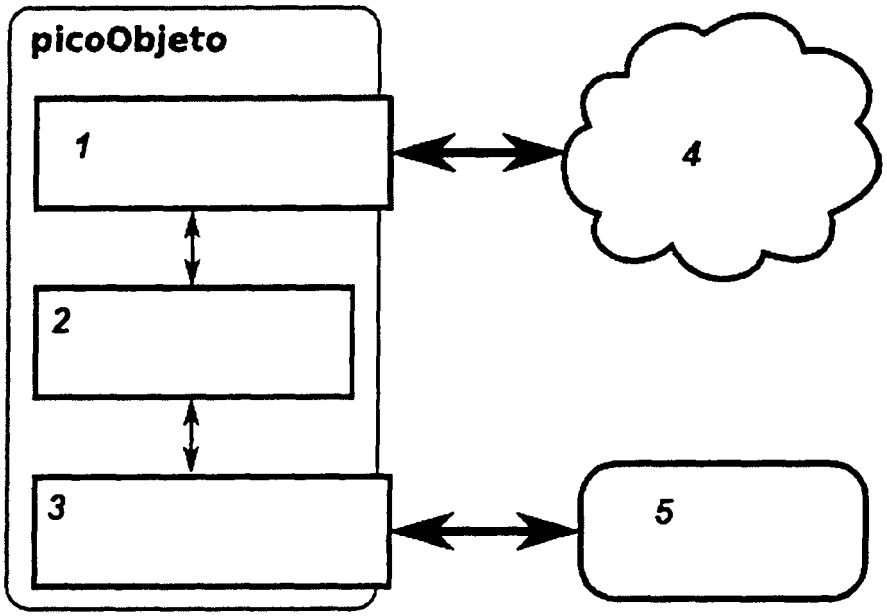

Figura 1

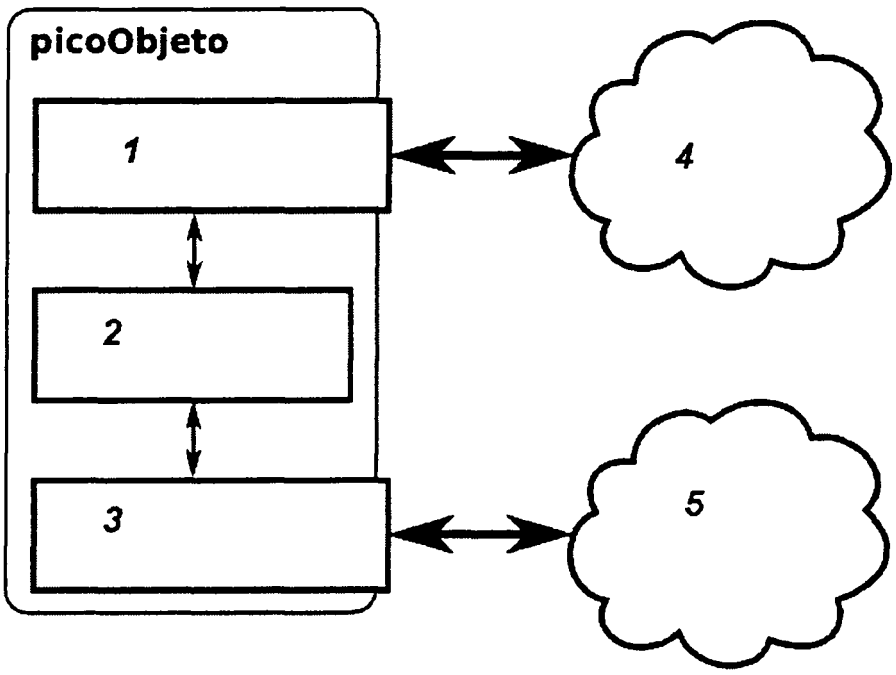

Figura 2

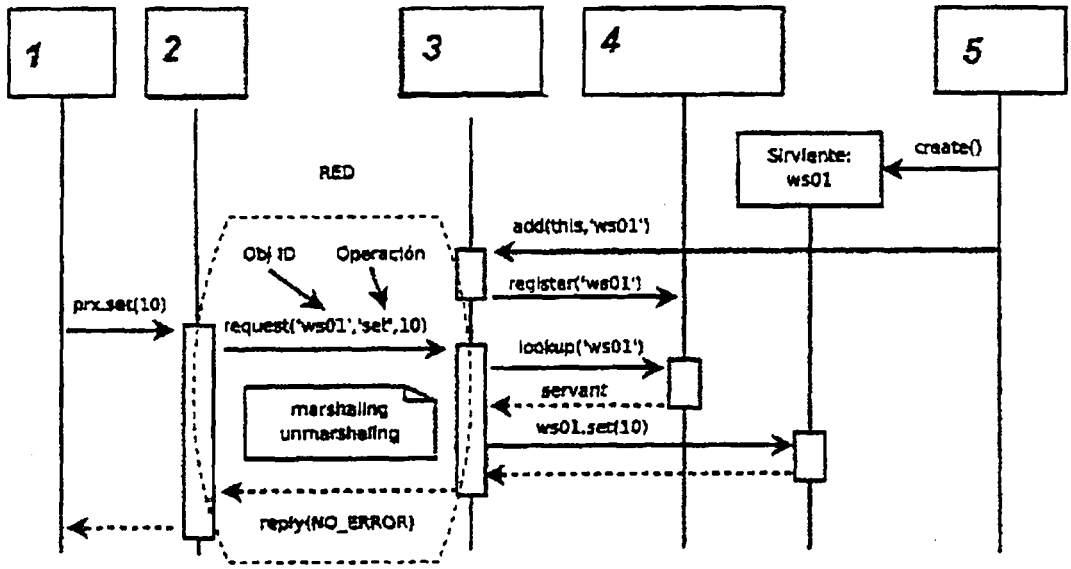
1
2
3
4
5
RED
Sirviente: ws01
create()
add(this,'ws01')
Obj ID
Operación
register('ws01')
prx.set(10)
request('ws01','set',10)
lookup('ws01')
servant
marshaling
unmarshaling
ws01.set(10)
reply(NO_ERROR)

Figura 3

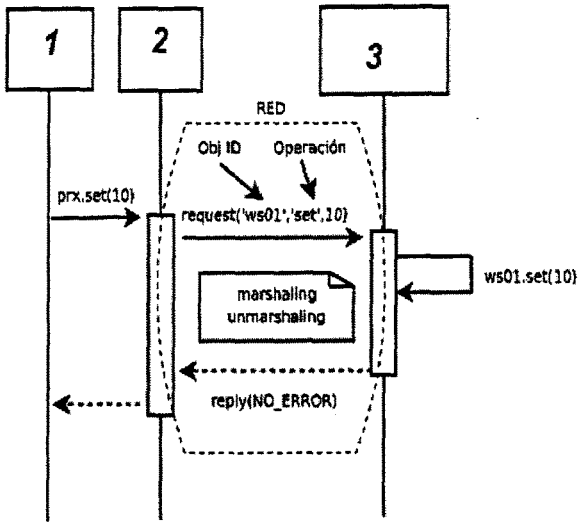

Figura 4

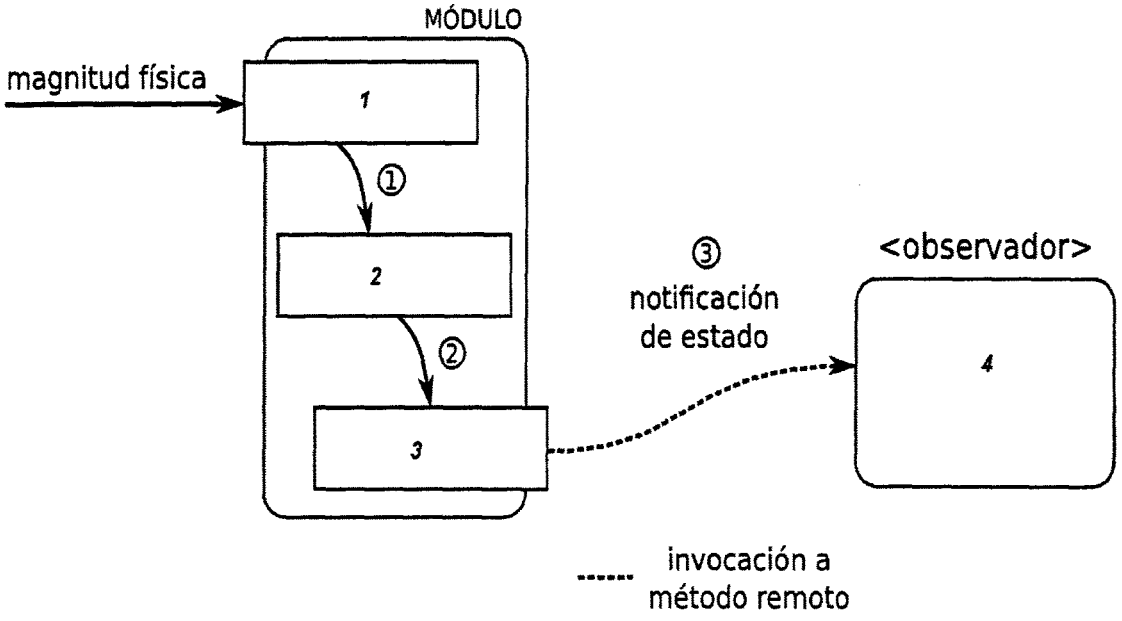

Figura 5

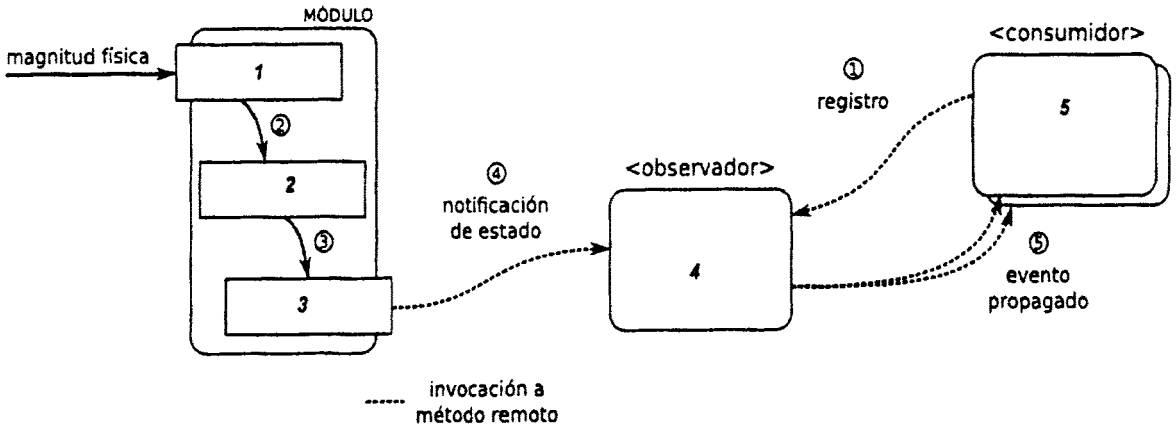
MÓDULO
magnitud física
1
2
3
④
notificación
de estado
<observador>
4
①
registro
<consumidor>
5
⑤
evento
propagado
invocación a
método remoto

Figura 6

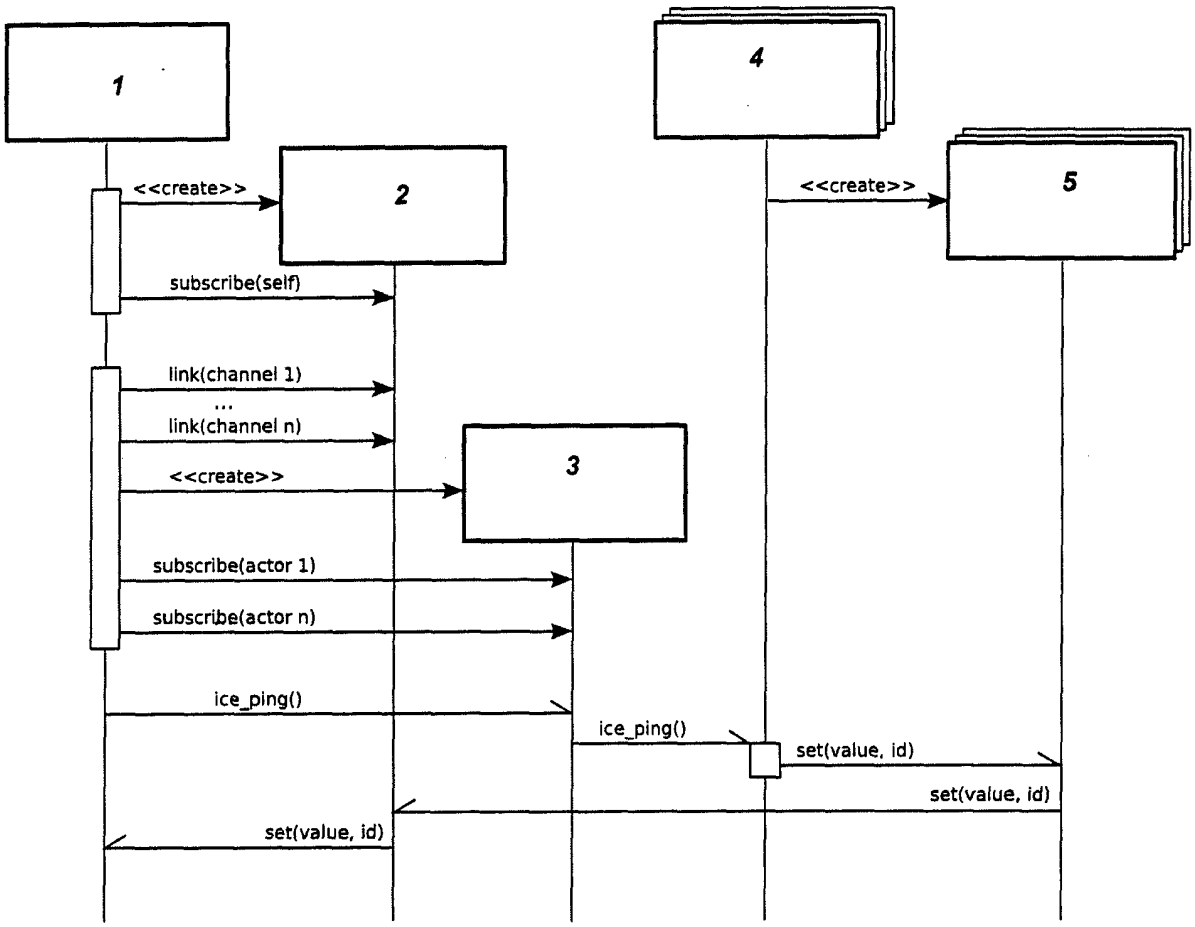
1
2
3
4
5
<<create>>
<<create>>
<<create>>
subscribe(self)
link(channel 1)
...
link(channel n)
subscribe(actor 1)
subscribe(actor n)
ice_ping()
ice_ping()
set(value, id)
set(value, id)
set(value, id)

Figura 7

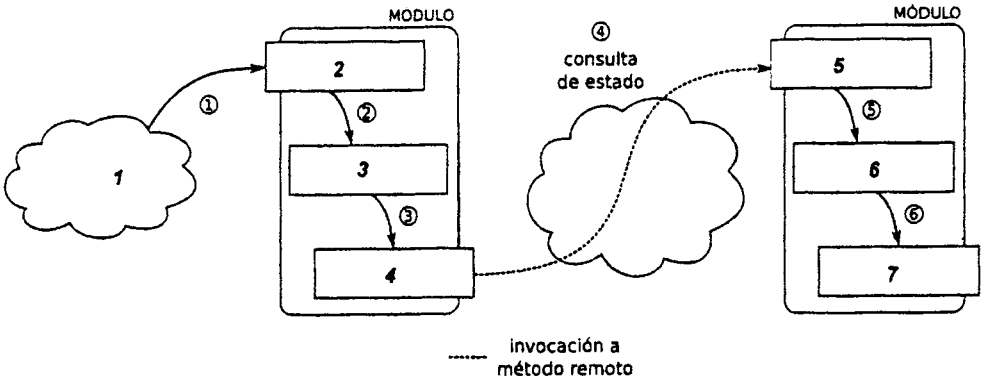
MODULO
MÓDULO
1
2
3
4
5
6
7
①
②
③
④
consulta
de estado
⑤
⑥
invocación a
método remoto

Figura 8

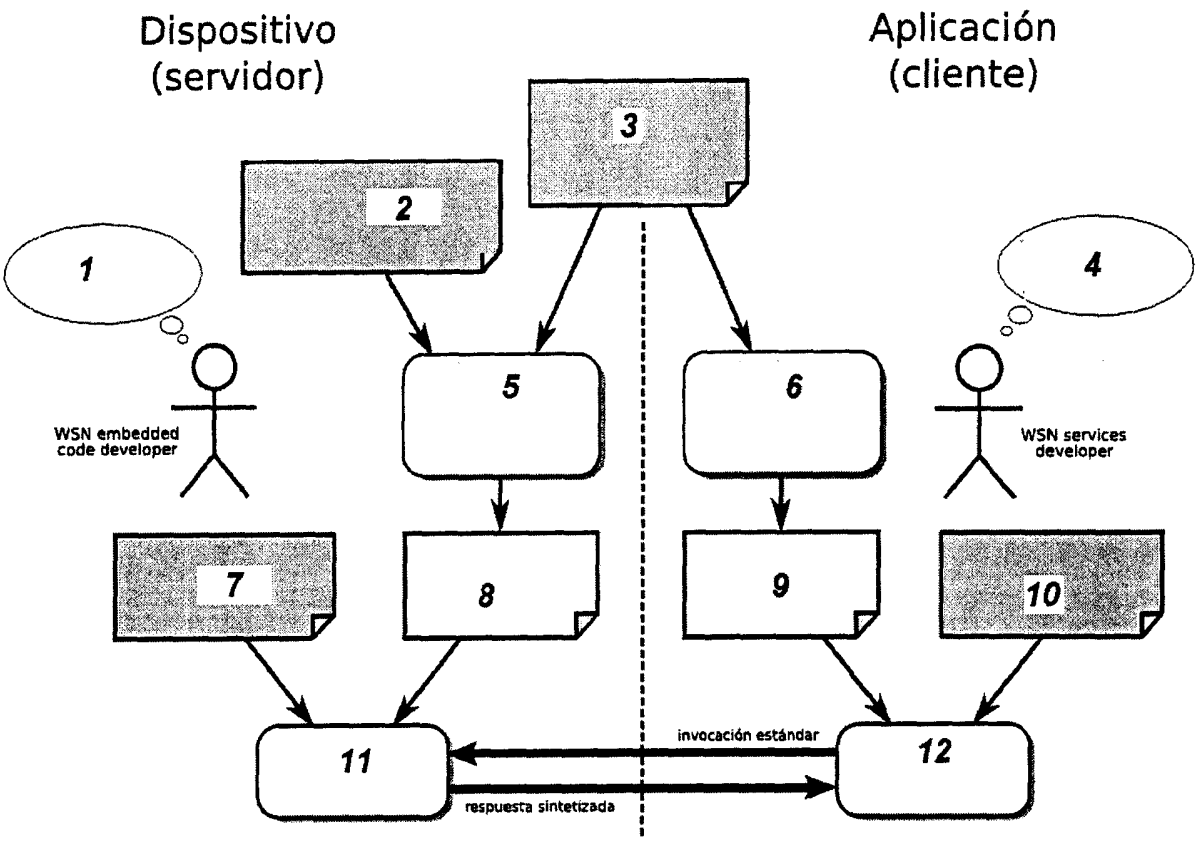
Dispositivo
(servidor)
Aplicación
(cliente)
1
2
3
4
5
6
7
8
9
10
11
12
WSN embedded
code developer
WSN services
developer
invocación estándar
respuesta sintetizada

Figura 9

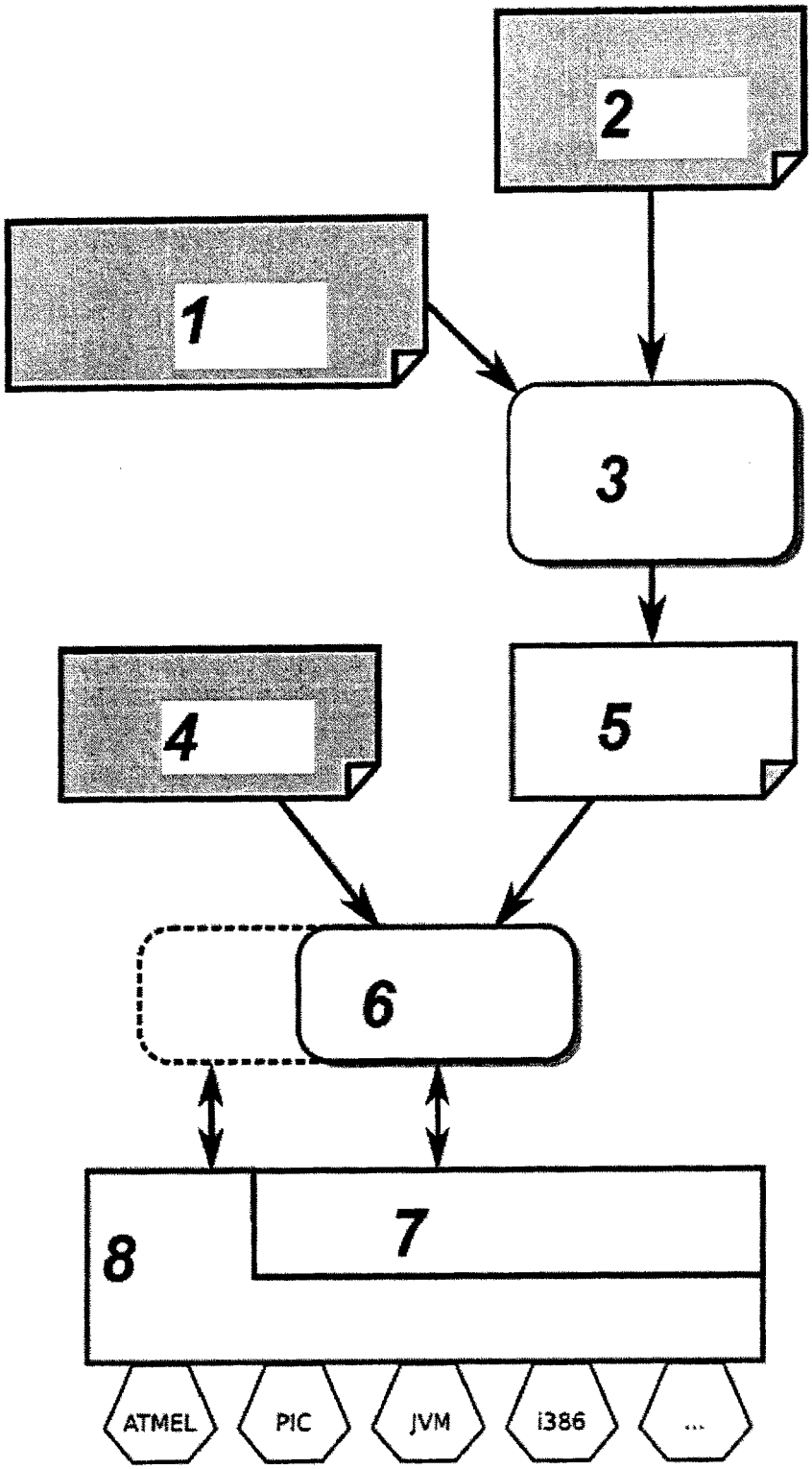

Figura 10

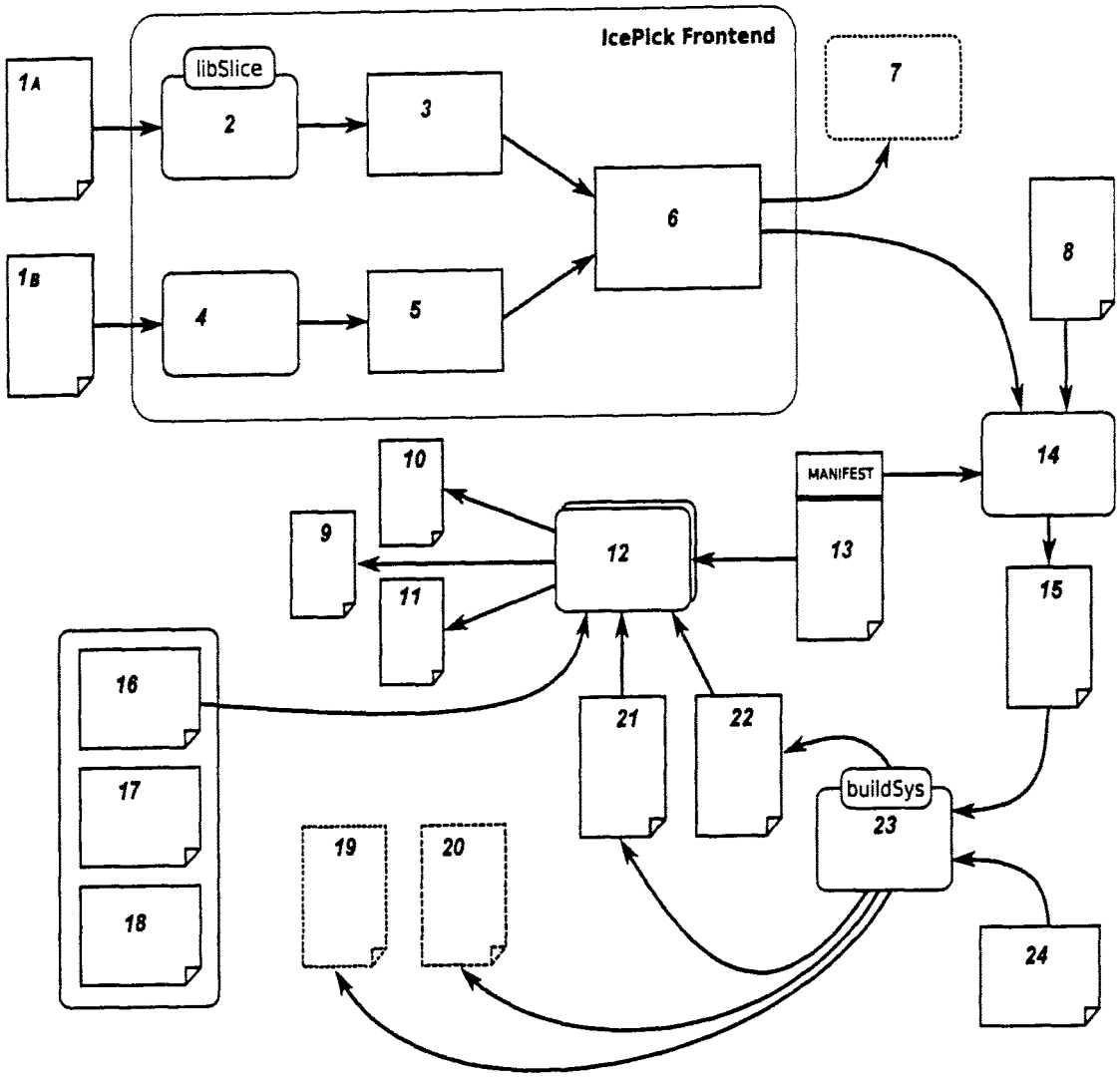
IcePick Frontend
libSlice
1A
1B
2
3
4
5
6
7
8
9
10
11
12
13
MANIFEST
14
15
16
17
18
19
20
21
22
buildSys
23
24

Figura 11

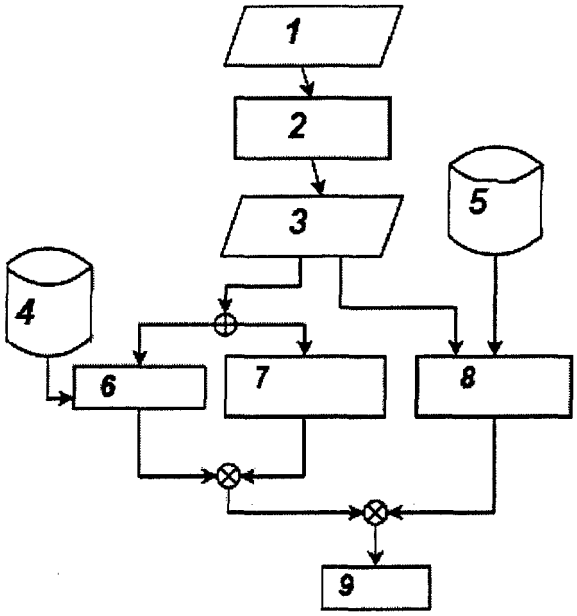

Figura 12

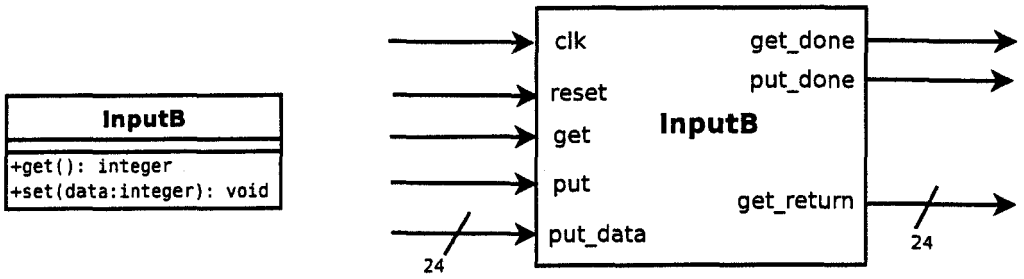

Figura 13

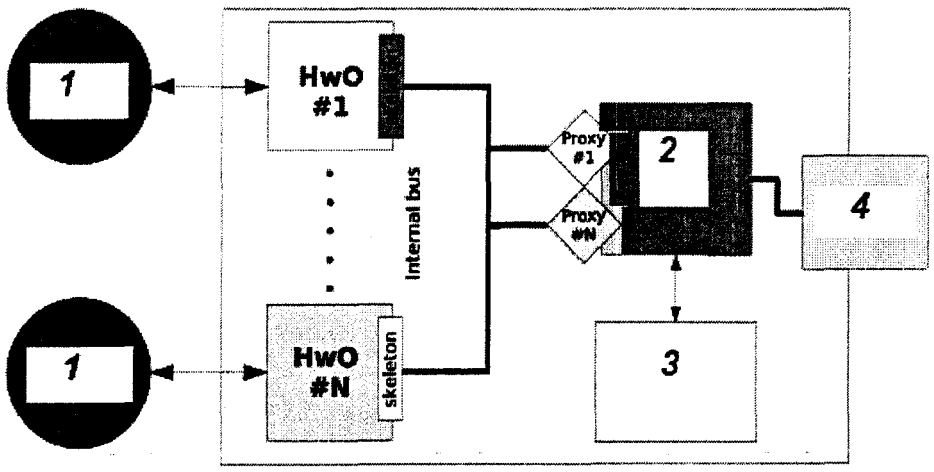

Figura 14

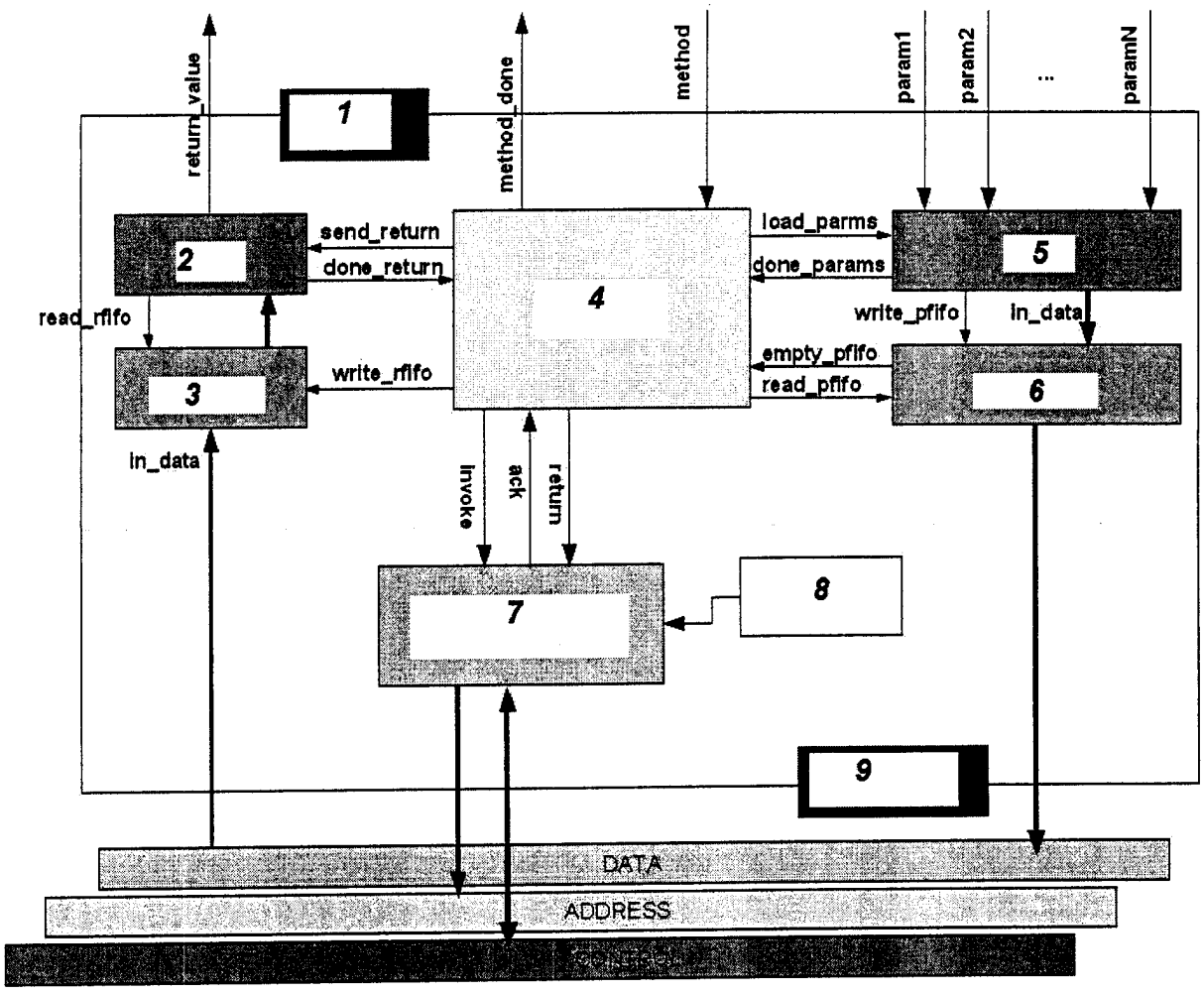
return_value
method_done
method
param1
param2
...
paramN
1
2
send_return
done_return
4
load_parms
done_params
5
read_rflfo
write_pflfo
in_data
3
write_rflfo
empty_pflfo
read_pflfo
6
in_data
invoke
ack
return
7
8
9
DATA
ADDRESS

Figura 15

OFICINA ESPAÑOLA
DE PATENTES Y MARCAS

ESPAÑA

(21) N.º solicitud: 200803715

(22) Fecha de presentación de la solicitud: 26.12.2008

(32) Fecha de prioridad:

INFORME SOBRE EL ESTADO DE LA TECNICA

(51) **Int. Cl. :** ***G06F9/54*** (2006.01)
H04Q9/00 (2006.01)

DOCUMENTOS RELEVANTES

Categoría	(56) Documentos citados	Reivindicaciones afectadas
X	F. MOYA, D. VILLA, F. J. VILLANUEVA, J. BARBA, F. RINCÓN, J. C. LÓPEZ "Embedding standard distributed object-oriented middlewares in wireless sensor networks" 28.08.2007. Documento recuperado de Internet. Dirección web: https://arco.esi.uclm.es/public/papers/2009-WCMC.pdf	1-11
A	VILLANUEVA F J; VILLA D; MOYA F; BARBA J; RINCON F; LOPEZ J C. "Leightweight Middleware for Seamless HW-SW Interoperability, with Application to Wireless Sensor Networks" Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07. 01.04.2007, Págs: 1-6, ISBN 978-3-9810801-2-4; ISBN 3-9810801-2-2, doi:10.1109/DATE.2007.364431.	1,5
A	DRUMEA A; POPESCU C "Finite state machines and their applications in software for industrial control" Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004. 27th International Spring Seminar on Bankya, Bulgaria 13.05.2004. Vol. 1, Págs. 25-29. ISBN 978-0-7803-8422-4; ISBN 0-7803-8422-9.	1

Categoría de los documentos citados
X: de particular relevancia
Y: de particular relevancia combinado con otro/s de la misma categoría
A: refleja el estado de la técnica
O: referido a divulgación no escrita
P: publicado entre la fecha de prioridad y la de presentación de la solicitud
E: documento anterior, pero publicado después de la fecha de presentación de la solicitud

El presente informe ha sido realizado

☒ para todas las reivindicaciones ☐ para las reivindicaciones nº:

Fecha de realización del informe	Examinador	Página
02.02.2012	M. Muñoz Sanchez	1/4

Documentación mínima buscada (sistema de clasificación seguido de los símbolos de clasificación)

G06F, H04Q

Bases de datos electrónicas consultadas durante la búsqueda (nombre de la base de datos y, si es posible, términos de búsqueda utilizados)

INVENES, EPODOC, WPI , XPMISC, XPI3E, XPIETF, XPIEE, XPESP, NPL, COMPDX

Fecha de Realización de la Opinión Escrita: 02.02.2012

Declaración

Novedad (Art. 6.1 LP 11/1986)	Reivindicaciones	5, 8-11	**SI**
	Reivindicaciones	1-4,6-7	**NO**
Actividad inventiva (Art. 8.1 LP11/1986)	Reivindicaciones		**SI**
	Reivindicaciones	5, 8-11	**NO**

Se considera que la solicitud cumple con el requisito de aplicación industrial. Este requisito fue evaluado durante la fase de examen formal y técnico de la solicitud (Artículo 31.2 Ley 11/1986).

Base de la Opinión.-

La presente opinión se ha realizado sobre la base de la solicitud de patente tal y como se publica.

1. Documentos considerados.-

A continuación se relacionan los documentos pertenecientes al estado de la técnica tomados en consideración para la realización de esta opinión.

Documento	Número Publicación o Identificación	Fecha Publicación
D01	F. MOYA, D. VILLA, F. J. VILLANUEVA, J. BARBA, F. RINCÓN, J. C. LÓPEZ "Embedding standard distributed object-oriented middlewares in wireless sensor networks" 28.08.2007. Documento recuperado de Internet. Dirección web: https://arco.esi.uclm.es/public/papers/2009-WCMC.pdf	
D02	VILLANUEVA F J; VILLA D; MOYA F; BARBA J; RINCON F; LOPE J C. "Leightweight Middleware for Seamless HW-SW Interoperability, with Application to Wireless Sensor Networks" Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07. 01.04.2007, Págs: 1-6, ISBN 978-3-9810801-2-4; ISBN 3-9810801-2-2, doi:10.1109/DATE.2007.364431.	
D03	DRUMEA A; POPESCU C "Finite state machines and their applications in software for industrial control" Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004. 27th International Spring Seminar on Bankya, Bulgaria 13.05.2004. Vol. 1, Págs. 25-29. ISBN 978-0-7803-8422-4; ISBN 0-7803-8422-9.	

2. Declaración motivada según los artículos 29.6 y 29.7 del Reglamento de ejecución de la Ley 11/1986, de 20 de marzo, de Patentes sobre la novedad y la actividad inventiva; citas y explicaciones en apoyo de esta declaración

Se considera D01 el documento más próximo del estado de la técnica al objeto de la solicitud.

Reivindicaciones independientes

Reivindicación 1: El documento D01 divulga un módulo de interconexión de objetos distribuidos accesible mediante un middleware de comunicaciones que implementa una máquina de estados finitos para reconocer mensajes de petición de clientes y generar respuestas con un microcontrolador (FPGA o similar) para procesar los mensajes. También cuenta con un dispositivo de comunicaciones y un transductor.

Por tanto el documento D01 afecta a la novedad de la reivindicación 1 según el artículo 6.1 de la Ley de Patentes.

Reivindicaciones dependientes

Reivindicaciones 2, 3, 7: el sensor, actuador, y el canal de eventos aparecen en D01.

Reivindicación 4: un punto de acceso remoto y otro local son necesarios para el funcionamiento del módulo y se consideran implícitos en D01.

Reivindicación 6: el contenido de esta reivindicación es un detalle explicativo del funcionamiento del módulo implícito en el funcionamiento de un módulo según la reivindicación 1.

Por tanto el documento D01 afecta a la novedad de las reivindicaciones 2-4, 6-7 según el artículo 6.1 de la Ley de Patentes.

Reivindicación 5: el intercambio asíncrono de mensajes es comúnmente conocido en el campo técnico de las comunicaciones digitales y por tanto evidente para el experto en la materia. Por otra parte aparece en el documento D02.

Reivindicación 8: los tipos de tráfico, unicast, multicast y broadcast son comúnmente conocidos en el campo técnico de las comunicaciones digitales y por tanto evidentes para el experto en la materia.

Reivindicaciones 9, 10: la interconexión de redes utilizando elementos habituales (puente) y sin un protocolo específico de comunicaciones que ejecute el dispositivo de comunicaciones son comúnmente conocidos en el campo técnico de las comunicaciones digitales y por tanto evidentes para el experto en la materia.

Reivindicación 11: es una particularización del módulo reivindicado para una aplicación concreta pero sin que incluya otras características técnicas distintas a las ya comentadas y específicas para la aplicación en cuestión. Por tanto también resulta evidente para el experto en la materia.

Por tanto el documento D01 afecta a la actividad inventiva de las reivindicaciones 5, 8, 9-11 según el artículo 8.1 de la Ley de Patentes.