Mobiware

Middleware Móvil y Fiable para el Internet del Futuro

http://mobiware.iti.es

María Blasco Roca
03/09/2011

Resumen

El objetivo del proyecto es la realización de un sistema software que facilite la implementación de aplicaciones sensibles al contexto, proporcionando una solución simplificada ante problemas comunes derivados de la comunicación entre dispositivos heterogéneos. El middleware que el presente documento describe pertenece al ámbito de la computación ubicua ya que, en base a ciertos sensores integrados en dispositivos móviles, es capaz de responder ante las necesidades de los individuos/usuarios involucrados.

Durante todo el proceso de diseño y desarrollo, se ha prestado especial atención en cuestiones críticas como la movilidad y heterogeneidad de las fuentes, las relaciones y dependencias de la información a tratar, la imprecisión de los datos recogidos y la usabilidad y eficiencia.

Palabras clave: Internet of things, context-aware, publish/subscribe, context modeling.

Contenido

Introducción

La evolución de las tecnologías de la información ha desempeñado un papel fundamental en la mejora de la calidad de vida de las personas y, por tanto, en su superación y desarrollo. Mucho ha cambiado desde aquéllos años en los que los usuarios requerían de ciertos conocimientos técnicos para poder hacer frente a tediosas tareas, que a su vez, eran ejecutadas sobre computadores cuya autonomía y movilidad eran prácticamente nula. Con la aparición de los ordenadores portátiles se consiguió mejorar este último aspecto, pero no fue hasta la llegada de las primeras ideas sobre computación ubicua (años 80) cuando realmente se dio el salto con respecto a la autonomía e interacción de los usuarios con éstos. Una reflexión interesante es la que realiza Mark Weiser en [WGB], en la que plantea la siguiente cuestión: "… existe más información a nuestro alcance durante un paseo por el bosque que en cualquier sistema informático, sin embargo, la gente encuentra un paseo entre los árboles placentero y las computadoras frustrantes". Como solución a esta frustración, propone que los computadores se ajusten o adapten al entorno humano, en lugar de que los seres humanos sean los que se tengan que ajustar a ellos.

A lo largo de estos últimos años, esta corriente se ha ido afianzando, llevándose a cabo gran cantidad de estudios y avances sobre el tema, por lo que podemos afirmar que la tendencia evoluciona hacia la creación de sistemas móviles y con un alto grado de transparencia para los usuarios. Como consecuencia, han surgido diversos términos que describen estas dos dimensiones (movilidad y transparencia) y que definen a este tipo de sistemas como sensibles al contexto. Aunque cada uno de ellos se centra en matices diferentes (tiempo de respuesta, capacidad de reacción, capacidad de auto-configuración, etc.) como idea común a todos subyace la satisfacción de las necesidades de los usuarios de una manera **autónoma**, **omnipresente** e **imperceptible** para ellos.

Los sistemas sensibles al contexto permiten adaptar sus operaciones (comportamiento) en función del estado actual (contexto) sin que sea necesaria la intervención de los usuarios. Un contexto puede ser físico (si la información se obtiene a partir de elementos hardware incrustados en el ambiente: temperatura, luminosidad, etc.) o lógico (si es el usuario el que especifica la información, ya sea a través de sus preferencias, o como resultado de su interacción: estados emocionales, objetivos, etc.). Pero, ¿qué información se considera relevante en situaciones cambiantes? A partir de ahora, se va a considerar de interés toda aquella información que permita obtener (calcular) el estado actual de las entidades del entorno involucradas en la interacción persona-aplicación.

Debido a que este tipo de sistema permite capturar y procesar contextos procedentes de diversas fuentes heterogéneas, resulta necesaria la creación de un middleware o mecanismo intermedio que sea capaz de gestionar, de manera eficiente, cuestiones relacionadas con la movilidad, heterogeneidad e imprecisión de la información entre dispositivos y aplicaciones. Con esta idea de fondo, se ha desarrollado el proyecto Mobiware, cuyo resultado es la creación de una solución simplificada ante problemas comunes derivados de la comunicación entre dispositivos heterogéneos.

El presente documento se divide en diversos capítulos o secciones que detallan el resultado del trabajo realizado. Tras presentar los objetivos y describir el entorno general del proyecto, se presenta un resumen o estado del arte acerca de la investigación realizada sobre los sistemas sensibles al contexto (su impacto, arquitecturas empleadas, resultados obtenidos, etc.). Más adelante, evolucionando hacia una parte más técnica del documento, se especifica la tecnología que se ha empleado para el desarrollo del software, así como, una vez realizada la investigación, el diseño de la arquitectura que se ha optado por seguir. A continuación, se presenta el resultado final obtenido, diversas aplicaciones que ponen en práctica este resultado y las conclusiones, tanto a un nivel técnico como a otro más personal, sobre todo el trabajo realizado. Para finalizar, se expone cuál es la siguiente línea de actuación y posibles extensiones futuras del proyecto.

A lo largo de todo el proyecto, he participado tanto en las fases de diseño y especificación

de requisitos, como en la fase de implementación y pruebas. Aunque se han implementado diversas aplicaciones cliente, mi trabajo se ha centrado en la parte de desarrollo del middleware, trabajando activamente en los diversos módulos que lo componen: autenticación de los usuarios en el sistema, envío y gestión de sensorización, inferencia y creación de nuevo conocimiento y gestión de noticias y contenidos.

Objetivos

En el libro *When Things Start to Think* (1999), Neil Gershenfeld del MIT (Massachusetts Institute of Technology), se sugiere la necesidad de "intentar que los ordenadores estén en todas partes y, además, intentar que éstos no estorben", es decir, hacer uso de la tecnología sin prestarle más atención que la que tarea que se quiere llevar a cabo se merece. Pero… ¿cómo hacer que estén en todas partes?

El Internet de las cosas o Internet del Futuro es un concepto emergente que está teniendo una gran proyección en los últimos años y que, de una forma u otra, ya está cambiando el modo de interacción tradicional entre persona-ordenador (*human-computer-interaction*). Así pues, las innovaciones derivadas de esta propuesta hacen hincapié, entre otras cosas, en la autonomía y transparencia de los servicios prestados (adaptación en función de las necesidades del usuario). El principio del *Internet of Things* se fundamenta en la inclusión de objetos físicos (es decir, objetos de la vida cotidiana, como pueden ser lavadoras, neveras, frigoríficos, etc.) en Internet, utilizando para ello mecanismos de identificación y monitorización.

La heterogeneidad de estos objetos, así como su muy diversa naturaleza hace necesaria la creación de soluciones que faciliten la implementación de aplicaciones que hagan uso de estos recursos, de la manera más simple posible. Con todo ello surgió la idea de Mobiware, cuyo principal objetivo es la investigación y posterior creación de un *middleware* o arquitectura de soporte para la implementación de aplicaciones adaptativas a las necesidades del usuario, en función de la información recogida del entorno.

Debido a la gran diversidad de información a tratar en el desarrollo de este software, se ha realizado un estudio y se ha puesto especial hincapié a los siguientes campos:

- Dispositivos Móviles. Creación de un mecanismo de gestión de contenidos y contextos para entornos de dispositivos heterogéneos y móviles, teniendo en cuenta cuestiones de eficiencia y robustez seguridad, imprecisión de los datos recogidos y cuestiones de movilidad y heterogeneidad de las fuentes, así como las relaciones y dependencias de la información a tratar.

- Sensorización/Monitorización. Definición de un algoritmo y una arquitectura para la configuración de los sensores. Gestión de la información histórica almacenada y necesaria para el correcto funcionamiento del sistema y la gestión de contextos (razonamiento lógico realizado por el motor de inferencia).

- Interoperabilidad. Definición de algoritmos, protocolos y una arquitectura que permita la interconexión y colaboración de distintos sistemas y servicios, teniendo en cuenta la usabilidad y eficiencia de la aplicación).

Descripción del entorno general del proyecto

Mobiware es el resultado de un proyecto de investigación dentro del programa IMPIVA de I+D para institutos tecnológicos. Su desarrollo se ha llevado a cabo íntegramente en el Área de Internet y Computación Ubicua (AICU) del Instituto Tecnológico de Informática (ITI) de Valencia.

La solución propuesta implementa un middleware, o capa de abstracción, cuyo objetivo es facilitar el desarrollo de aplicaciones sensibles al contexto; proporcionando una solución simplificada ante problemas comunes derivados de la comunicación entre dispositivos heterogéneos. Como información del entorno, el sistema utiliza los datos procedentes de sensores localizados en dispositivos móviles (como por ejemplo, teléfonos móviles sobre plataformas *android* o *iphone*). En una segunda fase, se ampliará este dominio de ejecución con el objetivo de adaptar el middleware al manejo de información procedente de redes de sensores, compuestas por diversos nodos distribuidos y localizadas en espacios cerrados (hospitales, fábricas, etc.) o abiertos (campos, bosque, etc.).

Para gestionar eficientemente las actualizaciones de la información recibida en un contexto dinámico, ha sido necesaria tomar decisiones de diseño e implementación con respecto a la información recogida por los sensores, al repositorio de datos recibidos y a la difusión del nuevo conocimiento generado. Estas decisiones se van explicando en los diversos capítulos que integran el documento.

Descripción del proyecto

En esta sección, se va a describir la investigación llevada a cabo sobre el modelado del contexto y los sistemas sensibles al él. El estudio de diversos modelos, para la gestión del contexto y sobre diversas aplicaciones de naturaleza similar, han sido dos factores clave a la hora de tomar decisiones en el diseño e implementación de Mobiware.

Aplicaciones sensibles al contexto: Estado del arte

"There is more information available at our fingertips during a walk in the woods than in any computer system, yet people find a walk among trees relaxing and computers frustrating. Machines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods".

Mark Weiser. The Computer for the 21st Century. Scientific American, Sept. 1991, 94-104.

Han pasado dos décadas desde las primeras investigaciones en los laboratorios PARC (Palo Alto Research Center) de Xeox, USA, en las que se dio a conocer el concepto de computación ubicua (ubicomp). No obstante, éste término sigue siendo un tema candente y de gran interés en la actualidad, gracias en gran medida a los avances, especialmente a nivel del hardware, que se han producido durante los últimos años. Nuevas investigaciones han originado nuevos términos [Lok07], aunque todos ellos, alrededor del mismo concepto base: La satisfacción de las necesidades de los usuarios de una manera *autónoma, omnipresente e imperceptible* para ellos. Algunos de estos términos son:

- Computación ubicua (Weiser, 1991): Uso invisible de ordenadores en el entorno físico de los usuarios.
- Computación omnipresente (IBM Charman Lou): Combinación de la computación móvil y la procedente de dispositivos integrados en el entorno.
- Computación invisible (Norman, 1998. Borniello, 2000): Se centra en la capacidad de los usuarios para realizar una tarea sin que sea necesario prestarle más atención que la propia tarea en sí merece.
- Computación proactiva (Tennenhouse, 2000) y computación autónoma (Horn, 2001): Utilización de la información contextual del mundo físico (sensores) y la procedente de la interacción con los usuarios para intentar adelantarse a sus decisiones y poder tomar decisiones en su nombre. Además, en la computación autónoma los sistemas deben ser capaces de auto-configurarse y auto-repararse (respuesta ante fallos).
- Inteligencia ambiental (Marzano y Aarts, 2003): Utilización de dispositivos inteligentes integrados y omnipresentes en la vida cotidiana, capaces de responder a las necesidades de los usuarios/individuos.
- Computación sensible (Hooper, 2007): Utilización de sensores para mantener un modelo del mundo que es compartido tanto por las aplicaciones, como por los usuarios.

El creciente impacto y auge en la evolución de la computación móvil, ha hecho que la tendencia en los últimos años se dirija hacia el estudio y desarrollo de las llamadas

aplicaciones sensibles al contexto (context-aware). Tal y como pronosticaba Weiser en su analogía entre un paseo por el bosque y el uso de un ordenador (la frustración que puede llegar a sufrir un individuo), las actuales investigaciones avanzan hacia la búsqueda de la integración de las tecnologías de la información en la vida diaria, de tal forma que una computadora se perciba como un elemento del ambiente, pudiendo mejorar con ello las condiciones y la calidad de vida de los individuos.

Un sistema sensible al contexto puede disponer y hacer uso de una gran cantidad de dispositivos heterogéneos, que interactúan entre sí intercambiando datos sobre su estado y sobre su percepción del entorno. Esto hace que las aplicaciones desarrolladas sobre estos sistemas deban adaptarse a las diferentes situaciones que puedan darse, de manera autónoma y transparente para los usuarios. Así pues, es posible tener desde fuera la percepción de que las aplicaciones se adaptan y son capaces de responder a los requisitos (necesidades) de los usuario o a las situaciones que se van generando a lo largo del tiempo. Por ejemplo, en un sistema domótico, el propio sistema podría determinar (de manera autónoma) que se encuentra en un estado “noche” y, por tanto, podría encender la calefacción si detecta que la luz solar se ha reducido y la temperatura ha bajado.

Así pues, se considera que la sensibilidad al contexto es la habilidad para capturar y procesar contextos, siendo éste cualquier información que puede ser obtenida de diferentes fuentes heterogéneas [SB08]. En 2000, Dey y Abowd definieron un contexto como cualquier información (temporal, de localización, computacional, de actividades/emociones del usuario...) que puede utilizarse para caracterizar la situación de una entidad (ya sea una persona, objeto o lugar relevante para la interacción persona-aplicación). Así pues, existen dos definiciones para este mismo término ampliamente aceptadas por la comunidad:

- Contexto es cualquier situación que caracteriza el estado de una entidad (ya sea un lugar, persona u objeto) relevante para la integración del usuario y la aplicación, incluyendo a estos [ATH07].

- Contexto es el entorno y su percepción por parte de un dispositivo, que es capturado por los sensores del dispositivo o de la infraestructura. Un dispositivo sensible al contexto puede ser categorizado como directo si el contexto es capturado por los sensores del mismo o indirecto en el cual el contexto es adquirido por la infraestructura [GSB02].

Debido a la complejidad que supone captar, entender y determinar el contexto proporcionado por todos los entes de un sistema, se muestra imprescindible realizar una abstracción del mismo para que éste pueda ser procesado y tratado por la aplicación a desarrollar. A esta tarea se la conoce como modelado de contexto y en ella se engloban (entre otras) las siguientes tareas [ATH07]:

- Identificación de la información contextual más apropiada que sirvan para modelar adecuadamente el dominio concreto de aplicación.

- Identificación y modelado de las relaciones entre las distintas partes de la información contextual.

- Identificación de los posibles cambios dinámicos en la información contextual y modelar las posibles reacciones ante estos cambios.

La implementación de aplicaciones sensibles al contexto varía en función de la infraestructura disponible y la capacidad de procesamiento de la misma, el ámbito o dominio de aplicación, el paradigma de programación utilizado y requisitos más específicos como los mecanismos o patrones propuestos para su implementación. Considerando todos estos factores, de las aproximaciones desarrolladas en los últimos años han surgido diversos

enfoques de gestión del contexto, que pueden clasificarse en base a [KPT+09]:

1. Utilización de aproximaciones basadas en modelos de contexto.

2. Implementaciones basadas en extensiones del lenguaje de programación a nivel de código fuente.

3. Interceptación de mensajes.

Los conceptos que se muestran en esta categorización no son disjuntos, es decir, el desarrollador puede optar por una combinación de varias de estas técnicas a la hora de gestionar el contexto. No obstante, como idea común a todos ellos subyace la necesidad de la adaptación de la lógica de negocio, desacoplando la lógica de los servicios de la capa de gestión del contexto. De este modo, dejando a un lado los mecanismos de soporte a la arquitectura del sistema (enfoques 1, 2 y 3), la siguiente tabla destaca las principales características de las aproximaciones (enfoques 4, 5 y 6) que facilitan el diseño y desarrollo centrado en la adaptación de la lógica de negocio, centrándose en los siguientes criterios:

- Flexibilidad: grado de libertad proporcionado por el mecanismo de adaptación del contexto.

- Habilidad de refactorización: facilidad para rediseñar el servicio.

- Facilidad de uso: facilidad de uso para un desarrollador no del todo diestro en mecanismos de adaptación del contexto

.

- Adaptación a los sistemas existentes: grado de dificultad a la hora de aplicar el mecanismo de adaptación del contexto a un sistema ya existente.

- Tipo de modelo de contexto: existencia de un metamodelo de contexto específico.

- Desacoplamiento con la lógica de negocio: grado de independencia entre la información de contexto y la lógica de negocio.

	Flexibility	Refactoring ability	Ease of use	Existing system adaptation	Context Model Type	Decoupling from business logic
Source code level handling	High	Limited	Medium	Depends on existing system architecture	OOP-based (including XML-based)	Depends on programming language
Model-based approaches	Medium	High (assuming code re-generation)	Medium	Low (not possible if the system has not been modeled)	UML/MOF Models (mainly)	High
Message interception	Medium	High	High	Medium	External or embedded in messages	High

Tabla 1. Evaluación de enfoques para la gestión del contexto [KPT+09].

Las aproximaciones que utilizan un enfoque dirigido por modelos parecen más propicias para la ingeniería de servicios, ya que aportan los beneficios del paradigma Model Driven Engineering (MDE). Además, en los últimos años las técnicas de MDE han sido aplicadas satisfactoriamente por desarrolladores, en cuanto a la gestión del contexto se refiere.

Para finalizar, no hay que olvidar que cuestiones como la privacidad y la seguridad, son de vital importancia en los sistemas sensibles al contexto. Desafortunadamente, la privacidad de la información de contexto no se soporta correctamente por la mayoría de los enfoques, aunque es posible utilizar técnicas de control de acceso, Role Based Access Control (RBAC) con tal de subsanar tales deficiencias.

Modelos de Contexto

La representación formal de la información recibida del entorno en un modelo resulta esencial para poder realizar un razonamiento lógico y convertir, estos datos, en conocimiento relacionado con un determinado contexto. Las aproximaciones más sencillas y surgidas en los primeros años (modelos atributo-valor y modelos basados en lenguajes de marcado), han evolucionado hasta aproximaciones más complejas y con mayor capacidad expresiva (modelos basados en ontologías y modelos basados en rol-objeto).

El contexto se crea a partir del procesamiento de datos que proceden de fuentes de naturaleza muy diversa. Además, la información de contexto es ambigua, en el sentido de que es dependiente del dominio en el que éste se aplique y de la capacidad expresiva que se desee reflejar. Así pues, a la hora de definir un modelo de contexto no sólo debe tenerse en cuenta la capacidad computacional del sistema, sino también su capacidad de razonamiento y gestión del comportamiento ante nuevos cambios.

Aunque no existe un modelo único que cumple todos estos requisitos, si existen diversos modelos que proporcionan soluciones adecuadas según los conceptos que se desean gestionar y las relaciones entre estos conceptos. La elección de un modelo u otro dependerá de cuál se adapta mejor a estos requisitos, de su complejidad y de la capacidad de procesamiento disponible. No obstante, algunos sistemas implementan soluciones ad-hoc para expresar la información y modelar los diferentes contextos que pueden darse.

Las principales características que definen a un modelo de contexto y su sistema de gestión, vienen dadas por el cumplimiento o soporte a los siguientes requisitos [BBH+10]:

- Movilidad y heterogeneidad: el modelo debe soportar la captación de diversas fuentes de información, en ocasiones procedentes de diferentes lugares (o dependen de su situación) debido a la movilidad de los sensores.
- Relaciones y dependencias: el modelo debe ser capaz de tratar las relaciones entre diferentes contextos, ya que determinadas situaciones pueden depender de informaciones de contexto procedentes de otras entidades.
- Historia: el modelo debe poder trabajar con estados anteriores o incluso derivar o anticipar estados futuros.
- Imperfección: el modelo debe gestionar correctamente la naturaleza implícita de la variabilidad en la calidad de la información durante el tiempo de vida en ejecución.
- Razonamiento: el modelo debe disponer de capacidad suficiente para derivar o extraer nuevos hechos a partir de los datos existentes (obtención de nuevo conocimiento).
- Usabilidad: el modelo debe ser lo más sencillo posible y fácil de utilizar, además debe poder portarse de una forma sencilla el mundo real a modelar hacia el propio modelo.
- Eficiencia: el modelo debe reflejar correctamente las características de escalabilidad y procesado de forma eficiente incluso en presencia de múltiples datos y entidades.

Si el modelo de contexto está bien definido, se reduce la complejidad de las aplicaciones involucradas, mejorando su mantenimiento y capacidad de cambio. Además, al igual que ocurre en el desarrollo software, un diseño adecuado facilita su implementación e influye positivamente en el despliegue de futuras aplicaciones.

A continuación, se describen las principales propuestas y técnicas utilizadas en el modelado de contextos [SB08]:

Modelos atributo-valor

Este tipo de estructura de modelado surge en las primeras aproximaciones de definición y diseño de contextos. Se trata de un modelo sencillo en el que la información que manejan las aplicaciones, y que describe un contexto, se expresa utilizando tuplas de tipo atributo-valor.

Schilit modela la información de contexto en tuplas de tipo atributo-valor, actuando como claves las posibles variables del entorno. Por ejemplo, un entorno de una habitación puede incluir como variable los ocupantes de la misma [STW93]:

```
OCCUPANTS=adams:schilit:theimer:weiser:welch
```

Esta variable en concreto, es mantenida y actualizada a partir de la información extraída de la localización de los usuarios concretos. La localización sería, en este caso, otra variable a modelar en el entorno.

Una solución propuesta que hace uso de este enfoque de pares clave-valor para representar la información contextual es Mobisaic Web Browser [VB94]. Se trata de una extensión para los navegadores web que utiliza información contextual dinámica para generar nuevos enlaces, denominados *dynamic URL".* Estos enlaces, contienen variables del entorno asociadas a un contexto, que cambia en función de la localización del usuario [CK00]. Un ejemplo de esta propuesta, podría ser el correo de Google, en el que en función de cierta información de localización recibida, es capaz de adaptarse y mostrar la página con la información meteorológica del lugar en el que el usuario se encuentra.

Entre las principales limitaciones de este tipo de modelos, destaca la falta de expresividad a la hora de definir relaciones y dependencias (reflejada en el carácter atómico del valor de los atributos), así como la falta de orden en la definición de las tuplas, ya cada relación se define como un conjunto (y en un conjunto, no hay relación de orden).

Modelos basados en lenguajes de marcado

Estos modelos definen la estructura y semántica de un contexto por medio de un conjunto de etiquetas, atributos y comentarios jerarquizados. Para ello, utilizan un lenguaje de marcas basado en eXtensible Markup Language (XML), y es el Standard Generic Markup Language (SGML) el de uso más extendido.

No obstante, para abordar el alto dinamismo de la información de contexto manejada en dispositivos móviles, se emplea Composite Capabilites/Preference Profile (CC/PP) desarrollado por el World Wide Web Consortium (W3C) y basado en Resource Description Framework (RDF). Los modelos que utilizan CC/PP se consideran una combinación de los basados en lenguajes de marcado y los basados en atributo-valor. Esto se debe a que la sintaxis RDF utiliza tuplas de tipo atributo-valor, que almacena a partir de etiquetas, para el modelado de contexto.

Una de las aplicaciones que utiliza un modelo basado en lenguajes de marcado (SGML) para definir el contexto, es Stick-e. El objetivo de este sistema es mostrar cierta información a los usuarios en función de la localización de los mismos. Para ello, utiliza un Stick-e Document que consta del propio contenido o mensaje, y del contexto relacionado. A grandes rasgos, sería como una forma electrónica de dejar post-its en determinados lugares de interés. Un ejemplo de Stick-e Document, podría ser el siguiente [Bro98]:

```
<note some attributes>
<required>
<with> J.D. Bovey <or> X. Chen
```

```
<at> coordinates of location
<content>
This is the content.
</note>
```

Como puede apreciarse, se utiliza el lenguaje SGML para definir el contexto. Así pues, el contexto se introduce a partir de la etiqueta `<require>` y utiliza otras etiquetas como `<at>` (localización) y `<with>` (compañía) para expresarlo.

Aunque este tipo de modelos es fácilmente exportable, su procesamiento y manejo resulta bastante complejo (ya que su estructura en forma de árbol establece un orden predeterminado).

Modelos gráficos

Esta aproximación representa las entidades y relaciones del modelo de contexto de forma gráfica. Los ejemplos más representativos de este tipo de modelos son Unified Modeling Language (UML), con respecto a los modelos orientados a objetos (ver apartado 2.4), y Object-Role Modeling (ORM), un estándar que se basa en hechos para modelar la información de un dominio específico según un enfoque basado en rol-objeto (ver apartado 2.5).

Modelos orientados a objetos

En los modelos orientados a objetos la información de contexto se modela como un estado de un objeto. Los objetos son entidades que se pueden distinguir de manera clara y definida, constando de propiedades, métodos y eventos. De este modo, es el propio objeto el que proporciona una interfaz para acceder y modificar su estado, pudiendo ocultar estos métodos de acceso a otros objetos definidos en el modelo. Este tipo de modelado se centra en la reutilización de los elementos del modelo y se caracteriza por su mecanismo de control de acceso a la información contextual. Así pues, aprovecha todas las ventajas que proporciona el paradigma de la orientación a objetos: herencia, encapsulación, reusabilidad y relaciones entre objeto.

UML es un estándar de facto ampliamente aceptado por la industria del software. Se trata de un lenguaje gráfico utilizado para especificar, construir y documentar los elementos que modelan un sistema. Además, al ser diseñado como un lenguaje de modelado de propósito general, puede ser utilizado en la mayoría de métodos orientados a objetos y componentes, puede ser aplicado en un amplio rango de dominios (financiero, industrial, telecomunicaciones…) y, además, puede ser implementado en plataforma muy diversas (JE22, .NET…) Es por estas razones por las que UML puede utilizarse para representar información relacionada con el diseño de sistemas sensibles al contexto [SB05]. Los aspectos estructurales del sistema se definen a partir de clases, que no son más que formalizaciones de objetos con servicios, propiedades y comportamiento comunes. A su vez, los servicios se describen a través de métodos y las propiedades a partir de atributos y asociaciones. Adicionalmente, puede utilizarse Object Constraint Language (OCL) para expresar condiciones adicionales.

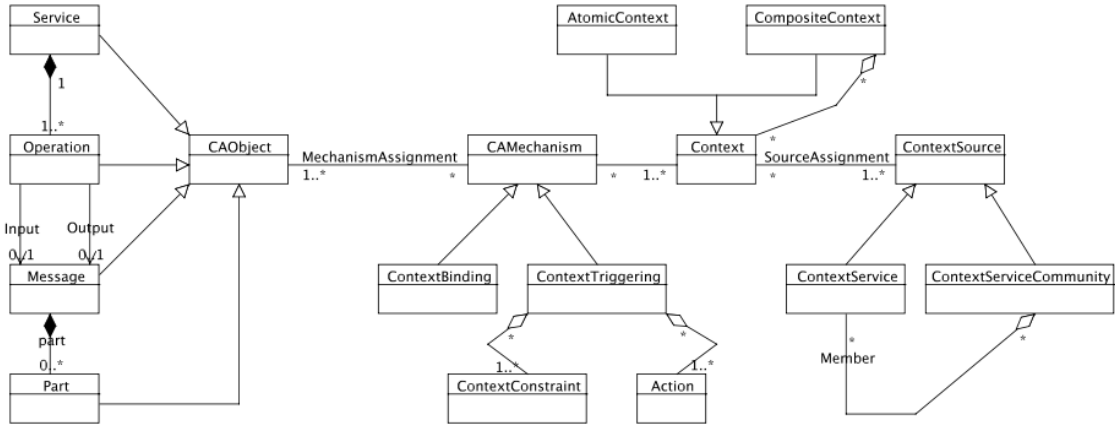


Figura 1. Metamodelo UML para la definición de contextos

Una de las arquitecturas diseñadas en base a un modelo orientado a objetos es Hydrogen [BDR07], en la que toda la información se representa, jerárquicamente a partir de objetos. Esto permite ofrecer al motor de razonamiento un subárbol jerárquico con la información que es relevante en un determinado instante. Esta arquitectura está especializada en la adquisición de contextos en dispositivos móviles. Para representar un contexto a partir de esta arquitectura, se debe considerar que un dispositivo consta de un contexto local y de un conjunto de contextos procedentes de otros dispositivos.

Otro ejemplo que utiliza esta aproximación orientada a objetos para la definición del contexto, es el proyecto GUIDE [CK00]. Este sistema implementa una guía turística sensible al contexto para la ciudad de Lancaster, Reino Unido. Se basa en un modelo de objetos para definir los aspectos relacionados con la posición (localización) de los usuarios. Los diferentes estados de los atributos, que implementan los objetos, se utilizan para describir el contexto. Como en cualquier sistema orientado a objetos, la obtención de la información se realiza a través de las interfaces que proveen los propios objetos.

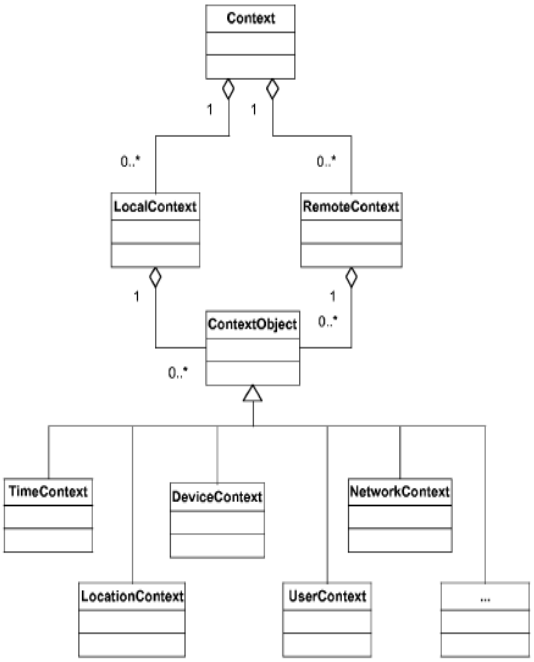


Figura 2. Aproximación orientada a objetos de Hidrogen.

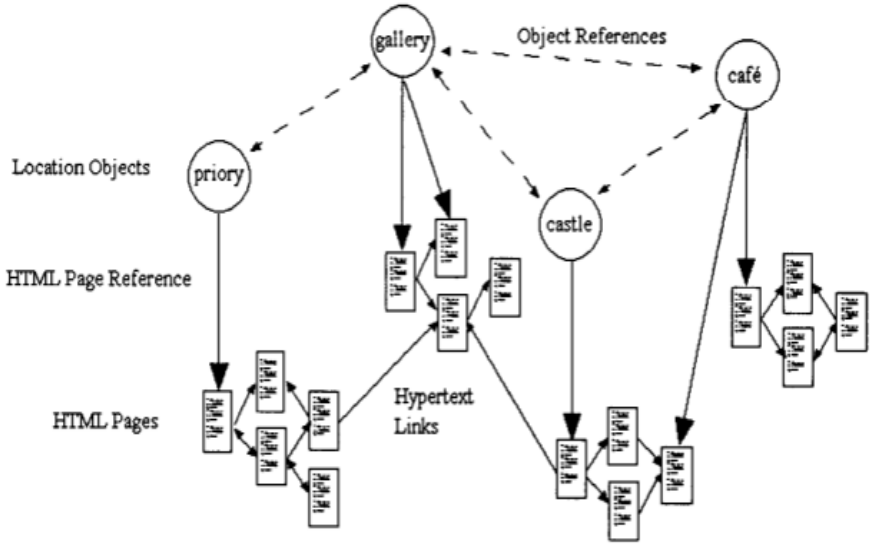


Figura 3. Modelo de información de GUIDE.

Modelos basados en rol-objeto

Los modelos basados en rol-objeto (subconjunto de los basados en hechos) tienen como objetivo la creación de modelos lo suficientemente formales como para realizar razonamientos y, a su vez, proporcionar modelos adecuados en tareas de diseño y análisis en ingeniería de software.

Con respecto al modelado de contexto, cabe destacar en este tipo de modelo su capacidad para procesar y hacer frente a la gran diversidad de datos e información recibida de fuentes heterogéneas, su capacidad de razonamiento, así como su capacidad de gestión de datos históricos [BBH+10]. En cuanto al soporte al razonamiento, una de las mayores ventajas que ofrece es que permite evaluar las aserciones con lenguajes similares a SQL ya que existen procedimientos para transformar este esquema a un esquema relacional.

Dentro de la aproximación rol-objeto, uno de los modelos más utilizados es el denominado Context Modelling Language (CML) [HI06]. CML está basado en Object-Rol Modelling (ORM) [H01] y está pensado para el modelado conceptual de bases de datos.

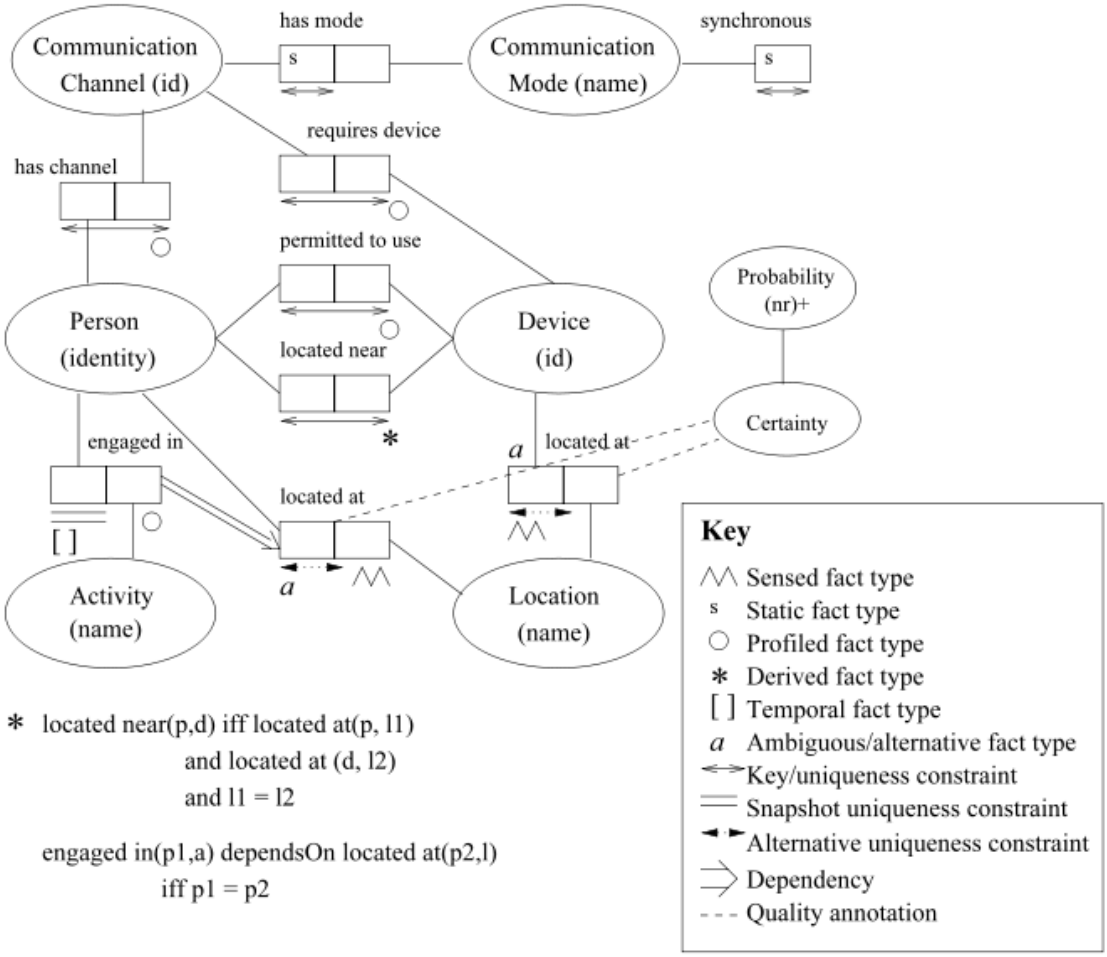


Figura 4. Ejemplo de un modelo CML [HI06]

El modelo captura las actividades de un usuario como hechos temporales que suceden a lo largo del presente, pasado y futuro. Además, modela las relaciones entre usuarios, los dispositivos y la comunicación a través de los canales, así como la localización (absoluta y

relativa) de los usuarios y de los dispositivos involucrados.

No obstante, se trata de un modelo plano en el que todos los tipos que representan al contexto son representados como hechos atómicos. Además, no ofrece soporte para el tratamiento de información imprecisa, es decir, no válida o con un cierto grado de incertidumbre.

Modelos espaciales

Los sistemas basados en este enfoque, utilizan un espacio o localización para determinar un contexto. Muchos de estos sistemas se basan en la aserción de hechos para definir una localización y se utilizan para dar respuesta a aplicaciones en las que se requiere conocer la situación y entorno de una entidad.

La información de localización suele ser proporcionada por los sistemas de posicionamiento que tenga el sistema. En función de cómo proporcionan esta información los sistemas de posicionamiento, existen dos tipos de sistemas de coordenadas:

- Coordenadas geométricas: los valores de las coordenadas representan puntos o áreas en un espacio métrico, normalmente dadas en coordenadas WGS84 (proporcionadas, por ejemplo, por un sistema GPS).

- Coordenadas simbólicas: las coordenadas se representan a partir de un identificador como, por ejemplo, un número de puerta o un nombre de lugar.

Los modelos espaciales permiten razonar sobre la localización y la relación espacial entre objetos. Los diferentes espacios que se definen, tienen asociadas relaciones de vecindad o afinidad [GBH+05] con otras localizaciones. Por ello, los sistemas que utilicen este enfoque deben responder adecuadamente ante cuestiones obvias como la posición de un objeto dentro de un determinado rango y su gestión con otros objetos en el caso de que éstos existan dentro de ese rango.

El modelo del mundo resultante, debe ser independiente de las aplicaciones particulares. Además, debe poder ser consultado utilizando un lenguaje que proporcione mecanismos de abstracción y desacoplamiento entre la aplicación, y la representación interna del mundo real. Para ello, los requisitos que debe cumplir un lenguaje de modelado espacial, de manera general, son [BBR02]:

- Describir la geometría de los objetos utilizando distintos sistemas de coordenadas: poder crear un modelo geométrico a partir de las descripciones y relaciones espaciales de un objeto (por ejemplo, si un objeto está encima de otro, si un objeto contiene a otro, etc.)

- Expresar información simbólica de los objetos (por ejemplo, su nombre y descripciones acerca de él).

- Expresar relaciones entre objetos que ayuden a describir aspectos topológicos del modelo (por ejemplo, la noción de proximidad en función de distancias geométricas).

- Soportar la descripción de objetos con diferentes niveles de detalle y expresar relaciones entre niveles.

Un ejemplo en el que se utiliza un enfoque basado en modelos espaciales es el NeXus Project. Se trata de una plataforma abierta implementada para facilitar el desarrollo de

aplicaciones sensibles al contexto y basadas en sistemas de localización. Así pues, la plataforma facilita la interacción entre este tipo de aplicaciones gracias, en gran medida, a la utilización y adopción de un mismo modelo de localización y un lenguaje estándar para la realización de las consultas. El núcleo del NeXus está formado por un modelo aumentado sobre el mundo, que proporciona un contexto de localización común para todas las aplicaciones context-aware. Este modelo incluye representaciones de objetos estáticos de la realidad, tales como casas, calles, oficinas, etc. Se describe utilizando el Augmented World Modeling Language (AWML) y puede ser consultado utilizando Augmented World Queryn Language (AWQL).

```
<awm>
  <object type="Room"

        NOL="nexus://...">
    <extension>

      <gml:st_polygon ...> ...

        <coord><X>5.0</X>
               ...

      <ml:st_polygon>
    </extension>

    <number>2.008</number>
     ...
  </object>
   ...
</awm>
```

Figura 5. ***Ejemplo de la descripción de una habitación en NeXus, utilizando AWML***

Modelos basados en la lógica

En los modelos basados en la lógica se definen condiciones, reglas y hechos que se utilizan para inferir y modelar nuevos contextos. Para ello, utilizan motores de inferencia y razonadores lógicos sobre términos y relaciones con el fin de obtener un determinado conocimiento (información procesada y de interés en el sistema).

Un ejemplo sencillo de un enunciado lógico sería siguiente:

```
brother(X,Y) if sibling(X,Y) and male(X).
child(X,Y) if parent(Y,X).
child(X,Z) if child(X,Y) and married(Y,Z).
child(X,Y) if daughter(X,Y).
child(X,Y) if son(X,Y).
daughter(X,Y) if female(X) and child(X,Y).
female(X) if daughter(X,Y).
female(X) if married(X,Y) and male(Y).
```

El proyecto Gaia [SB08] es un middleware diseñado con el objetivo de facilitar el desarrollo

y la ejecución de aplicaciones context-aware (concretamente, aplicaciones móviles, sensibles al contexto y ejecutadas en espacios activos). Para representar la información de contexto define reglas y hechos lógicos, que son utilizadas por su motor de inferencia para generar nuevo conocimiento y aumentar con ello su capacidad de razonamiento.

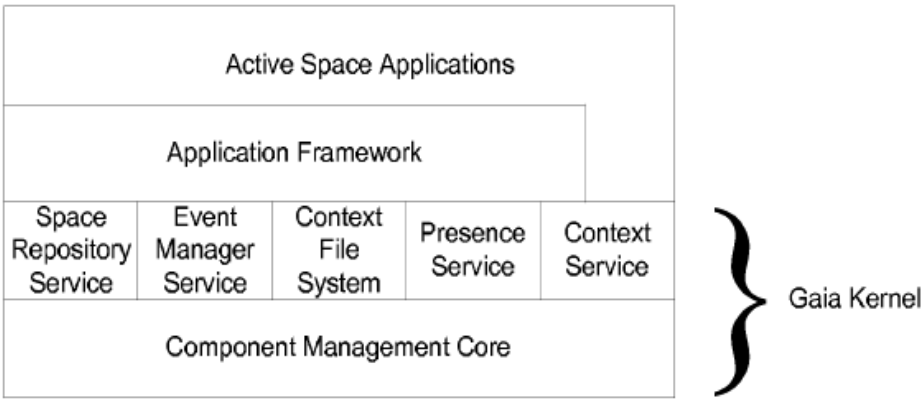


Figura 6. Representación de la arquitectura de Gaia

El servicio de gestión de eventos (Event Manager Service) es el responsable de la distribución de eventos en el espacio activo e implementa un modelo de comunicación basada en proveedores, consumidores y canales. A su vez, cada canal tiene uno o varios proveedores que le proporcionan información uno o más consumidores que la reciben. Con la ayuda del Servicio de contexto, las distintas aplicaciones pueden consultar la información del contexto. El sistema de ficheros del contexto (Context File System) hace que la información esté siempre disponible, independientemente de la ubicación del usuario. Este sistema de ficheros está construido en base a una jerarquía de directorios virtuales que representar el contexto, y donde la ruta de los componentes representa los tipos de contexto y sus valores.

Por ejemplo, el archivo que contendría el contexto:

```
localization == RM2401 && situation == reunion
```

sería:

```
/location:/RM2401/situation:/meeting
```

Para la definición de las reglas y hechos se utilizan lenguajes formales, que representan tanto las condiciones como el estado del modelo. Y es por ello, debido a su complejidad, por lo que se necesita cierta capacidad de dominio matemático para su comprensión.

Modelos basados en ontologías

Existen numerosas definiciones de ontologías ya que este término se utiliza en muy diversos campos (filosofía, documentación, inteligencia artificial, etc). Las ontologías son esencialmente descripciones de conceptos y sus relaciones. No obstante, dentro de los sistemas sensibles al contexto, la ontología se define como una extensión natural de los enfoques basados en CC/PP y RDF, que cumple los requisitos de: heterogeneidad, relaciones y dependencias entre conceptos y razonamiento.

Las ventajas del uso de ontologías son [BBH+10]:

- Expresividad del lenguaje, que en general es mayor que en el resto de modelos aunque existen ciertas limitaciones. Se basa en la utilización de un vocabulario común para representar y compartir el conocimiento, utilizando un formato de intercambio.

- Proporciona la suficiente capacidad semántica para integrar contexto de diferentes fuentes y que, además, permite la reutilización del conocimiento generado.

- Existencia de diferentes herramientas y motores para comprobar la consistencia y razonar sobre el mismo.

Normalmente este tipo de modelos está basado en OWL-DL [HPH03].

Gruber define las ontologías a partir de conceptos, instancias y relaciones entre los propios conceptos, funciones y reglas de restricciones o axiomas. Además, distingue distintos tipos de ontologías en función de diversos aspectos, tales como el ámbito de conocimiento al que se aplican, al tipo de agente al que están destinadas y al grado de razonamiento lógico y abstracción que permitan.

Uno de los aspectos que resulta interesante en el uso de ontologías es la capacidad de razonamiento y expresividad que ofrecen. Haciendo uso de una base de conocimiento (clases, propiedades y relaciones) y de la información asociada a un determinado contexto (por ejemplo, información de sensorización), pueden inferir y derivar, de manera automática, nuevo conocimiento. Además, a través de un chequeo robusto de todo el conocimiento del que disponen, pueden detectar inconsistencias en la información recibida (por ejemplo, si detectan que una misma persona está en dos habitaciones distintas al mismo tiempo).

CoBrA (Context-Broker Architecture) [CFJ03] fue uno de los primeros frameworks context-aware basado en ontologías. Utiliza OWL para el modelado del contexto y gestiona y mantiene la información de forma centralizada. Entre sus características más destacadas, cabe señalar el hecho de que permite a los usuarios definir sus propias políticas de privacidad para la protección de sus datos.

CoBrA Ontology Classes		CoBrA Ontology Properties	
"Place" Related	Agents' Location Context	"Place" Related	Agent's Location Context
Place AtomicPlace CompoundPlace Campus Building AtomicPlaceInBuilding AtomicPlaceNotInBuilding Room Hallway Stairway OtherPlaceInBuilding Restroom Gender LadiesRoom MensRoom ParkingLot	ThingInBuilding SoftwareAgentInBuilding PersonInBuilding ThingNotInBuilding SoftwareAgentNotInBuilding PersonNotInBuilding	latitude longitude hasPrettyName isSpatiallySubsumedBy spatiallySubsumes accessRestricted-ToGender lotNumber	locatedIn locatedInAtomicPlace locatedInRoom locatedInRestroom locatedInParkingLot locatedinCompoundPlace locatedInBuilding locatedInCampus
"Agent" Related	Agent's Activity Context	"Agent" Related	Agent's Activity Context
Agent Person SoftwareAgent Role SpeakerRole AudienceRole IntentionalAction ActionFoundInPresntation	PresentationSchedule Event EventHappeningNow PresentationHappeningNow RoomHasPresentationHappeningNow ParticipantOfPresentation-HappeningNow SpeakerOfPresentationHappeningNow AudienceOfPresentationHappeningNow PersonFillsRoleInPresentation PersonFillsSpeakerRole PersonFillsAudienceRole	hasContactInformation hasFullName hasEmail hasHomePage hasAgentAddress fillsRole isFilledBy intendsToPerform desiresSomeone-ToAchieve	participatesIn startTime endTime Location hasEvent hasEventHappeningNow invitedSpeaker expectedAudience presentationTitle presentationAbstract presentation eventDescription eventSchedule

Tabla 2. Listado de las clases y propiedades de la Ontología de CoBrA.

```
<owl:Class rdf:ID="Place">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlace"/>
    <owl:Class rdf:about="#CompoundPlace"/>
  </owl:unionOf>
  …
</owl:Class>

<owl:Class rdf:ID="AtomicPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
         rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#spatiallySubsumes"/>
      <owl:cardinality>0</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#AtomicPlaceInBuilding"/>
    <owl:Class
       rdf:about="#AtomicPlaceNotInBuilding"/>
  </owl:unionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="CompoundPlace">
  <rdfs:subClassOf rdf:resource="#Place"/>
  <owl:disjointWith rdf:resource="#AtomicPlace"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#isSpatiallySubsumedBy"/>
      <owl:allValuesFrom
         rdf:resource="#CompoundPlace"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
         rdf:resource="#spatiallySubsumes"/>
      <owl:allValuesFrom
         rdf:resource="#Place"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Campus"/>
    <owl:Class rdf:about="#Building"/>
  </owl:unionOf>
</owl:Class>
```

Figura 7. Definición parcial (OWL) de las clases AtomicPlace y CompoundPlace definidas en CoBrA.

El modelo baso en ontologías es de los más expresivos y con mayor soporte a los aspectos dinámicos que caracterizan la computación ubicua o sensible al contexto. No obstante, los complejos motores ontológicos necesarios para la gestión de las ontologías en uso, hacen que requieran gran capacidad de procesamiento, con un alto coste computacional, y disminuyen su rendimiento. Además, los constructores de OWL-DL (de tipo rol-valor-mapa) no permiten la definición de relaciones compuestas con tal de preservar su capacidad durante la toma de decisiones [BBH+10].

Modelos híbridos

Los modelos híbridos integran diversos tipos de modelos de contexto con el fin de mejorar las capacidades que éstos aportan de forma individual, creando con ello modelos más flexibles y aplicables a dominios más generales.

Existen diversas aproximaciones de este tipo de modelos. Una de ellas, es la propuesta por Henricksen en [HLJ04], donde describe un modelo que integra las características de un modelo basado en ontologías, con las de otro, basado en hechos (CML). Así pues, este modelo híbrido resultante, combina la capacidad de los modelos CML para procesar información de contexto ambigua o imprecisa, con la interoperabilidad y el manejo de contextos heterogéneos que suministran los modelos basados en ontologías.

La plataforma CoCA (Collaborative Context-Aware Service Platform) [ESL07] es otro ejemplo que se basa en un modelo híbrido para el modelado del contexto. Se trata de una plataforma que puede utilizarse para el desarrollo de aplicaciones context-aware en entornos ubicuos. Utiliza un modelo ontológico para gestionar la información semántica del contexto, y un enfoque basado en una base de datos relacional para almacenar los datos. En función de la aplicación de unas determinadas heurísticas, procesa sólo los datos que considera relevantes en cada momento.

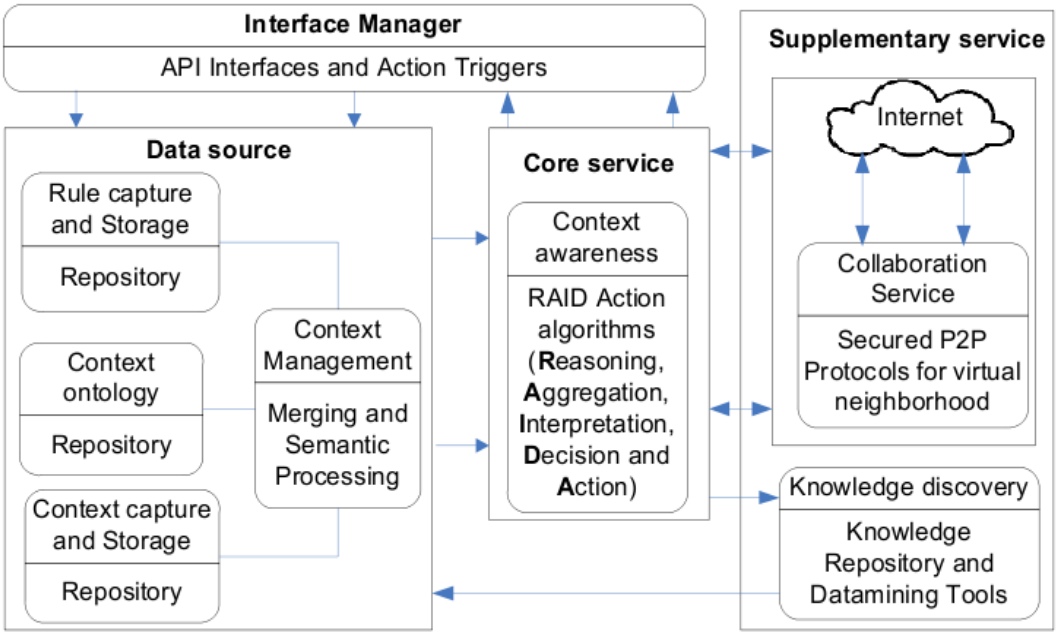


Figura 8. Arquitectura de la plataforma CoCA.

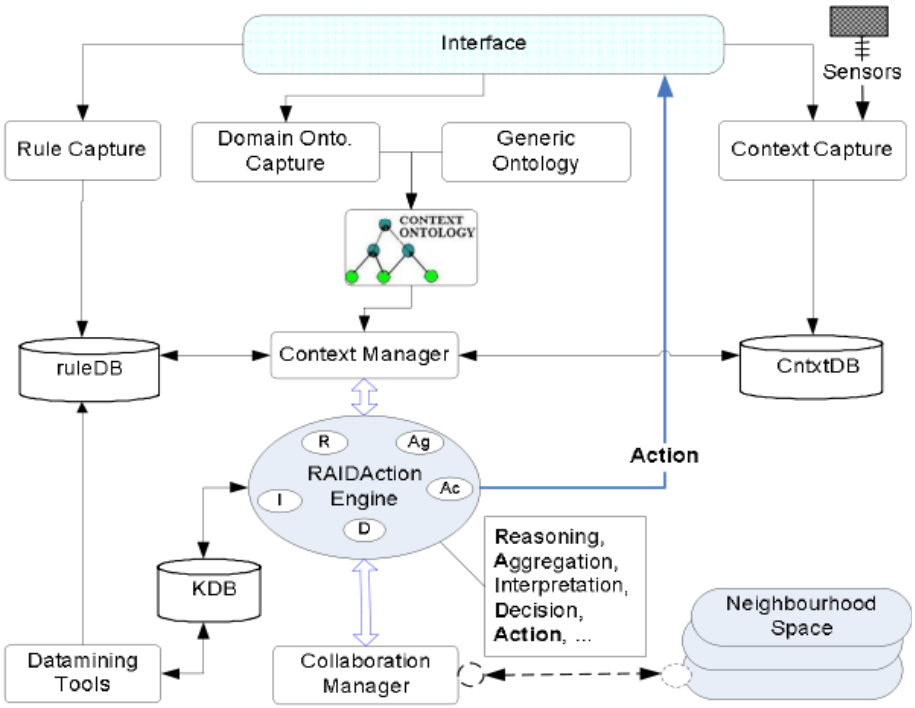


Figura 9. *Componentes de la plataforma CoCA*

No obstante, algunos de los problemas comunes que surgen a de la unión de distintos modelos son: la necesidad de converger en un único esquema de razonamiento, la integración de semánticas, en ocasiones de carácter muy diverso y, con ello, la alta complejidad del modelo resultante.

Otros modelos

Existen otras soluciones, menos extendidas, empleadas para el modelado del contexto. Un ejemplo, es el proyecto TEA [GSB02], en el que el contexto se describe a partir de un vector bidimensional que consiste en un conjunto de valores simbólicos que describen situaciones con una probabilidad asociada.

Dentro del dominio de los sistemas sensibles al contexto, este modelo está formado por una capa de bajo nivel, que recibe información sin procesar, procedente de dispositivos de diversa naturaleza (sensores). Una capa de nivel superior, realiza una interpretación semántica de los datos recibidos en el nivel inferior y crear un comportamiento.

Arquitectura de los sistemas sensibles al contexto

La implementación de una aplicación sensible al contexto se suele realizar mediante la adopción de un middleware estructurado por capas. Esto facilita la implementación de las propias aplicaciones, ya que permite que éstas se abstraigan de la gestión interna del sistema en cuanto al manejo y procesamiento de los datos que el entorno va produciendo. Así pues, el objetivo principal es hacer transparente a la aplicación que va a ser implantada el proceso de recogida de datos, su procesamiento y la detección de los contextos definidos. Otro de los motivos de utilizar un middleware como ayuda a la construcción de aplicaciones sensibles al contexto, es el de enmascarar los problemas de heterogeneidad, la asincronía en la recepción de los datos y el dinamismo que puede presentarse con los diferentes sensores [BB09].

Tal y como se ha comentado, resulta difícil implantar un sistema general que sea válido para cualquier aplicación, por lo que en la mayoría de casos se sigue una arquitectura estructurada en niveles como la que se presenta a continuación [SB08] y que puede estudiarse detenidamente en los trabajos de [CGK05,CRE05,YKW+02]:

- Sensado del contexto: en el que se adquieren los datos por los diferentes sensores y estos se refinan, en caso necesario, para obtener datos estructurados que representen contextos de mayor nivel. La tarea esencial de este nivel es la recolección de datos y la transformación de estos (si se precisa) en información contextual. Los sensores pueden ser tanto físicos (que miden diferentes variables del entorno de forma directa), como lógicos (que son sensores virtuales que se crean en el sistema y proporcionan valores según el estado del mismo).

- Gestión del contexto: los datos se actualizan en el repositorio de contexto. En este nivel también puede englobarse el razonador y el motor de contexto, que son los encargados de analizar los datos y determinar las situaciones de contexto que sean interesantes. En este caso los datos que provienen de los diferentes sensores son refinados y preprocesados antes de ser analizados por el motor de contexto implementado. A su vez, esta capa puede dividirse en los siguientes apartados:

    - Repositorio: representa al contenedor en el que se almacena el contexto actual y, si es posible, la historia y los estados anteriores. Estos datos son almacenados según las estructuras definidas que vienen dadas por el modelo de contexto a utilizar.

    - Razonamiento: responsable de detectar el contexto actual en el que se encuentra el sistema, en base a la información que obtenida del repositorio. El diseño e implementación del motor estará estrechamente ligado al modelo de contexto que se haya optado utilizar. En general, los contextos son definidos por reglas o patrones [JLR+07][BC04] que son activados y comunicados a la aplicación o al nivel superior en el caso de su cumplimiento. Las reglas o patrones son definiciones de estados o valores asociados a diferentes atributos del sistema y que son introducidos en el sistema. Periódicamente, éste revisa si estos concuerdan con el estado actual en que se encuentra el sistema, tomando la acción pertinente en caso positivo.

- Nivel de API: facilidades que ofrece el middleware a las aplicaciones sensibles al contexto de nivel superior, comunicando los diferentes contextos mediante eventos, mensajes o cualquier otra forma que implemente el interfaz. A este nivel ya suelen tratarse aplicaciones específicas como la "adaptación contextual" (donde la aplicación se adapta o realiza las funciones necesarias según el contexto proporcionado), "descubrimiento de recursos contextual" (en donde el contexto dinámico permite descubrir recursos en función del contexto actual) y la *"context augmentation*" (donde el contexto actual es enriquecido con datos e información digital). A partir de aquí es donde la aplicación se programa.

Al igual que en la implementación de los sistemas software tradicionales, existen dos modelos que también pueden aplicarse a las arquitectura de los sistemas sensibles al contexto: centralizado o no. En un modelo centralizado (servidor de contextos) tanto la implementación, como el diseño resultan sencillos. No obstante, en los casos en los que exista un gran volumen de información, pueden producirse problemas de congestión (se trata de un cuello de botella). Además, a esto hay que sumarle problemas de disponibilidad debido a su baja tolerancia a fallos (consecuencia de la existencia de un único nodo central que da soporte a todo el sistema). Un modelo descentralizado punto a punto (peer to peer), no tiene problemas de congestión, en cuando al procesamiento de la información, y al no disponer de un servidor adicional aporta una alta tolerancia a fallos. No obstante, este tiempo de sistemas no puede ser utilizado en aplicaciones con un alto grado de dinamismo.

Se pueden encontrar diferentes proyectos que tratan de ofrecer middlewares para la construcción de aplicaciones sensibles al contexto y que, de alguna u otra forma, siguen una estructura similar a la presentada anteriormente. En el artículo [E07] se listan algunos de ellos.

Aplicaciones comerciales/proyectos

Unos de los primeros frameworks de propósito general para la gestión de contexto fueron Context Toolkit (2001) e Hydrogen (2002). En ambos casos, eran capaces de tratar información de contextos de tipos distintos y estaban basados en arquitecturas punto a punto. No obstante, Context Toolkit utilizaba un modelo de contexto muy simple basado en tuplas de tipo atributo-valor, mientras que Hydrogen hacía uso de un modelo más complejo y orientado a objetos.

Posteriormente, surgió CoBrA (2003) como un sistema sensible al contexto y basado en agentes. Fue el primero en utilizar OWL y una ontología para el modelado del contexto, así como una política de privacidad. Al igual que CoBrA, en SOCAM (Service-Oriented Context-Aware Middleware, 2004) se utilizó OWL para modelar el contexto aunque, en este caso, además se empleó una estructura jerárquica para el diseño de la ontología del contexto.

En 2008, se fue un paso más allá al emplear en un mismo sistema dos enfoques distintos. Así, la plataforma CoCA, basada en HcoM (Hybrid Context Management) combina la utilización de ontologías para definir la semántica del contexto, con un enfoque basado en una base de datos relacional, para gestionar los datos.

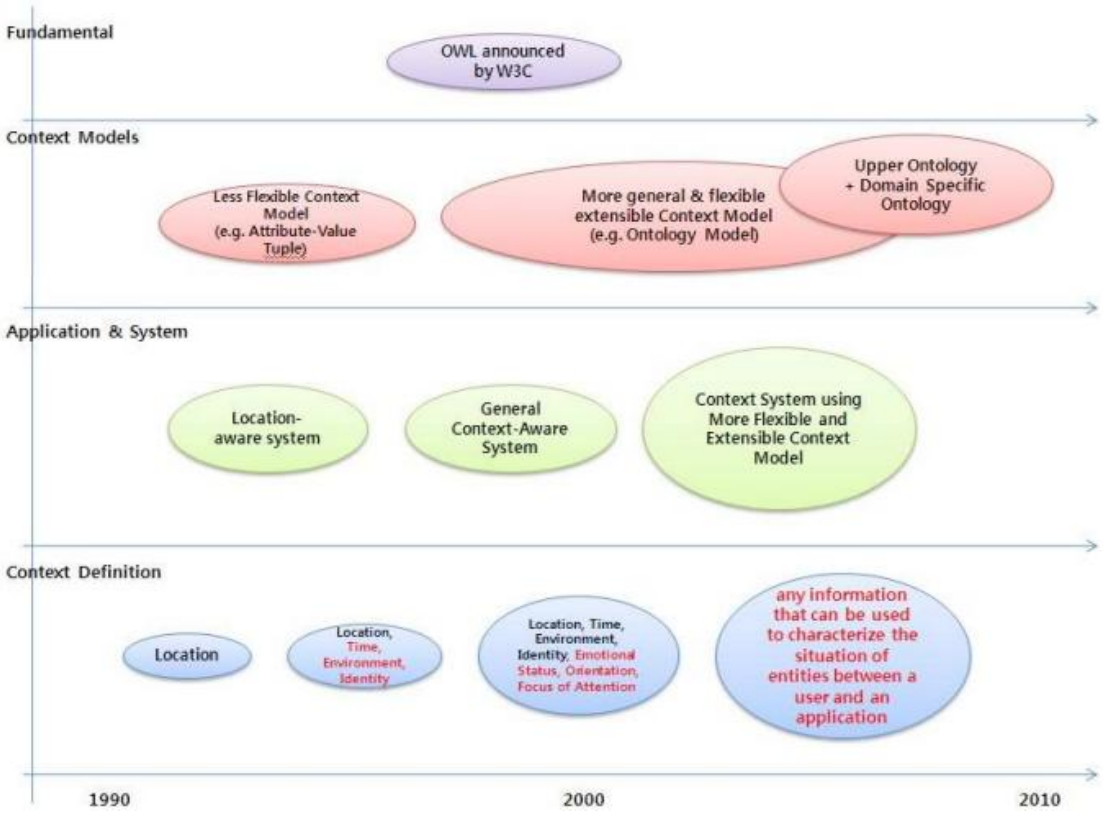


Gráfico 1. Evolución de los sistemas context-aware.

Con toda esta información, resulta fácil apreciar cómo los primeros modelos sencillos y específicos para un tipo de contexto o dominio determinado, han evolucionado hacia otros más complejos, aplicables a sistemas de propósito general y que dan soporte a varios tipos de contextos [LPL09].

En el trabajo de [HNC+07] se encuentra un trabajo con un amplío listado de aplicaciones que pueden ser analizadas.

Finalmente, se adjuntan algunos enlaces a las páginas web de los proyectos de las aplicaciones que se han mencionado a lo largo del documento:

- Context Toolkit: http://contexttoolkit.sourceforge.net/
- CoBrA: http://cobra.umbc.edu/
- GUIDE: http://www.guide.lancs.ac.uk/overview.html
- NeXus Project: http://www.nexus.uni-stuttgart.de/index.en.html
- Gaia: http://gaia.cs.uiuc.edu/html/publications.htm

Descripción técnica del proyecto

Tras realizar el estudio previo, se ha observado que muchas de las aplicaciones sensibles al contexto existentes se basan en ontologías o soluciones híbridas para el modelado de la información del entorno. No obstante, el elevado coste computacional que requieren los motores ontológicos, y su consecuente disminución de la capacidad de procesamiento, han hecho que en Mobiware se optara por un modelo basado en patrones o reglas lógicas. Esta decisión fue debida a que en las primeras fases de análisis y diseño del sistema se optó por primar una respuesta rápida del servidor (procesamiento ágil de la información en tiempo de ejecución) ante una mayor capacidad expresiva a la hora de crear contextos.

A grandes rasgos, el sistema se define como un middleware en el que la definición de una única arquitectura permite la gestión de contenidos de distinta naturaleza. En este caso, al igual que se proponía en los diversos sistemas de la literatura, y teniendo siempre como referencia el desacoplamiento de los componentes que lo integran, sí se ha seguido una estructura dividida en capas. En esta arquitectura modular, los niveles que se han definido son los siguientes: sensorización, gestión de contextos y API de aplicaciones.

El software de conectividad desarrollado, es capaz de administrar contenidos relacionados con un conjunto de temas. Un ejemplo de este uso sería en usuarios que estuvieran interesados en recibir información o noticias acerca de deportes, cultura, ciencia, etc. Además, el middleware permite gestionar contenidos asociados a contextos o entornos adaptativos. Generalmente el contexto, hace referencia a todo lo que envuelve a un punto o centro de interés, proporcionando nuevas fuentes de información e incrementando, con ello, el conocimiento sobre esa localización, usuario, dispositivo, etc. Para la correcta ejecución de este tipo de entornos, ha sido necesaria la definición de un modelo, que permita adaptar el software conectado a las características en ejecución de cada instante, facilitando con ello la entrega del contenido más adecuado. Así pues, al hablar de contextos en el ámbito de Mobiware, debe hacerse hincapié en que el objetivo es el ofrecimiento de la información adecuada en función de las preferencias y de la condiciones de ejecución en un instante determinado; según el tipo de dispositivo, el entorno de ejecución, el tipo de aplicación, etc.

La obtención de contextos puede estar relacionada con dispositivos que envíen periódicamente cierta información de sensorización, por ejemplo, datos de localización, temperatura, humedad, presión, o incluso, datos relacionados con un usuario que se encuentre asociado, a su vez, con otro dispositivo; ritmo cardíaco, tensión arterial, etc. Todo ello, va a permitir la adaptación del contexto no solo al entorno, sino también al estado del usuario, lo que puede resultar muy útil en aplicaciones sanitarias de monitorización de pacientes.

Como mecanismo para el intercambio de mensajes entre clientes del middleware, se ha optado por seguir la filosofía *publish/subscribe*, frente a los sistemas tradicionales de tipo clientes-servidor. Este tipo de sistemas resultan mucho más efectivos en entornos dinámicos, ya que el número de consumidores (*subscribers)* y productores (*publishers*) de mensajes puede variar en cortos períodos de tiempo, por ejemplo, debido a fallos en la conexión (sistemas con procesamiento de la información en tiempo real).

Dentro del entorno de Mobiware, el concepto de *publish/subscribe*, va a permitir que los clientes se suscriban a determinados eventos relacionados con contextos, así como a temas relacionados con contenidos. En cada caso, el servidor enviará los mensajes relacionados con esas suscripciones conforme se vayan produciendo.

Así pues, el middleware permite dos tipos de suscripciones:

1. Suscripciones relacionadas con la detección de contextos. Estos registros serán gestionados internamente por el motor de contextos y el motor de sensorización del middleware.

2. Suscripciones a temas. Estos registros serán gestionados por el motor interno de gestión de contenidos y una arquitectura de soporte como sistema de mensajería.

Del mismo modo, las publicaciones serán gestionadas internamente por los motores adecuados del middleware, en función de si se trata de información relacionada con temas y contenidos, o bien, con el resultado de la evaluación de un contexto.

A la hora de gestionar las actualizaciones de la información en un contexto dinámico, se han tomado las siguientes decisiones:

- Con respecto al repositorio de información, los datos de sensorización que los dispositivos móviles van a ir recogiendo, se van a almacenar periódicamente en él, de modo que estén accesibles por el motor de inferencia y éste pueda inferir nueva información o conocimiento. La actualización de este repositorio de información histórica se ha diseñado para que pueda ser configurable en función de la aplicación que va a hacer uso del middleware. Por ejemplo, puede resultar interesante almacenar en el repositorio información de sensorización durante cortos intervalos de tiempo (por ejemplo, cada pocos minutos, información relacionada con la localización de un dispositivo). Pero esta misma idea, puede no tener sentido para otro tipo de sensores en los que no interesan conocer su valor en un instante, sino su evolución a lo largo de un periodo de tiempo (por ejemplo, la humedad, la temperatura, etc.). Para ello, se han creado las siguientes variables:
    - Tiempo máximo de histórico (en milisegundos): superado este umbral, los nuevos datos recibidos irán reemplazando los más antiguos ya almacenados.
    - Tiempo mínimo de recepción (en milisegundos): tiempo mínimo que debe pasar entre la recepción de dos valores para un mismo sensor. Si se recibe un valor antes de que haya pasado este tiempo mínimo, será ignorado.

- Con respecto a la información recogida de los sensores, debido a que se trata de una arquitectura dividida en módulos, será el módulo/capa de sensorización la encargada de recibir dicha información, y enviarla a una capa de nivel superior para su procesamiento.

- Con respecto a la difusión del nuevo conocimiento generado, se ha optado por seguir un mecanismo basado en *listeners* (para la creación de los mensajes sobre la información generada) y de *publish/subscribe* (para la difusión de estos mensajes).

El sistema debe ser consciente de que es posible que los datos obtenidos sean imprecisos e incluso no válidos para poder gestionarlos correctamente. Para el tratamiento de esta incertidumbre, junto a cada valor enviado, el dispositivo debe enviar meta-información acerca de él: grado de validez, precisión, etc.

Tecnología empleada

La tecnología empleada en la implementación del middleware ha sido la siguiente:

1. Lenguaje de programación:
    - Java: compatibilidad a partir de la versión de JDK 1.6 o superior.

2. Configuración y gestión de los objetos de negocio:
    - Spring IoC.

3. Implementación de los servicios web para la comunicación entre componentes del sistema distribuidos en distintos hosts:
    - Spring Web Services v1.5.6
    - Axis + WSDL
    - JDOM
    - Dozer v5.0

4. Serialización de objetos XML:
    - XStream v1.3.1

5. Plataforma de desarrollo:
    - Eclipse Helios v3.6.1

6. Backend de mensajería:
    - XMLBlaster v2.0.0 2009-09-09 'Premium'

7. Logging frameworks:
    - Simple Logging Facade for Java (slf4j) v1.6.1
    - Logging for Java (log4j) v1.2

8. Plataformas de comunicación con el middleware (implementadas hasta el momento):
    - Android.
    - J2ME.

9. Servidor Web:
    - Apache Tomcat v7.0.12

Diseño de la arquitectura

Al igual que el lenguaje de programación utilizado, el diseño del middleware se ha realizado cuidando los principios de la orientación a objetos: abstracción, encapsulamiento, modularidad y jerarquía entre objetos, entre otros. Utilizando un diseño modular se ha conseguido un mayor grado de desacoplamiento, lo que permite que el middleware pueda ser dividido en pequeñas partes funcionales (módulos) que, a su vez, pueden ser reutilizados en distintos entornos, dotando al sistema de una fuerte flexibilidad. Además, el uso de protocolos e interfaces estándar en las comunicaciones entre las diferentes partes del sistema, hace que aumente su interoperabilidad.

Para ofrecer una visión global del middleware y del funcionamiento de sus interfaces, se adjunta algún esquema (ver anexo al final de la sección de resultados obtenidos) que puede utilizarse como material de apoyo.

A continuación, se describen los módulos que forman Mobiware. Para cada uno de ellos, se presentan las dependencias con otros módulos y la funcionalidad proporcionada al sistema. Desde el punto de vista de la implementación, cada uno de ellos se ha definido como un proyecto de la plataforma Eclipse.

Mobiware-core

Representa el núcleo de Mobiware. En este módulo se definen las interfaces del sistema que son comunes y/o accesibles tanto desde los dispositivos clientes, como desde el propio núcleo del middleware para su gestión interna. Representan a los servicios que el middleware ofrece y que, cualquier dispositivo cliente remoto, debe utilizar para establecer la comunicación con el middleware. Por esta razón, se consideran esenciales para el buen funcionamiento y uso del sistema.

Mobiware-utils

Este componente está formado por clases que se utilizan desde otros paquetes del sistema y que implementan funcionalidad común. Es decir, se trata de un módulo de factorización, utilizado para evitar duplicación de código, en el que se definen constantes globales accesibles tanto desde los dispositivos clientes, como desde el propio servidor.

Mobiware-context

Este componente recoge la funcionalidad relacionada con la gestión de contextos. En el ámbito de Mobiware, el modelado del contexto viene dado por el entorno que rodea al usuario. Es decir, todas las condiciones y eventos que pueden ser útiles a la hora de extraer nueva información, pudiendo actuar en consecuencia. Con tal de facilitar la comprensión de la definición de contextos, a continuación, se definen tres escenarios de ejemplo con características muy diversas:

- Escenario 1: Suscripción a noticias/deportes, etc. En este caso, a través del middleware se envía una lista de tópicos a los que un usuario puede suscribirse. Cada vez que un proveedor de contenidos (por ejemplo, otro usuario, aplicación, etc.) envía a través del middleware un nuevo contenido, los usuarios que están suscritos a él lo reciben.
    - En este caso, no se hace uso del contexto, es decir, no se considera el estado del usuario suscriptor.

- Escenario 2: Monitorización de pacientes. A un paciente hospitalizado podría proporcionársele un dispositivo capaz de activar diversas alarmas al personal adecuado.
    - Para este escenario, el contexto del paciente viene dado por el conjunto de situaciones que representan los diferentes estados del paciente.

- Escenario 3: Visita guiada a una ciudad. Podría considerarse una aplicación de visita guiada en un espacio exterior y relativamente amplio entre puntos de interés. El usuario se podría suscribirse a un tópico (por ejemplo, información turística en Valencia). Cuando éste se acerca físicamente a un lugar de interés, recibiría información acerca de su ubicación actual.
    - En este caso, se requiere modelar la localización del dispositivo cliente como contexto. Para ello, cada contenido almacenado en el servidor tendría asociado un contexto, representado mediante datos de localización, que sería el utilizado para su publicación en función de las coordenadas enviadas por el dispositivo cliente.

Para definir los contextos de los escenarios 2 y 3, se utilizan uno o más patrones. Cada uno de éstos puede estar compuesto, a su vez, por un conjunto de expresiones lógicas y, del mismo modo, éstas pueden estar compuestas por un conjunto de operadores numéricos, de cadenas, valores procedentes de la sensorización de los dispositivos conectados, valores y marcas temporales, etc. Además, un patrón está asociado a uno o más tipos de entidades. Estos tipos (dispositivo, temporal...) se definen en los metadatos del patrón durante su definición. De esta forma, cuando el motor de contextos detecte un cambio en una entidad, reevaluará todos los patrones en los que el tipo de esa entidad esté involucrado.

La definición de estos patrones y sus transformaciones se realiza en el módulo mobiware-treePatterns.

Mobiware-treePatterns

Este módulo contiene todas las clases relacionadas con el tratamiento de patrones: desde que son recibidos por los endpoints hasta que se transforman y adaptan a entidades evaluables por el motor de contextos.

Mobiware-sensing

En este módulo están definidas todas las clases e interfaces relacionadas con la gestión de la sensorización. Cada aplicación, relacionada con dispositivos sensibles al contexto y ofrecida por el sistema, dispondrá de una configuración establecida, que podrá estar almacenada en el propio servidor o en un dispositivo de almacenamiento externo.

Debido a que la meta-información de un sensor varía en función de su marca, modelo, proveedor, etc. se ha optado por utilizar nombre canónico o alias para referenciar el concepto que representa. Por ejemplo, el nombre canónico “longitud” representa el ángulo a lo largo del ecuador desde cualquier punto de la Tierra y, esta clave, longitud, será la utilizada tanto por el motor de sensorización para la gestión de los datos recibidos por los sensores, como por el motor de contextos durante la evaluación de los patrones. No obstante los valores de la longitud podrán proceder de diversos dispositivos con sensores distintos, aunque el concepto seguirá siendo el mismo en todos ellos.

Además de datos de sensorización procedentes de sensores físicos localizados en dispositivos remotos, otros valores que resultan interesantes recibir son aquéllos que, tras su evaluación conjunta, cumplen una cierta condición. Este tipo de valores se han denominado limitadores. Así pues, un limitador puede verse como un sensor lógico localizado en el cliente, que puede estar compuesto, o no, por uno o más sensores físicos. Ejemplo de limitadores son: un sensor lógico que se encargue de enviar los datos de sensorización cada cierto periodo de tiempo, un sensor lógico que detecte la caída libre del dispositivo sobre el que se define, etc.

Por otro lado, debido a que los datos de sensorización proceden de sensores que pertenecen a proveedores/modelos y marcas distintas, se ha definido, para cada nombre canónico y limitador, unos parámetros de normalización. De este modo, antes de que un valor pase a ser almacenado en el contenedor de sensorización, éste se procesa y normaliza teniendo en cuenta los normalizadores que se han configurado.

Una vez definida la posible procedencia de los datos de sensorización, resulta fácil suponer que cada una de las aplicaciones registradas en el servidor dispondrá de cierta información de configuración sobre los limitadores disponibles para su correcto funcionamiento, y sobre los nombres canónicos o alias a partir de los cuales podrán definirse nuevos contextos.

Mobiware-ws

Este componente implementa la funcionalidad que permite el acceso remoto al middleware a través de servicios web. De este modo, sirve de puente entre las aplicaciones remotas localizadas en los clientes y el servidor, representado mediante las implementaciones de las interfaces del paquete mobiware-core.

Se ha elegido Spring Web Services (http://static.springsource.org/spring-ws/sites/1.5) como framework de apoyo para la implementación de los servicios web. El empleo de este framework hace más fácil, y de un modo más flexible, la interacción con la definición declarativa de las instancias vía Spring.

Spring Web Services está alojado dentro de un contenedor de aplicaciones web. Recibe mensajes SOAP de petición de contenidos, dentro de peticiones HTTP, los analiza y, en base a un mapeo establecido del elemento raíz de la petición, se decide qué endpoint será el encargado de servir la petición. En Mobiware, un endpoint puede verse como el punto al que irán dirigidas las peticiones remotas de los clientes, es decir, la clase que servirá de entrada al servicio. En ella, se decodificarán (unmarshalling) y procesarán los datos recibidos y enviados como documentos XML a través del servicio. De igual modo, las respuestas del servicio llegarán a los clientes, a través de los servicios web, utilizando los endpoints de codificación (marshalling) definidos en cada caso.

Para la codificación/decodificación (marshalling/unmarshalling) de la información enviada/recibida en los documentos XML, el middleware utiliza JDOM como biblioteca para la manipulación de los datos.

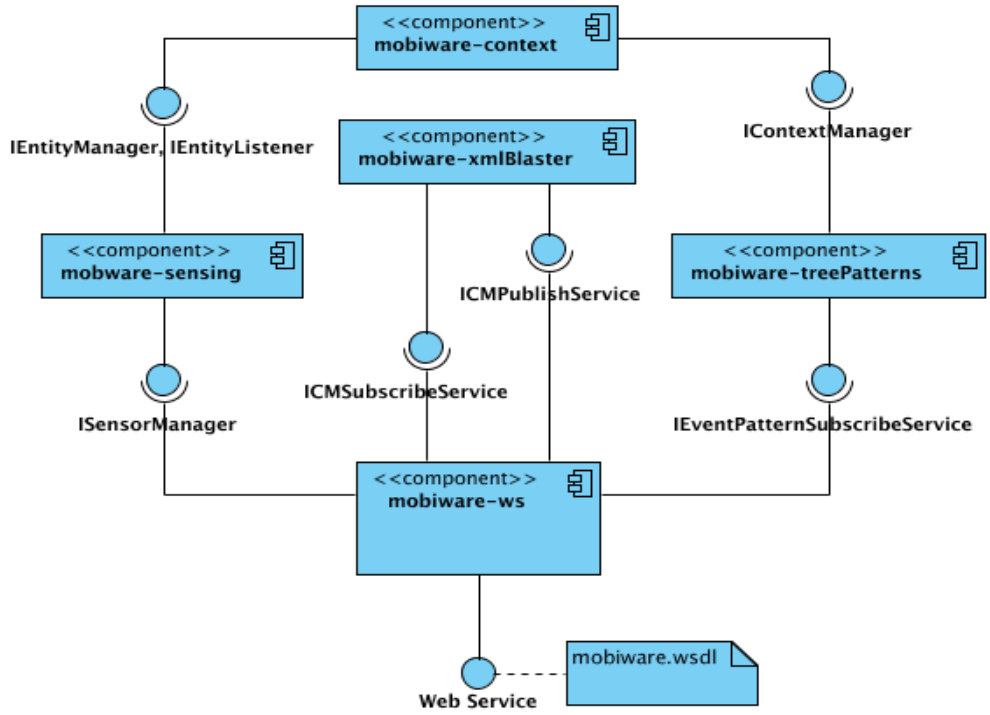


Figura 10. Distribución del módulo mobiware-ws

Mobiware-wsclient

Este módulo actúa como cliente del componente servidor Mobiware-ws. Ofrece, a los clientes externos del middleware, las interfaces de los servicios que deberán utilizar para su interacción con el sistema. Para la implementación de este componente se ha utilizado el generador de stubs que incluye Axis contra la WSDL (Web Services Description Language) generada por Mobiware-ws. No obstante, gracias al desacoplamiento e independencia de los módulos que integran este componente, los usuarios pueden desarrollar y utilizar otras implementaciones, distintas a Axis, que implementen la funcionalidad de las interfaces de los servicios (por ejemplo, éste podría ser el caso de sistemas basados en Android).

Desde un punto de vista externo, el cliente sólo es consciente de los servicios ofrecidos por el middleware, a los que tiene acceso a partir de las interfaces que los implementan. Así pues, la abstracción al cliente de la estructura interna y de la implementación de este componente, facilita su uso.

A parte de los servicios e interfaces ofrecidas en el componente Mobiware-core, este módulo implementa y contiene la interfaz del servicio de sensorización. Ya que pueden existen diversas soluciones, dependiendo del sistema o dispositivo utilizado (Android, J2ME, iOS, etc…), se ha permitido al usuario la posibilidad de realizar sus propias implementaciones:

- ISensingService: Servicio de inicio/fin de envío de información sobre sensores. Los dispositivos clientes intercambiarán los datos de sensorización e inicialización de sensores para una aplicación en el servidor a partir de este servicio. Estos datos son los que se utilizarán en la evaluación de los eventos de patrón en los que intervengan valores sensibles al contexto (estado actual).

Desde un punto de vista interno, en base a criterios funcionales, este componente se divide en cinco módulos independientes que implementan los servicios ofrecidos. Además, cada uno de ellos, consta de un bridge en el que se definen los métodos utilizados para la comunicación con el servidor. Por ejemplo, en el caso de una implementación basada en Axis, contendrá la gestión del stub utilizado para el envío de los datos. Los módulos definidos son los siguientes:

- WebServicesAuthenticationService: Este módulo implementa la funcionalidad del servicio de autenticación. Para ello, hace uso de IAuthenticationBridge (a través de sus métodos initSession y closeSession) para enviar las credenciales del usuario al servidor.

- WebServicesPublishSubscribeService: Implementa la funcionalidad de los servicios de publicación y suscripción a contenidos.

- WebServicesPatternServices: Contiene la funcionalidad del servicio encargado de realizar la suscripción a eventos de patrón y del envío de nuevos patrones.

- WebServicesSensingService: Implementa la funcionalidad del servicio de sensorización.

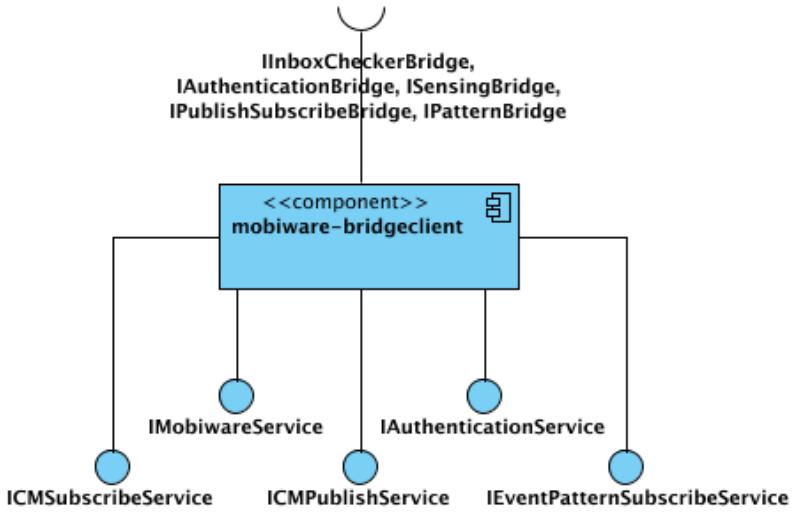


Figura 11. Servicios proporcionados por Mobiware

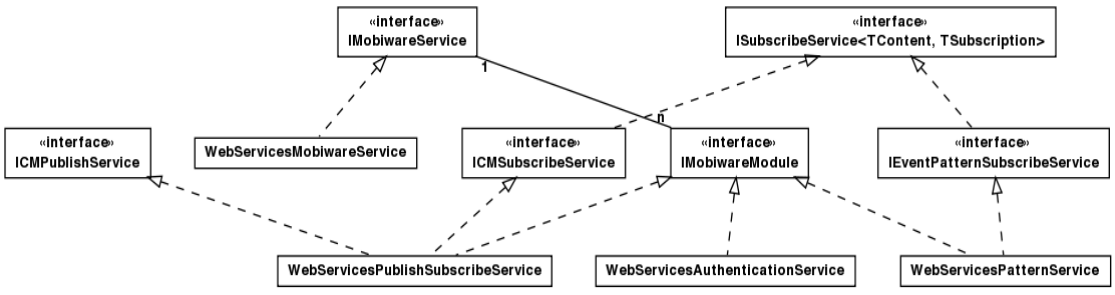


Figura 12. Esquema de las interfaces proporcionadas por el middeware

Mobiware-xmlBlaster

Este componente contiene las implementaciones de ICMSubscribeService y de ICMPublishService relacionadas con el módulo de gestión de contenidos, para la publicación, subscripción y anulación de subscripciones a diversos tópicos.

Se ha seguido la filosofía de publish/subscribe para el intercambio de mensajes entre clientes, y se ha utilizado xmlBlaster como middleware para la gestión de estos.

La arquitectura básica de un sistema publish/subscribe está basada en el concepto de tópico (tema, o conjunto de palabras clave) al que los usuarios (subscribers) pueden suscribirse, y del que serán informados cada vez que se realice una publicación referente a ese tópico. Así pues, los generadores de información de estos tópicos (publishers) publican los mensajes, que serán entregados de forma asíncrona en función de las características del sistema. Finalmente, el/los servidores de mensajes son los encargados de gestionar las suscripciones, la cola de mensajes y demás características.

La elección de xmlBlaster como sistema de mensajería ha permitido la abstracción de tareas relacionadas con la transmisión y gestión de los mensajes, todo ello una de forma fiable y asíncrona. XmlBlaster es un MOM (Message Oriented Middleware) y, de una manera simplificada y sencilla, puede considerarse que estos tipos de sistemas gestionan los mensajes de forma similar a cómo un DBMS (Data Base Management System) gestionaría la persistencia de los datos.

Resultado final

Tras la especificación estructural del middleware, a continuación, se describe el modo de funcionamiento del mismo. Para ello, se ha considerado oportuno dividir esta explicación en función de los servicios y módulos que proporciona. Finalmente, se propone un ejemplo de uso para facilitar la comprensión de todo lo detallado y poder tener así una visión global del funcionamiento del sistema.

Configuración de Mobiware

Para inicializar el middleware, basta con realizar una llamada a la clase estática createInstance de MobiwareFactory, pasando como parámetro un mapa con la configuración deseada. En la definición de este mapa de configuración, deben indicarse los valores de las siguientes constantes:

- Constants.REFRESH_TIME: tiempo de refresco/chequeo entre peticiones de nuevos mensajes en el servidor.

- Constants.WS_END_POINT: (en función del mecanismo de comunicación a utilizar con el servidor) valor de la URL utilizada por la WSDL.

- Constants.CLASS_CONVERTER_MODULE: (en función del mecanismo de comunicación a utilizar con el servidor). En caso de utilizar la implementación basada en Axis propuesta, este valor será la ruta del fichero xml en el que se definen los convertidores utilizados para transformar las clases Axis (utilizadas por el stub) en clases entendibles por los módulos del servicio.

A continuación, se muestra un ejemplo.

Inicialización del sistema:

```
Map<String, String> moduleConfiguration = new HashMap<String, String>();
moduleConfiguration.put(Constants.REFRESH_TIME, "3000");
moduleConfiguration.put(
      Constants.WS_ENDPOINT, "http://localhost:8080/mobiware-ws/ws");
moduleConfiguration.put(Constants.CLASS_CONVERTER_MODULE,
      "es/iti/mobiware/wsclient/axis/dozer/dozer-mapping.xml");

IMobiwareService mobiwareService =
      MobiwareServiceFactory.createInstance(moduleConfiguration);
```

No obstante, la configuración de Mobiware puede realizarse en función de los servicios del middleware de los que se van a hacer uso. Hay que tener en cuenta que es imprescindible la configuración del servicio de autenticación (imprescindible para que un usuario pueda iniciar sesión en el sistema y comunicarse con él), del servicio de gestión/chequeo de mensajes (necesario para realizar peticiones de nuevos mensajes al servidor) y, especialmente, del módulo de comunicación con el servidor (bridge). En este último caso, el cliente puede utilizar la implementación que el propio middleware proporciona, basada en Axis, o bien, utilizar una implementación propia. Lo mismo sucede para el servicio de sensorización: el cliente debe implementar su propio mecanismo de gestión de datos, dependiendo de sus sensores. A modo de ejemplo, se proporciona una implementación de prueba, en Java, de la interfaz que implementa este servicio.

Además de la configuración de los módulos a utilizar, se deben registrar las factorías que se vayan a utilizar, así como otros parámetros (por ejemplo, el tiempo de chequeo/refresco entre peticiones de mensajes al servidor).

A continuación, se muestran los pasos a seguir para la configuración del middleware (puede verse un ejemplo detallado, con el código a utilizar, en los archivos MobiwareServiceFactory.java y MobiwareApplication.java del paquete de testing):

- Configuración/registro de las factorías.
- Configuración del bridge. Implementa el mecanismo de comunicación con el servidor, en este caso, está basado en los stubs que proporciona Axis.
- Configuración de los servicios de:
  - Autenticación.
  - Sensorización.
  - Subscripción a eventos de patrones.
  - Subscripción a temas.
- Configuración del módulo de gestión de mensajes.
- Inicialización de Mobiware:
  - Inicialización de los módulos.
  - Inicialización del servicio (middleware):

La relación entre todos estos módulos, servicios, bridges y paquetes, puede verse en el siguiente esquema. Cada cubo representa una máquina conectada a la red con algún componente desplegado en el middleware. Los componentes propios del middleware están sombreados. El resto de componentes son externos e interactúan con el sistema.

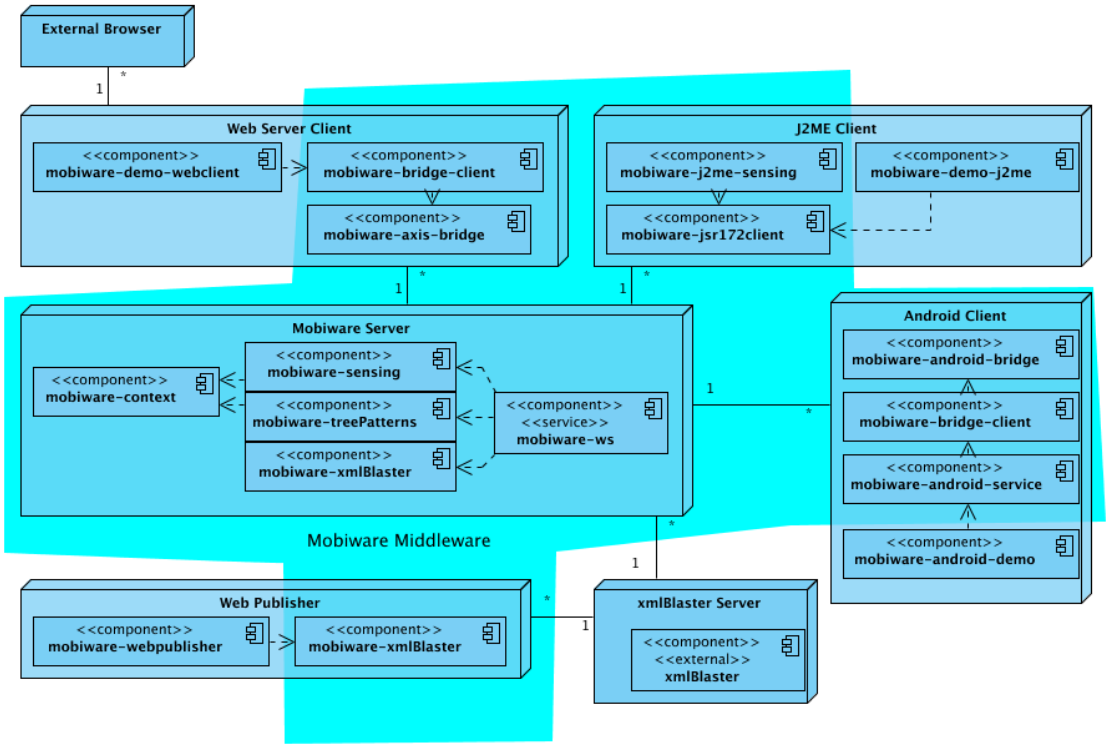


Figura 13. Estructura interna e interacción del Middleware con otras aplicaciones.

Módulo de autenticación

El funcionamiento del módulo de autenticación es trivial. Una vez configurado e inicializado Mobiware, el cliente sólo tiene registrarse en el sistema, utilizando sus credenciales (nombre de usuario, contraseña e identificador de la aplicación a utilizar). Para ello, debe utilizar el método de inicio de sesión definido en este módulo.

La implementación de ejecución de este método conlleva las siguientes acciones:

- Acceso al bridge de autenticación que será el encargado de enviar los datos del usuario a través del servicio web.

- Llegada la petición al endpoint de autenticación, éste realizará la decodificación de los datos recibidos, los enviará al mecanismo de autenticación y, una vez validados, almacenará en la sesión las credenciales. A partir de este momento, el interceptor de peticiones remotas sobre los endpoints, podrá validar la sesión de este usuario, por lo que habrá luz verde para el intercambio de datos entre ambos.

- Tanto si la validación del usuario falla como si no, se comunica al usuario el resultado de la validación en la respuesta a la petición de inicio de sesión.

El proceso es similar con el mecanismo de cierre de sesión.

Para simplificar las posteriores descripciones, a partir de este momento, se va a suponer que los usuarios inician sesión correctamente en el sistema.

Módulo de sensorización

Antes de iniciar el envío de datos de sensorización, el dispositivo debe inicializar sus sensores en el sistema, en función de la aplicación a la que desea conectarse. Para ello, independientemente de la implementación que vaya a emplear, debe utilizar la interfaz del servicio de sensorización.

Cuando una petición llega al servidor, el endpoint de inicio de la sensorización decodifica la información recibida, acerca de los sensores que el dispositivo tiene disponibles. A continuación, se la envía al gestor de sensorización asociado a la aplicación que el dispositivo a indicado cuando ha iniciado sesión. Internamente, el gestor envía al configurador de dispositivos los datos recibidos y configura el contenedor de sensorización a partir de la información de los sensores que ha obtenido en la respuesta del configurador. Antes de enviar esta respuesta al endpoint para que el dispositivo cliente pueda configurar sus sensores, el gestor activa el dispositivo y dispondrá de un listener que se mantendrá a la espera de recibir los datos de sensorización.

Para crear la respuesta de configuración de los sensores enviados por el dispositivo remoto, el configurador de dispositivos realiza un matching entre la información recibida (driver de sensor, modelo, etc.) y la configurada almacenada para la aplicación en el servidor (alias o nombres canónicos, limitadores, etc.) Para ello, éste se apoya en un gestor de nombres canónicos, que es el encargado de realizar este matching.

A partir de este momento, el dispositivo remoto puede empezar a enviar periódicamente los datos de sensorización.

Cuando una petición llega al endpoint que gestiona la recepción de los datos, tras su decodificación, éste los suministra al gestor de sensorización correspondiente. El gestor los almacenará en el contenedor de sensorización que dispone y lanzará un evento de sensorización. Este evento se utiliza para avisar al motor de contextos de que se ha recibido nueva sensorización y, por tanto, que ha habido un cambio en un dispositivo activo en el sistema y asociado a una entidad. Todo esto hará que el motor de contextos reevalúe todos los patrones en los que el tipo esta entidad esté involucrado.

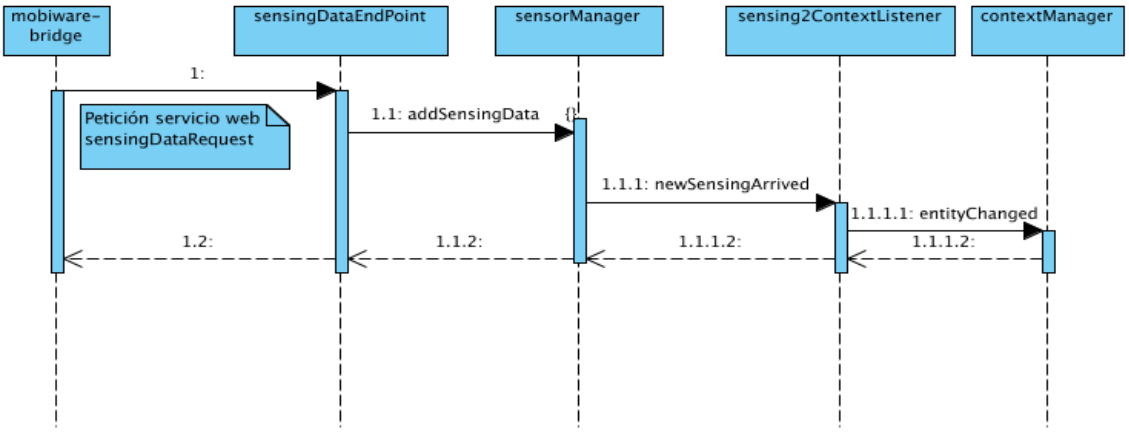


Figura 14. Propagación de la sensorización.

Módulo de gestión de contextos

Para definir el funcionamiento de este módulo, deben tenerse en cuenta tanto las acciones que se realizan en el lado del cliente, como las del lado del servidor. Así pues, para describir estas acciones, se va a suponer el ejemplo de uso descrito anteriormente (ver Ejemplo de uso del módulo de gestión de contextos).

En el cliente, cuando el usuario instancia el sistema de Mobiware, de manera automática y totalmente transparente para él, se crea un hilo que, periódicamente, va realizando peticiones al servidor para comprobar si existen nuevos mensajes relacionados con sus subscripciones.

De forma paralela, cuando se realiza una petición utilizando el servicio de suscripción a eventos el módulo encargado de procesar la petición, comprueba si el usuario, efectivamente, no ha realizado con anterioridad una suscripción con las mismas características. En caso afirmativo, se delega al bridge de suscripción la petición para que sea enviada, a través del servicio web, al servidor.

Las acciones que se suceden en el servidor, una vez la petición llega al endpoint correspondiente, son las siguientes:

- El endpoint decodifica los datos recibidos a través del servicio web y se los pasa al módulo encargado de la gestión de patrones (treePatterns) para su conversión en un objeto procesable por el motor de contextos.

- El servicio que gestiona los patrones para su inclusión en el contexto (ContextManagerPatternService), utiliza el manejador de patrones (ContextPatternHandler), y éste a su vez la factoría de Mobiware (IMobiwareFactory) para, de manera recursiva, recorrer el árbol que forma la expresión del patrón y convertirlo, junto con sus metadatos, en el objeto (ImeasurablePattern) que se utilizará para la evaluación.

- Es posible que, tras su transformación, al patrón se le apliquen algunos post-procesadores (IMeasurablePatternPostProcessor), en función de los operandos que contenga.

- Este servicio de gestión, antes de enviar el evento de suscripción, envía el nuevo patrón generado para su inclusión en la gestión de contextos. Como respuesta, recibe un identificador que asociará al nuevo patrón. Todo ello, volverá a ser enviado al motor de contextos. Por otro lado, el endpoint desde el que fue invocado también recibe este identificador, para que se incluya en la respuesta al módulo de gestión de suscripciones a través del servicio web.

- En la parte cliente del middleware, el identificador recibido será utilizado para mantener la lista de eventos de patrón a los que está suscrito el usuario.
    - Una vez el evento de suscripción llega al motor de contextos (IContextManager) se comprueba si existe algún listener, dado de alta previamente y que esté asociado a él.

- Si no existe, se crea un nuevo conjunto de listeners y, en ambos casos, se asocia un nuevo listener al evento.

- Cuando se produce un cambio en una entidad dada de alta en el sistema, se reevalúan todos los patrones que tienen asociado el tipo de esta entidad. En el caso de que se detecte la creación de un nuevo contexto, se notifica este cambio al conjunto de listeners asociados al evento.

- Por ejemplo, considerando la suscripción al evento de patrón descrito anteriormente “pulsaciones_del_usuario_xyyy > 120” se considera que se ha creado un nuevo contexto cuando la condición pasa a ser cierta.

- Cada uno de estos listeners, obtendrá las entidades involucradas en el patrón y notificará a su vez, de la creación de este nuevo contexto (enviando el identificador del patrón y el tipo de evento que lo ha producido). Como consecuencia, se almacenará en el buzón de mensajes del servidor un nuevo mensaje, cuya causa contendrá tanto el identificador del patrón, como el evento que se ha producido (con su tipo y subtipo).
    - Para gestionar los mensajes de forma genérica en el servidor (independientemente de si están asociados con la suscripción a eventos de

patrón o con la suscripción a temas), se han creado unos manejadores capaces de procesar estos mensajes en función del su tipo.

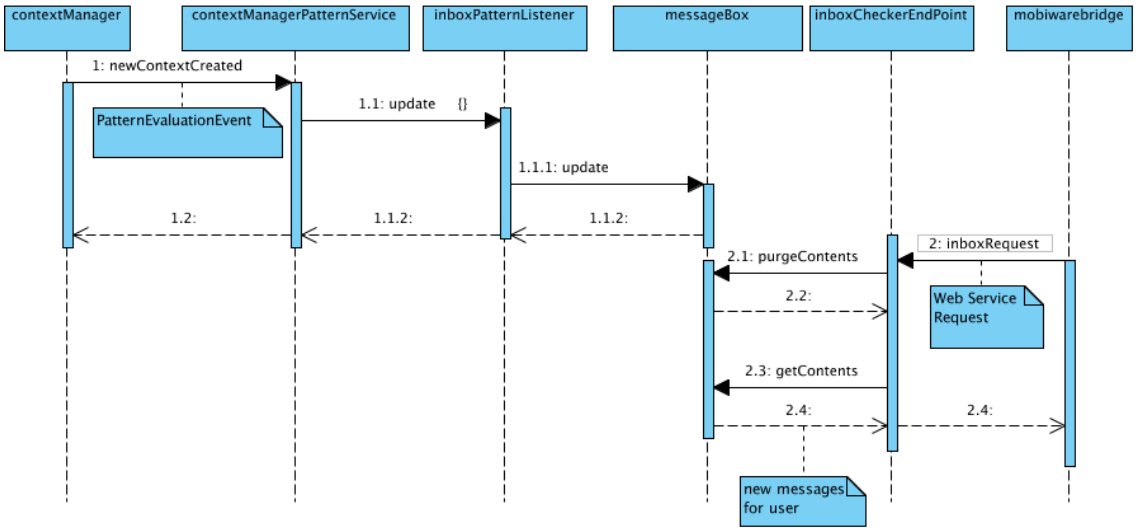


Figura 15. Propagación de un nuevo contexto.

Módulo de gestión de contenidos

Al igual que ocurre con el módulo de gestión de contextos, cuando un usuario envía la suscripción de un tema a través del middleware, el endpoint correspondiente es el encargado de decodificar los datos recibidos y de enviárselos, en este caso, al gestor de contenidos. Tal y como se ha comentado anteriormente, para la gestión de contenidos se utiliza xmlBlaster como backend.

De este modo, el servicio de suscripción (XmlBlasterSubscribeService) encapsula una conexión al backend con sus propias suscripciones particulares. Las traducciones que realiza son las siguientes:

- Cuando se recibe una subscripción a un ITopic, éste se traduce a un SubscribeKey de xmlBlaster.
- Cuando se recibe una petición de eliminación de una subscripción a un ITopic, éste se traduce a un UnSubscribeKey.
- Cuando xmlBlaster reparte un nuevo contenido en forma de UpdateKey y UpdateQos, éstos se traducen en una nueva instancia de IContent.

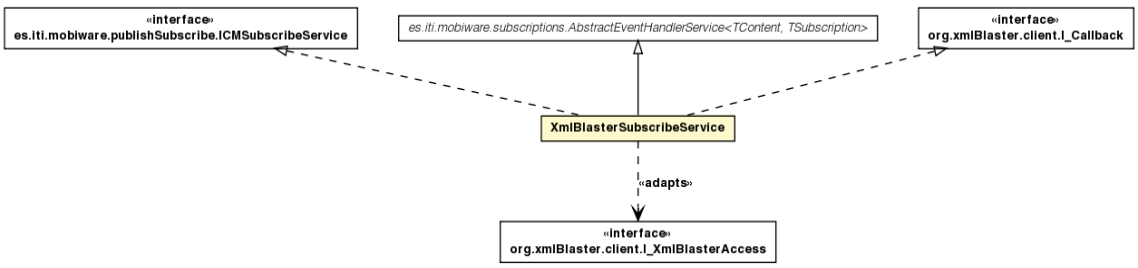


Figura 16. Abstracción de la subscripción en el backend de mensajería.

En el caso de las publicaciones, la interfaz IPublishService se encarga de encapsular las acciones pertinentes en el backend. El servicio de publicación (XmlblasterPublishService) realiza las siguientes traducciones:

- Cuando llega una petición de publicación de un IContent, éste se traduce a un PublishKey y a un PublishQos.

- Cuando el backend proporciona una lista de contenidos MsgUnit, ésta se traduce a una lista de IContent.

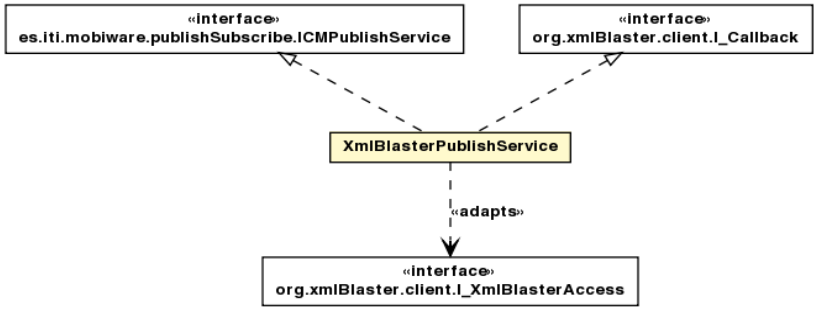


Figura 17. Abstracción de la subscripción en el backend de mensajería.

Aplicación práctica

Comprender el funcionamiento de este módulo es sencillo si, a modo de ejemplo, se toma alguno de los dos últimos escenarios descritos en el módulo de Mobiware-context. Así pues, se va a considerar el segundo de ellos (escenario 2 sobre la monitorización de pacientes) para describir tanto el comportamiento, como los pasos que se van siguiendo en la gestión de contextos.

El mecanismo utilizado para realizar una petición de suscripción (o anulación de suscripción) a un evento varía en función de si el contexto ya se ha definido previamente o no (es decir, depende de si el patrón que lo representa está, o no, registrado en el motor de contextos).

Ya que en Mobiware los contextos se definen por medio de patrones, y las suscripciones se realizan a partir de eventos sobre ellos, se va a suponer el siguiente caso de uso:

Un usuario (A) desea conocer cuándo el ritmo cardíaco de un paciente (usuario B) supera las 100 pulsaciones por minuto.

Módulo de gestión de contextos

Las acciones que deben realizar ambos usuarios varían en función de cada uno de ellos.

El usuario A, debe registrarse en el sistema y, mediante el servicio de suscripción a eventos de patrón, realizar una petición de suscripción. Para ello, debe expresar el contexto anterior, a partir del patrón y la suscripción a los siguientes eventos:

- Declaración del sensor involucrado en la generación del contexto (en este caso, un pulsómetro). Además, en el atributo nParameter se debe indicar la referencia al tipo de entidad asociada al patrón (ya que el servidor utilizará este tipo para la

evaluación del patrón). Este valor de nParameter, indica a qué elemento del array de metadatos (definido posteriormente) se corresponde.

```
Expression pulsometer = new Expression("sensor");
pulsometer.setParameter("canonicalName", "pulsometer");
pulsometer.setParameter("nParameter", "0");
```

- Definición del valor máximo que se desea comprobar y que se utilizará en la evaluación del patrón en el servidor (en este caso, pulsaciones > 100 por minuto):

```
Expression upperThresholdValue = new Expression("Integer");
upperThresholdValue.setParameter("value", "100");

Expression upperThresholdRange =
        new Expression(">", pulsometer, upperThresholdValue);
```

- Definición de la procedencia de los valores de sensorización (en ese caso, se va a suponer que el nombre del usuario B es xyyy).

```
Expression belongsTo = new Expression("belongsTo");
belongsTo.setParameter("nParameter", "0");
belongsTo.setParameter("value", "xyyy");

Expression isOtherPulsometer =
        new Expression("AND", pulsometerRange, belongsTo);
```

- Definición del tipo de entidad involucrada en los metadatos del patrón (en este caso, el tipo de entidad es “dispositivo”).

```
ParameterMetadata[] metadata = new ParameterMetadata[1];
metadata[0] = getEntityMetadata(Constants.SENSING_ENTITY_TYPE);
```

- Definición del patrón (compuesto por la expresión y sus metadatos):

```
IPattern pattern = new PatternImpl();
pattern.setExpression(isOtherPulsometer);
pattern.setMetadata(metadata);
```

- Definición del evento de suscripción. Como el usuario desea que se le notifique cuando el paciente supera las pulsaciones, se va a suponer que sólo desea saber cuándo el patrón pasa de ser falso a ser cierto. Así pues, el evento se debe definir del siguiente modo:

```
IEventPatternType event =
```

```
        new EventPatternType(PatternEvaluationEventType.CHANGE,
                    PatternEvaluationEventSubtype.TRUE);
```

- Si, además, el usuario deseara durante cuánto tiempo se da la condición expresada por el patrón, debería también suscribirse al siguiente evento:

```
IEventPatternType event2 =
        new EventPatternType(PatternEvaluationEventType.IS,
                    PatternEvaluationEventSubtype.TRUE);
```

- Para finalizar, el usuario debe añadir al servicio un listener asociado al evento al que acaba de suscribirse. Las notificaciones enviadas por el servidor, llegarán al usuario a través de este listener.

El servicio y la interfaz, así como los métodos que este usuario debe utilizar, son los siguientes:

- Interfaz: IEventPatternSubscribeService.
- Métodos:
  - **public** IMobiwareList<Integer> getSubscribedPatternsId() **throws** SubscribeException, SessionException, NotImplementedException;
  - **public** Integer subscribeToNewPattern(Integer patternId, IPattern pattern, IEventPatternType event)**throws** SessionException, SubscribeException;
  - **public** Integer subscribeToNewPattern(IPattern pattern, IEventPatternType event) **throws** SessionException, SubscribeException;
  - **public** Integer subscribeToNewPattern(Integer patternId, IPattern pattern, Set<IEventPatternType> event) **throws** SessionException, SubscribeException;
  - **public** Integer subscribeToNewPattern(IPattern pattern, Set<IEventPatternType> event) **throws** SessionException, SubscribeException;
  - **public void** setUpdateListeners(Set<IUpdateListener<TContent, TSubscription>> updateListeners);
  - **public void** addUpdateListener(IUpdateListener<TContent, TSubscription> listener);
  - **public void** addUpdateListener(TSubscription subscription, IUpdateListener<TContent, TSubscription> listener);
  - **public void** removeUpdateListener(IUpdateListener<TContent, TSubscription> listener);
  - **public void** removeUpdateListener(TSubscription subscription, IUpdateListener<TContent, TSubscription> listener);
  - **public** IMobiwareList<TSubscription> subscribe(IMobiwareList<TSubscription> events) **throws** SubscribeException, SessionException, NotImplementedException;
  - **public** IMobiwareList<TSubscription> unsubscribe(IMobiwareList<TSubscription> subscription) **throws** SubscribeException, SessionException, NotImplementedException;
  - **public** TSubscription subscribe(TSubscription event) **throws** SubscribeException, SessionException;
  - **public** TSubscription unsubscribe(TSubscription subscription) **throws** SubscribeException, SessionException, NotImplementedException;
  - **public** IMobiwareList<TSubscription> unsubscribeAll() **throws** SessionException, SubscribeException,NotImplementedException;

- **public** IMobiwareList<TSubscription> getSubscribedEvents();
- **public** String close();

Por otro lado, el usuario B, debe registrarse en el sistema e inicializar los sensores que tiene disponibles (se va a suponer que, entre ellos, un pulsómetro). Una vez obtenga la respuesta del servidor y configure sus sensores, a partir del servicio de sensorización, puede empezar a enviar los datos de sensorización a través del middleware.

El servicio y la interfaz, así como los métodos que este usuario debe utilizar, son los siguientes:

- Interfaz: ISensingService.
- Métodos:
  - **public void** startSendingSensingInformation() **throws** SessionException;
  - **public void** endSendingSensingInformation() **throws** MobiwareException

Las siguientes tablas muestran los operadores y tipos de valores soportados por el middleware. No obstante, es posible definir nuevos tipos y actuadores, para la definición de patrones y contextos, en los casos que fueran necesarios.

Valor	Descripción	Identificador	Parámetros
DOWValue	Day Of Week. Permite especificar un valor como un día determinado de la semana. Ejemplo: Lunes, Martes, Miércoles...	dow	Value
DateValue	Day Of Year. Permite especificar un valor como un día determinado del año, utilizando un formato para ello. Por defecto, se ha definido el DEFAULT_DATE_FORMAT = “dd/MM”. Ejemplo: 21/05	date	Value
TimeValue	Hora. Permite especificar un valor como una determinada hora, a partir de un determinado formato. Por defecto, se ha definido el DEFAULT_TIME_FORMAT="HH:mm".	time	Value
NowValue	Ahora. Hace referencia al momento actual (HH:mm)	now	
TodayDOWValue	Today Day Of Week. Hace referencia al día de la semana actual. Ejemplo: Si estamos a lunes, TodayDOWValue = “Lunes”	todayDow	
TodayValue	Día Actual. Hace referencia a hoy (es decir, el día actual)	today	
IntegerValue	Permite especificar un valor como número entero.	Integer	Value
FloatValue	Permite especificar un valor como número flotante.	Float	Value

Valor	Descripción	Identificador	Parámetros
DoubleValue	Permite especificar un valor como número real.	Double	Value
StringValue	Permite especificar un valor de tipo cadena.	String	Value

Tabla 3. Tipo de valores soportados por el middleware para la creación de contextos.

Operador (global)	Descripción	Identificador	Parámetros
AndOperator	Permite realizar una conjunción lógica entre dos expresiones (soportadas por Mobiware).	AND	left, right
OrOperator	Permite realizar una disyunción lógica entre dos expresiones (soportadas por Mobiware).	OR	left, right
DistanceOperator	Operador de distancia. Permite calcular la distancia entre dos puntos, definiendo cada punto como un par: latitud-longitud.	distance	latitude1, longitude1, latitude2, longitude2
ConcatenationOperator	Operador de concatenación. Permite concatenar dos o más cadenas.	+	left, right
MetadataOperand	Operando de acceso de metadatos. Permite utilizar como valor en las expresiones un dato que procede del mapa de metadatos de la expresión sobre la que se ha definido.	metadata	nParameter
StringComparator, MetadataOperand, StringValue	Comprueba si la entidad especificada por nParameter coincide con la sesión actual del usuario.	isMine	nParameter
StringComparator, MetadataOperand, StringValue	Comprueba en las credenciales de la entidad especificada por nParameter si ésta pertenece al usuario con nombre igual valor del parametro value	belongsTo	nParameter, value
NumericArithmeticOperator	Operador aritmético. Permite realizar operaciones aritméticas sobre dos o más valores.	+ - * /	left, right
NumericComparator	Operador numérico. Permite realizar comparaciones entre dos valores/expresiones numéricas.	= < > <= >=	left, right
StringComparator	Comparador de cadenas. Permite comparar dos o más cadenas.	= < > <= >=	left, right
StringExpressionMatcher	Matching entre cadenas. Permite comprobar si dos expresiones (de tipo String) son coincidentes.	RegMatch	value, pattern

Tabla 4. Tipo de operadores (de propósito general) soportados por el middleware para la creación de contextos.

Operador (específico)	Descripción	Identificador	Parámetros
CanonicalSensorOperand	Permite referenciar a un valor suministrado por un determinado sensor. Por ejemplo: latitud, longitud, velocidad...	sensor	nParameter

Tabla 5. Tipo de operadores (de propósito específico) soportados por el middleware para la creación de contextos.

Módulo de gestión de contenidos

Un usuario que desea suscribirse a uno o varios temas y recibir información de ellos, debe registrarse en el sistema y realizar las siguientes acciones:

- Instanciar el servicio de suscripción a temas (ISubscribeService).
- Configurar en el servicio algún listener para recibir los contenidos a las suscripciones.
- Solicitar al servidor la lista de temas disponibles para la suscripción.
- Seleccionar uno de esos temas (ITopic) y utilizar el servicio anteriormente instanciado para su suscripción.

Cuando el servidor detecta un nuevo contenido que repartir asociado a alguna suscripción activa, lo empaqueta en el contenido (IContent) de un mensaje y se lo pasa a los listeners configurados en el servicio junto con una referencia al tema (ITopic) suscrito que ha ocasionado el reparto.

El servicio de suscripción y la interfaz, así como los métodos que el usuario debe utilizar, son los siguientes:

- Interfaz: ICMSubscribeService.
- Métodos:
  - **void** publish(ICMContent content)**throws** PublishException, SessionException;
  - **public** ImobiwareList<IContent> getContents(String topic) **throws** PublishException;
  - **public void** remove(String path) **throws** PublishException;
  - **public** String close() ;

Con el objetivo de facilitar la comprensión del diseño y funcionamiento del middleware, en la sección de anexos, se han incluido algunos esquemas sobre los objetos creados y las dependencias entre ellos. En concreto, se muestran dos diagramas de clases, uno sobre la gestión de la información de sensorización en este módulo, y otro sobre la distribución de la información en el motor de contextos. Además, se incluye el esquema relacional de la base de datos para almacenar la configuración de los sensores y limitadores de las aplicaciones context-aware que hacen uso del middleware.

Conclusiones

Mobiware es el resultado de dos años de investigación y desarrollo. Actualmente, se encuentra en una versión estable y su aplicación está orientada al envío de sensorización por parte de dispositivos móviles.

No existe un único modelo de contexto, ni el modelo mejor de todos, sino que todo depende de los requisitos existentes y de los recursos disponibles. Si bien unos modelos son más expresivos que otros, puede suceder que, para una aplicación que se vaya a desarrollar, el tema de la expresividad no sea un factor clave. Sin embargo, puede resultar más interesante disminuir al máximo el tiempo de procesamiento del motor de inferencia, y disponer de respuestas en tiempo real que ayuden en la toma decisiones y permitan actuar en consecuencia. Éste podría ser el caso de un modelo basado en reglas lógicas para una aplicación agrícola. El simple hecho de disponer de unas sencillas reglas podría bastar para la correcta gestión del campo: control del PH, humedad, actuación ante plagas, etc.

Tal y como se ha comentado a lo largo de este documento, ésta ha sido la razón por la que, en el desarrollo del middleware, se ha optado por utilizar un modelo basado en expresiones lógicas: apostamos por una manera sencilla de representar la información y creemos en la importancia de aportar una respuesta lo más rápidamente posible. No obstante, el diseño modular de su arquitectura permite el desacoplamiento entre sus módulos. Esto hace que sea posible reemplazar un módulo por otro en caso necesario, dependiendo de las nuevas aplicaciones a desarrollar, y que hacen uso del middleware.

Un ejemplo de este hecho, sería la sustitución del módulo de inferencia de nuevo conocimiento por otro más complejo, capaz de aportar mayor expresividad a la hora de describir situaciones y generar contextos. De este modo, podría optarse por una solución híbrida, basada en la combinación de un modelo ontológico y otro creado a partir lógica de primer orden, tal y como se describe en [WZT+01]. Así pues, podría utilizarse OWL+DL para la representación formal de la información de contexto y, al igual que en el modelo ontológico de CONON, distinguir dos niveles: un nivel superior, formado por un conjunto de entidades o conceptos generales; y un nivel inferior, más específico, que permitiría añadir nuevas entidades o conceptos, de manera jerárquica, en función del dominio de aplicación (es decir, se trataría de un mecanismo de extensión del nivel superior).

Por otro lado, podría aumentarse la capacidad de generar contextos del middleware, si se expandiera el ámbito de aplicación del middleware y se modificara el módulo de sensorización. Así pues, podría ampliarse su funcionalidad si, además de la sensorización procedente de dispositivos móviles, fuera capaz de almacenar y gestionar información procedente de sensores incrustados en el propio entorno, tal y como se detalla, a continuación, en el apartado de Extensiones futuras.

Aunque en la actual versión del middleware se ha optado por utilizar XmlBlaster como *backend* para la gestión de los mensajes, al igual que sucede con el resto de módulos, el componente de gestión de contextos podría sustituirse por otro que implementara una solución propia, teniendo en cuenta la filosofía *publish/subscribe* que se ha seguido a lo largo de todo el proyecto. No obstante, debida a la baja complejidad a la hora de su implantación, y a la funcionalidad que ofrece (publicación de nuevos temas, suscripciones, anulación de suscripciones, etc.) se ha considerado que XmlBlaster cumple con los requisitos existentes y, por el momento, es la mejor a escoger ya que se sitúa ante una buena posición con respecto a la relación esfuerzo-resultado.

Respecto al rendimiento del sistema, éste podría mejorarse si en lugar de un único servidor central, se utilizara una arquitectura distribuida en diversos nodos, capaces de trabajar de manera concurrente. De esta forma, un contexto podría subdividirse en diversas partes para que cada una de ellas pudiera ser procesada, en paralelo, por nodos distintos. Si a esto le sumásemos un balanceador de carga, que distribuyera las peticiones al motor de inferencia en

tiempo real y en función de la carga de cada uno de los nodos, es fácil suponer que los tiempos de respuesta del módulo de gestión del contexto disminuirían considerablemente.

Para finalizar, considero que Mobiware es el resultado de un esfuerzo conjunto por ofrecer una solución general y sencilla dentro del abstracto mundo de la computación ubicua. No hay que olvidar que se trata de un middleware, y no de una aplicación, por lo que he de hacer hincapié en que su propósito es el de facilitar la comunicación entre dispositivos de distinta naturaleza y aplicaciones cliente. Por esta razón, se deja en manos de los desarrolladores de las aplicaciones, la posibilidad de implementar y hacer uso de sus propios módulos, más específicos, en los casos en los que sea necesario.

Centrada en un ambiente más técnico, la ejecución de este proyecto me ha permitido adentrarme en el campo de la investigación y conocer más acerca de conceptos como el Internet de las cosas, la computación ubicua y los sistemas sensibles al contexto. Además, a la hora del diseño e implementación del middleware, he podido poner en práctica los conocimientos adquiridos en asignaturas como: Sistemas de Información Ubicuos (SIU), Sistemas de Inteligencia Ambiental (SIA), Modelado y Desarrollo de Servicios Web (MSW), Gestión de Modelos (GMO), Extracción Automática de Conocimiento en Bases de Datos e Ingeniería del Software (EAC), Calidad de Sistema de la Información (CSI) etc.

Debido a que el proyecto se ha ejecutado en un instituto tecnológico (cuyo fin es el de promover la transferencia tecnológica) aunque todavía no se ha realizado ninguna publicación científica sobre el tema, sí que ha sido divulgado en artículos de prensa escrita y en presentaciones en eventos de difusión con carácter comercial. Por ejemplo, se realizó una ponencia invitada en el Mobile World Congress de 2011 (puede verse la ficha técnica del proyecto en los apéndices almacenados junto a este documento). Del mismo modo, actualmente, se está preparando una descripción de la arquitectura para enviarla a un taller del congreso IEEE Pervasive Computing and Communication (PerCom) 2012 (9th IEEE Workshop non Context Modeling an Reasoning – CoMoRea).

Por otro lado, el Área de Calidad del propio Instituto está realizando un conjunto de pruebas de escalabilidad de la arquitectura, cuyo resultado se intentará publicar en un congreso de carácter científico de alta calidad o revista indexada.

Extensiones futuras

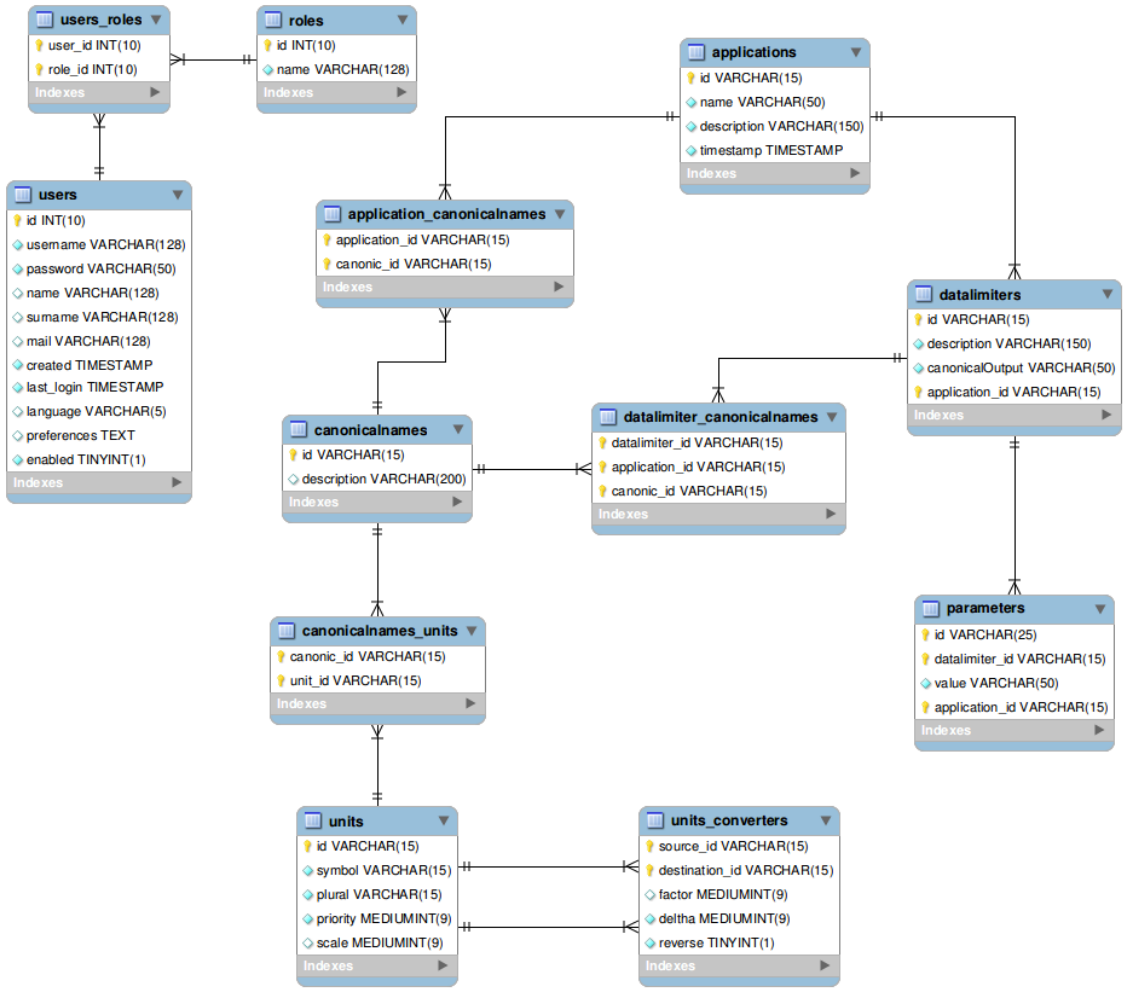
En la actualidad, se está llevando a cabo la ampliación del middleware para extender su dominio de aplicación a entornos inteligentes. Fruto de este trabajo, ha nacido el proyecto Smartware (Soporte Middleware para entornos inteligentes), una extensión de Mobiware en la que se está haciendo hincapié en la mejora de los módulos de sensorización y gestión de contextos.

El objetivo es que Smartware sea capaz de gestionar información procedente tanto de dispositivos móviles, como de redes de sensores distribuidas o incrustadas a lo largo de grandes superficies. Así pues, se centra en las siguientes cuestiones:

- Sensorización: configuración, programación y despliegue. Conjunto de nodos distribuidos, que disponen de elementos captadores (sensores y receptores multimedia) y actuadores.
- Paradigmas de comunicación y recogida de datos. Transmisión y procesamiento de la información recogida.
- Modelado de contexto. Partiendo ya de una infraestructura funcional de recogida de datos, el siguiente paso es procesar toda la información y utilizarla para razonar sobre el contexto de los usuarios, del entorno u otros contextos ambientales. Para este modelado de contexto se desarrollará un soporte genérico y configurable.

Además del cliente de sensorización Android, actualmente, se está trabajando en una versión para IPhone y otra para plataformas .NET.

Anexos



Esquema 1. Configuración de los sensores y limitadores de las aplicaciones en Mobiware.

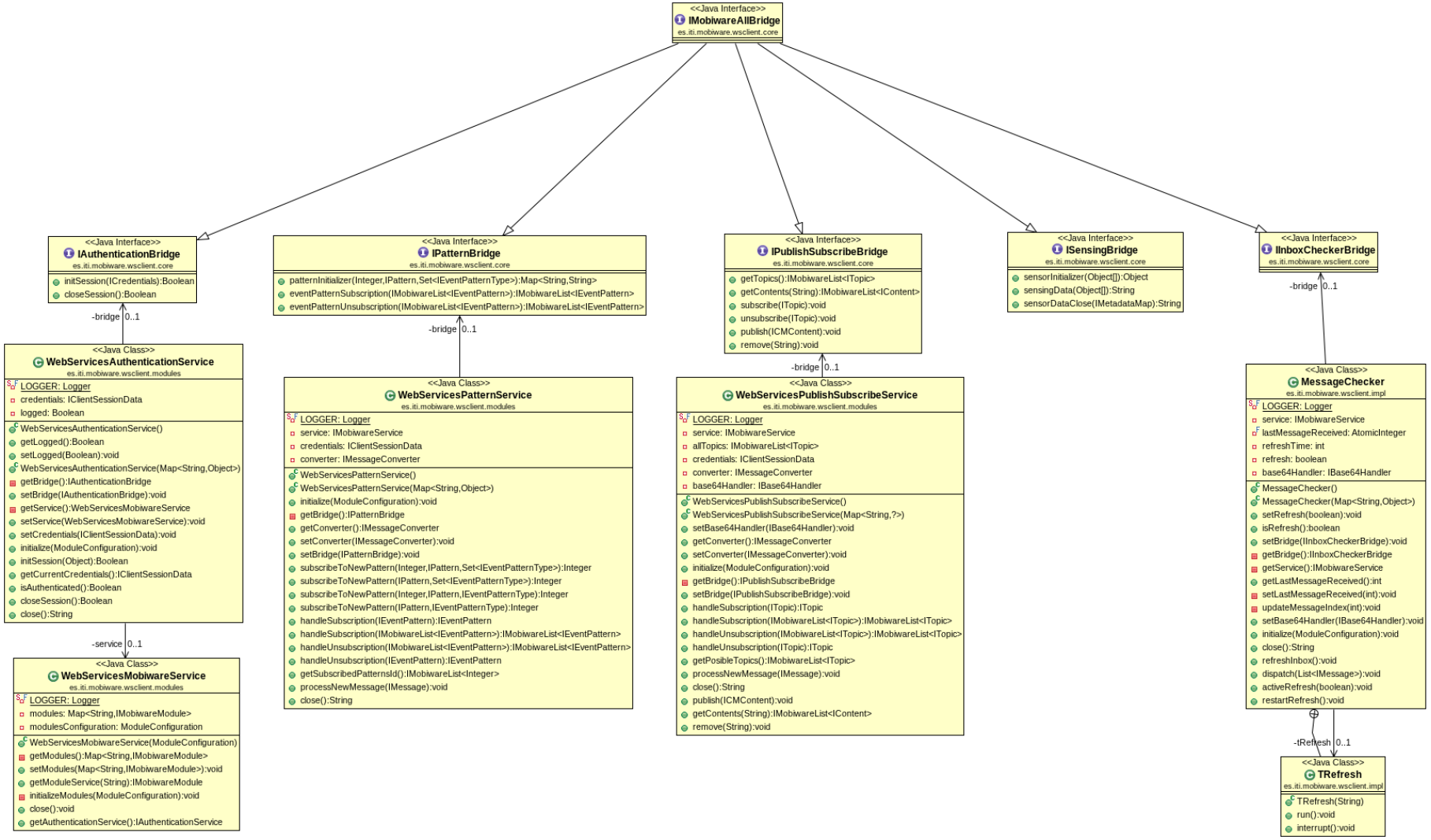


Diagrama de clases 1. Implementación de los servicios ofrecidos por Mobiware.

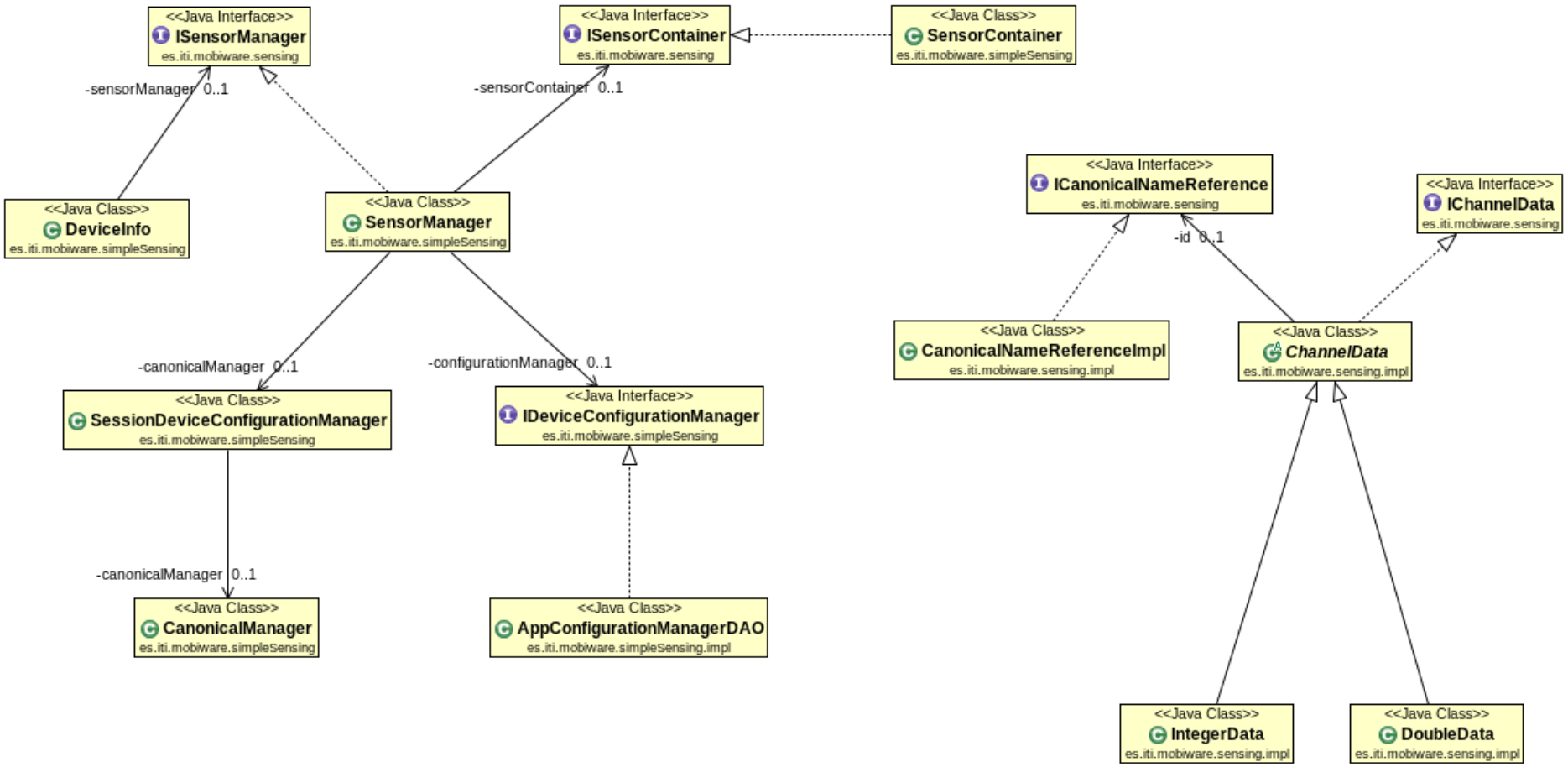


Diagrama de clases 2. Almacenamiento de la información de sensorización.

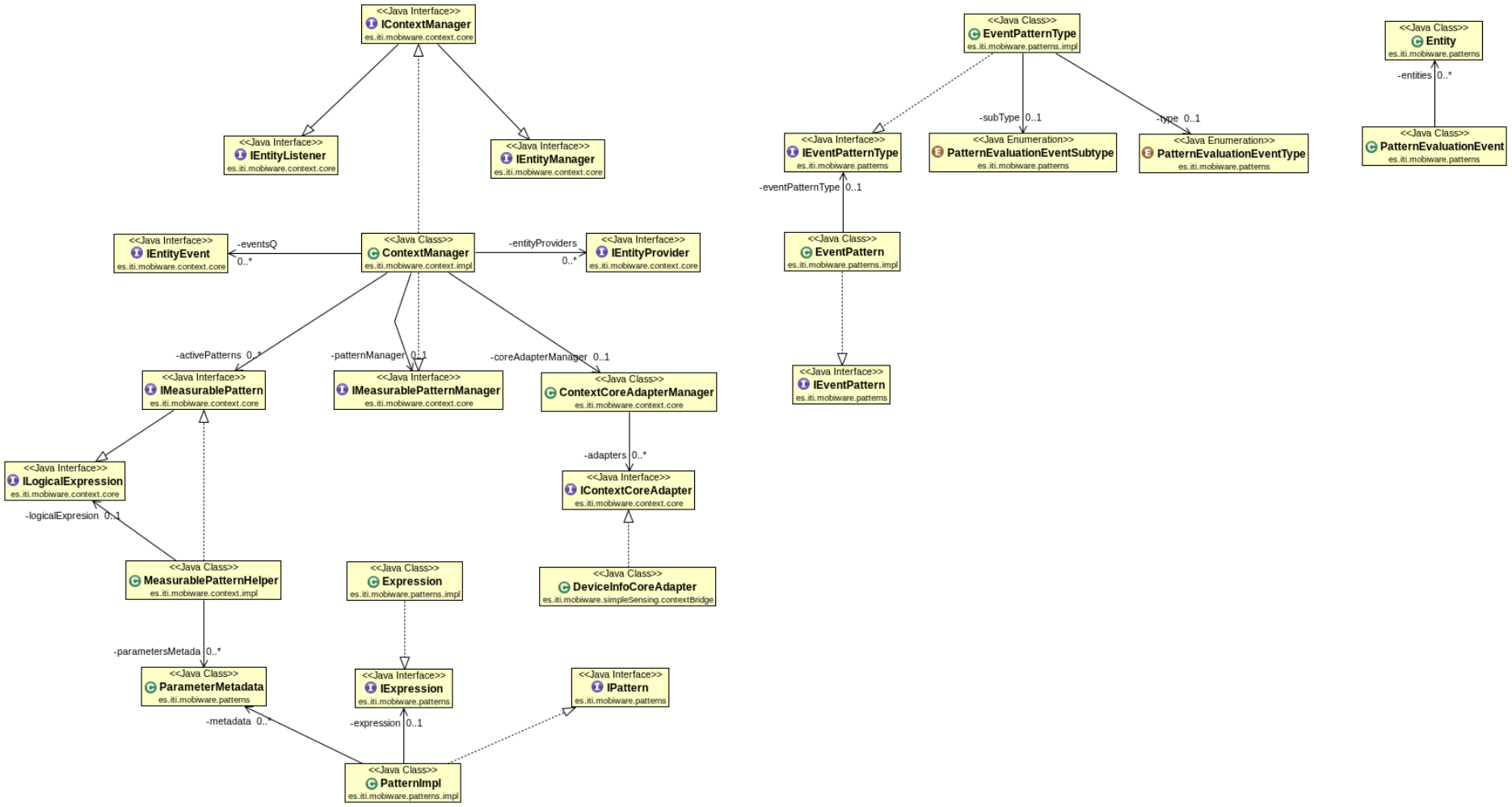


Diagrama de clases 3. Distribución del motor de contextos.

Bibliografía

[ATH07] Context awareness in mobile computing environments. Christos B. Anagnostopoulos, Athanasios Tsounis, Stathes Hadjiefthymiades. Wireless Personal Communications 42(3). Páginas 445-464. 2007.

[BB09] Bessi M., Bruni L. Context-aware middleware. (http://www.slideshare.net/LeoBruni/a-survey-about-contextaware-middleware). 2009.

[BBH+10] A survey of context modelling and reasoning techiniques. Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Arnand Ranganathan, Daniele Riboni. Pervasive and Mobile Computing 6. Elsevier. Páginas 161-180. 2010.

[BBR02] Bauer M., Becker C., Rothermel K.: Location Models from the Perspective of Context-Aware Applications and Mobile Ad Hoc Networks. Personal and Ubiquitous Computing Volume 6, Issue 5-6 (December 2002) Pages: 322 – 328, 2002.

[BC04] Biegel g., Cahill V. A framework for developing mobile, context-aware applications. Percom 00:361. 2004.

[BDR07] Baldauf, Matthias; Dustdar, Schahram & Rosenberg, Florian: *A Survey on Context-Aware Systems.* International Journal of Ad Hoc and Ubiquitous Computing, Vol. 2, Nr. 4, S. 263-277. 2007.

[Bro98] Brown, P.J., The Stick-e Document: a Framework for Creating Context-Aware Applications. Proceedings of the Electronic Publishing, 1996. 8(2): p. 259-272.

[CK00] Guanling Chen, David Kotz. A survey of context-aware mobile computing research. Tech. Rep. TR2000-381, Dartmouth, November 2000.

[CKG05] Christopoulou E., Goumopoulos C., Kameas A. An ontoloy-based context management and reasoning process for ubicomp applications. Proceedings of the 2005 joint conference on Smart objetvs and ambiental intelligence, 265-270. 2005.

[CRE05] Couto R., da Rocha A., Endler M. Evolutionary and efficient context management in heterogeneous environments. Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing. 1-7. New York, USA. 2005.

[CFJ03] Chen, H., Finin, T. and Joshi: A. An Ontology for Context-Aware Pervasive Computing Environments. Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 2003.

[E07] Ellebaek Kjaer. A survey of context-aware middleware. Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering. 2007.

[ESL07] Ejigu D, Scuturici M., Lionel B: CoCA: A Collaborative Context-Aware Service Platform for Pervasive Computing. IEEE/CS International Conference on Information Technology: New Generations, ITNG 2007, Las Vegas, Nevada, USA.

[GBH+05] Efficiently Managing Context Information for Large-Scale Scenarios. M. Grossmann, M. Bauer, N. Honle, U. Kappeler, D. Nicklas, T. Schwarz. Proceedings of Pervasive Computing and Communications. IEEE Computer Society. 2005.

[GSB02] Multi-sensor context-awareness in mobile devices and smart artifacts. Hans W. Gellersen, Albercht Schmidt, Michael Beigl. Mobile Networks Applications 7(5). Páginas 341-351. 2002.

[H01] Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design. T.A. Halpin. San Francisco. 2001.

[HI06] Developing context-aware pervasive computing applications: Models and approach. K. Henricksen, J. Indulska. Pervasive and Mobile Computing 2(1). Páginas 37-64. 2006.

[HLJ04] Towards a hybrid approach to context modelling, reasoning and interoperation . Karen Henricksen, Steven Livingstone, Jadwiga Indulska . Proceedings of the First International Workshop on Advanced Context Modelling, Reasoning and Management, University of Southampton, Nottingham, England, 2004.

[HNC+07] Modelado de Contexto: Una Ontología Adaptativa al Usuario en Ambientes Inteligentes. Ramón Hervás, Salvador W. Nava, Gabriel Chavira, José Bravo. Informe técnico. 2007.

[HPH03] From SHIQ and RDF to OWL: The making of a web Ontology language. I. Horrocks, P.F. Patel-Schneider, F. van Harmelen. Journal of Web Semantics 1. Páginas 7-26. 2003.

[JLR+07] Jacob C., Linner D., Radusch I., Steglich S. Loosely coupled and context-aware service provision incorporating the quality of rules. Proceedings of the 2007 International Conference on Internet Computing. CSREA Press. 2007.

[KPT+09] Context-aware service engineering: A survey. Georgia M. Kapitsaki, George N. Prezerakos, Nikolaos D. Tselikas, Iakovos S. Venieris. The Journal of Systems and Software 82. Elsevier. 1285-1297, 2009.

[Lok07] Loke, S. Context-Aware Pervasive Systems: Architectures for a New Breed of Applications. Auerbach Publications, Taylor & Francis Group, Boca Raton; New York. ISBN: 0-8493-7255-0, 2007.

[LPL09] Sangkeun Lee, Sungchan Park and Sang-goo Lee: A Study on Issues in Context-Aware Systems Based on a Survey and Service Scenarios. Proceedings of Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing, 2009. SNPD '09. 10th ACIS International Conference on, pp.8-13, 27-29 May 2009, IEEE Computer Society Washington, DC, USA .

[SB05] Quan Z. Sheng and Boualem Benatallah: ContextUML: A UML-Based Modeling Language for Model-Driven Development of Context-Aware Web Services, The 4th International Conference on Mobile Business (ICMB'05), IEEE Computer Society, Sydney, Australia. 2005.

[SB08] Context-aware Computing: a Survey Preparing a Generalized Approach. Robert Schmohl, Uwe Baumgarten. Proceedings of the International MultiConference of Engineers and Computers Scientist 2008 Vol I. IMECS 2008.

[STW93] Bill N. Schilit , Marvin M. Theimer and Brent B. Welch. Customizing mobile applications. Proceeding of USENIX Mobile & Location-Independent Computing Symposium, pages 129-138, Cambridge, Massaschusetts, 1993.

[VB94] Geoffrey M. Voelker and Brian N. Bershad. Mobisaic, an information system for a mobilw wireless computing environment. Proceedings of IEE Workshop on Mobile Computing System and Applications, pages 185-190, Santa Cruz, California. 1994. IEEE Computer Society Press.

[WZT+01] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology Based Context Modeling and Reasoning using OWL. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, page 18--. Washington, DC, USA, IEEE Computer Society, 2004.

[WGB] M. Weiser, R. Gold, J. S. Brown. The Computer for the 21st Century.

[YKW+02] Yau S.S., Karim F., Wang Y., Wang B., Gupsta S.K.S. Reconfigurable context-sensitive middleware for pervasive computing. 2002.] Jacob C., Linner D., Radusch I., Steglich S. Loosely coupled and context-aware service provision incorporating the quality of rules. Proceedings of the 2007 International Conference on Internet Computing. CSREA Press. 2007.